aws

Aurora のユーザーガイド

Amazon Aurora

Amazon Aurora: Aurora のユーザーガイド

Table of Contents

Amazon Aurora とは

Amazon Aurora (Aurora) は完全マネージド型のリレーショナルデータベースエンジンで、MySQL および PostgreSQL と互換性があります。MySQL と PostgreSQL が、ハイエンドの商用データベースのスピードおよび信頼性と、オープンソースデータベースのシンプルさとコスト効率を併せ持っていることは既にご存じでしょう。既存の MySQL および PostgreSQL データベースで現在使用しているコード、ツール、アプリケーションを Aurora でも使用できます。Aurora では、既存のアプリケーションのほとんどを変更せずに、ほんの少しのワークロードで MySQL のスループットの 5 倍、PostgreSQL のスループットの 3 倍を実現します。

Aurora には、高性能のストレージサブシステムが含まれています。MySQL と PostgreSQL との互換性のあるデータベースエンジンは、その高速分散ストレージを利用するようにカスタマイズされています。基本ストレージは、必要に応じて自動的に拡張されます。Aurora クラスターボリュームは、最大 128 tebibytes (TiB) のサイズまで増やすことができます。また、Aurora はデータベースのクラスター化とレプリケーションを自動化しスタンダード化します。通常、これらはデータベースの設定と管理に伴う最も困難な作業に属します。

Aurora は、マネージド型データベースサービスである Amazon Relational Database Service (Amazon RDS) の一部です。Amazon RDS を使用して、クラウドでリレーショナルデータベースをセットアップ、運用、スケーリングできます。Amazon RDS にまだ精通していない場合は、『Amazon Relational Database Service ユーザーガイド』を参照してください。Amazon Web Services で利用できるさまざまなデータベースオプションの詳細については、「AWS で組織に適したデータベースを選択する」を参照してください。

トピック

- Amazon RDS 責任共有モデル
- Amazon Aurora と Amazon RDS の連携
- Amazon Aurora DB クラスター
- Amazon Aurora バージョン
- リージョンとアベイラビリティーゾーン
- AWS リージョン と Aurora DB エンジンにより Amazon Aurora でサポートされている機能
- Amazon Aurora 接続管理
- Aurora DB インスタンスクラス
- Amazon Aurora ストレージと信頼性

- Amazon Aurora のセキュリティ
- Amazon Aurora の高可用性
- Amazon Aurora でのレプリケーション
- Aurora 向け DB インスタンスの請求

Amazon RDS 責任共有モデル

Amazon RDS は、DB インスタンスと DB クラスターのソフトウェアコンポーネントとインフラストラクチャをホストする責任があります。ユーザーはクエリチューニングに責任があります。これは、SQL クエリを調整してパフォーマンスを向上させるプロセスです。クエリのパフォーマンスは、データベースの設計、データサイズ、データ分布、アプリケーションのワークロード、クエリパターンに大きく依存しますが、これらは大きく異なる可能性があります。モニタリングとチューニングは、RDS データベースごとに高度に個別化されたプロセスです。Amazon RDS Performance Insights やその他のツールを使用して、問題のあるクエリを特定できます。

Amazon Aurora と Amazon RDS の連携

以下では、Amazon Aurora と Amazon RDS で利用可能なスタンダードの MySQL エンジンと PostgreSQL エンジンとの関係を示しています。

- Amazon RDS を通じて新しいデータベースサーバーをセットアップする際には、DB エンジンのオプションとして Aurora MySQL または Aurora PostgreSQL を選択します。
- Aurora には、使い慣れた Amazon Relational Database Service (Amazon RDS) の機能を管理に利用できるという利点があります。Aurora では、Amazon RDS AWS Management Console インターフェイス、AWS CLI コマンド、および API オペレーションを使用して、プロビジョニング、パッチ適用、バックアップ、復旧、障害検出、修復などのルーチンデータベースタスクを処理します。
- Aurora 管理オペレーションには、通常、個々のデータベースインスタンスではなく、レプリケーションによって同期化されたデータベースサーバーのクラスター全体が含まれます 自動クラスタリング、レプリケーション、およびストレージの割り当てにより、MySQL と PostgreSQL の最大規模のデプロイを簡単に、コスト効率よく設定、操作、スケーリングすることができます。
- スナップショットの作成とリストア、または一方向レプリケーションの設定により、Amazon RDS for MySQL と Amazon RDS for PostgreSQL から Aurora にデータを持ち込むことができます。押しボタン式の移行ツールを使用すると、既存の RDS for MySQL および RDS for PostgreSQL アプリケーションを Aurora に変換できます。

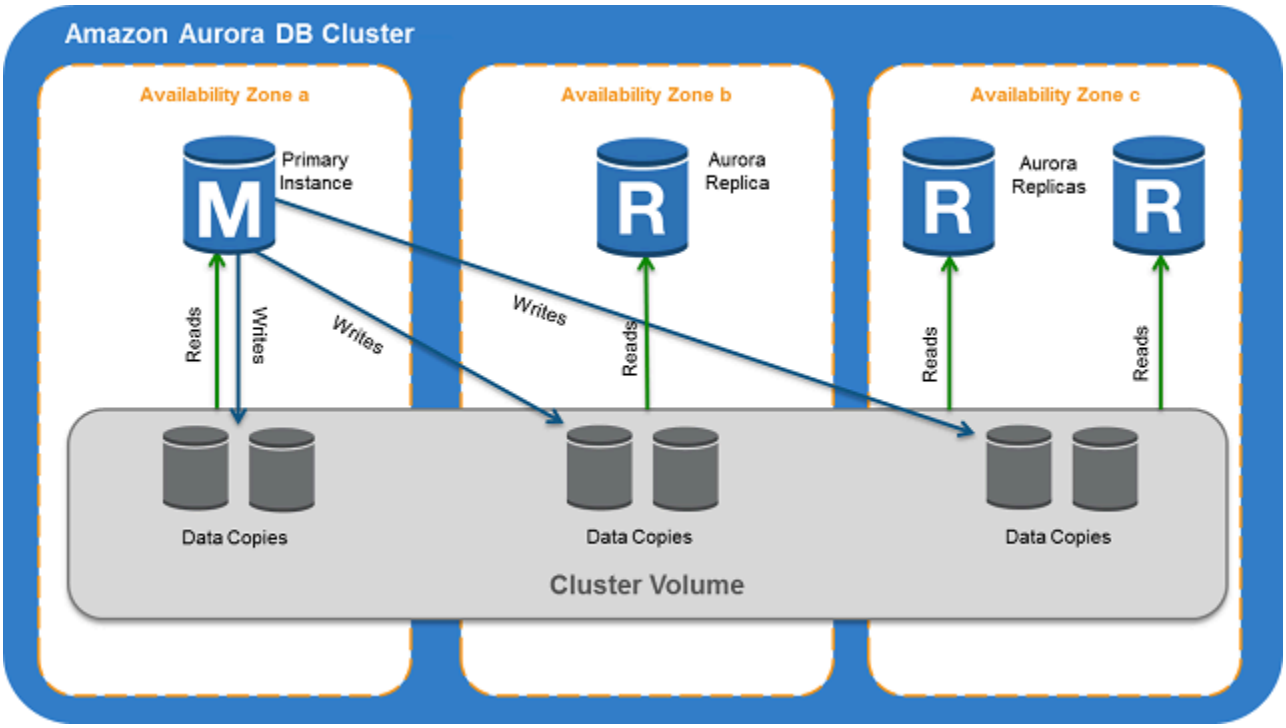
Amazon Aurora を使用する前に、「Amazon Aurora の環境をセットアップする」の手順を完了してから、「Amazon Aurora DB クラスター」で Aurora の概念と機能を確認してください。

Amazon Aurora DB クラスター

Amazon Aurora DB クラスターは、1 つ以上の DB インスタンスと、これらの DB インスタンスのデータを管理する 1 つのクラスターボリュームで構成されます。Aurora クラスターボリュームは、複数のアベイラビリティーゾーンにまたがる仮想データベースストレージボリュームです。各アベイラビリティーゾーンには DB クラスターデータのコピーが保存されます。Aurora DB クラスターは 2 種類の DB インスタンスで構成されます。

- プライマリ DB インスタンス - 読み書きオペレーションをサポートし、クラスターボリュームに対するすべてのデータ変更を実行します。各 Aurora DB クラスターには 1 つのプライマリ DB インスタンスがあります。
- Aurora レプリカ - プライマリ DB インスタンスと同じストレージボリュームに接続し、読み取りオペレーションのみをサポートします。各 Aurora DB クラスターは、プライマリ DB インスタンスに加えて 15 Aurora までのレプリカを持つことができます。Aurora レプリカを複数のアベイラビリティーゾーンに配置することで、高可用性を維持します。Aurora は、プライマリ DB インスタンスが使用できなくなった場合に自動的に Aurora レプリカにフェイルオーバーします。Aurora レプリカのフェイルオーバー優先順位を指定することができます。また、Aurora レプリカは、プライマリ DB インスタンスから読み取りワークロードをオフロードします。

次の図は、Aurora DB クラスター内のクラスターボリューム、プライマリ DB インスタンス、および Aurora レプリカの関係を示しています。



> **Note**
>
> これらの情報は、プロビジョンドクラスター、パラレルクエリクラスター、グローバルデータベースクラスター、Aurora Serverless クラスター、および MySQL 8.0 互換、5.7 互換、PostgreSQL 互換のすべてのクラスターに適用されます。

Aurora クラスターは、コンピューティング容量とストレージの分離を示しています。例えば、基になるストレージボリュームには複数のアベイラビリティーゾーン (AZ) に分散された複数のストレージノードが含まれるため、DB インスタンスが 1 つのみの Aurora 設定は依然としてクラスターです。

Aurora DB クラスターの入出力 (I/O) オペレーションは、ライター DB インスタンスとリーダー DB インスタンスのどちらで行われているかに関係なく、同じ方法でカウントされます。詳細については、「Amazon Aurora DB クラスターのストレージ設定」を参照してください。

Amazon Aurora バージョン

Amazon Aurora はコードを再利用することで、基盤となる MySQL および PostgreSQL DB エンジンとの互換性を維持しています。ただし Aurora では、独自のバージョン番号、リリースサイクル、バージョン廃止のタイムラインなどを使用します。以下のセクションでは、共通点とその相違点について説明します。この情報は、選択するバージョンの決定や、各バージョンで利用できる機能や修正プログラムを確認する方法の決定に役立ちます。また、アップグレードの頻度やアップグレードプロセスの計画方法を決定する際にも有用です。

トピック

Aurora で利用可能なリレーショナルデータベース

Aurora では、次のリレーショナルデータベースを使用できます。

- Amazon Aurora MySQL 互換エディション 使用に関する情報については、「Amazon Aurora MySQL の操作」を参照してください。利用可能なバージョンの詳細なリストについては、「Amazon Aurora MySQL のデータベースエンジンの更新」を参照してください。
- Amazon Aurora PostgreSQL 互換エディション 使用に関する情報については、「Amazon Aurora PostgreSQL の操作」を参照してください。利用可能なバージョンの詳細なリストについては、「Amazon Aurora PostgreSQL の更新」を参照してください。

コミュニティデータベースと Aurora 間でのバージョン番号の違い

Amazon Aurora の各バージョンは、MySQL または PostgreSQL の特定のコミュニティデータベースバージョンと互換性があります。データベースのコミュニティバージョンを確認するには `version` 関数を使用し、Aurora バージョンを確認するには `aurora_version` 関数を使用します。

Aurora MySQL および Aurora PostgreSQL に関する例を以下に示します。

```
mysql> select version();
+------------------+
|   version()      |
+------------------+
|  5.7.12          |
+------------------+

mysql> select aurora_version(), @@aurora_version;
+------------------+------------------+
| aurora_version() | @@aurora_version |
+------------------+------------------+
| 2.08.1           | 2.08.1           |
+------------------+------------------+
```

```
postgres=> select version();
-------------------------------------------------------------------------------
PostgreSQL 11.7 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.9.3, 64-bit
(1 row)

postgres=> select aurora_version();
aurora_version
----------------
3.2.2
```

詳細については、SQL を使用した Aurora MySQL バージョンの確認およびAmazon Aurora PostgreSQL のバージョンの識別を参照してください。

Amazon Aurora メジャーバージョン

Aurora バージョンでは、*major*.*minor*.*patch* スキームを使用します。Aurora メジャーバージョンは、Aurora と互換性のある MySQL または PostgreSQL コミュニティ版のメジャーバージョンを指します。Aurora MySQL および Aurora PostgreSQL メジャーバージョンは、少なくとも対応するコミュニティバージョンのコミュニティが終了するまでの期間、標準サポートの対象です。Aurora の標準サポート終了日を過ぎてもメジャーバージョンは有料で引き続き使用できます。詳細については、Amazon RDS 延長サポートの使用 と 「Amazon Aurora 料金表」を参照してください。

Amazon は、Aurora バージョンのサポートを当初発表よりも長く延長した場合、新しい日付を反映してこの表を更新します。

コミュニティメジャーバージョン	Aurora メジャーバージョン	コミュニティサポート終了日	Aurora 標準サポート終了日	RDS 延長サポート開始 1 年目の価格設定日	RDS 延長サポート開始 3 年目の価格設定日	RDS 延長サポートの終了日	RDS 延長サポートの対象となるマイナーバージョン
MySQL 5.6 (廃止)	Aurora MySQL バージョン 1 (廃止)	2021 年 2 月 5 日	2023 年 2 月 28 日	該当なし	該当なし	該当なし	該当なし
MySQL 5.7	Aurora MySQL バージョン 2	2023 年 10 月	2024 年 10 月 31 日	2024 年 12 月 1 日	該当なし	2027 年 2 月 28 日	Aurora MySQL 2.11、2.12
MySQL 8.0	Aurora MySQL バージョン 3	2026 年 4 月	2027 年 4 月 30 日	2027 年 5 月 1 日	該当なし	2029 年 7 月 31 日	今後決定

コミュニティメジャーバージョン	Aurora メジャーバージョン	コミュニティサポート終了日	Aurora 標準サポート終了日	RDS 延長サポート開始 1 年目の価格設定日	RDS 延長サポート開始 3 年目の価格設定日	RDS 延長サポートの終了日	RDS 延長サポートの対象となるマイナーバージョン
PostgreSQL 9.6 (廃止)	Aurora PostgreSQL 1 (廃止)	2021 年 11 月 11 日	2022 年 1 月 31 日	該当なし	該当なし	該当なし	該当なし

コミュニティメジャーバージョン	Aurora メジャーバージョン	コミュニティサポート終了日	Aurora 標準サポート終了日	RDS 延長サポート開始 1 年目の価格設定日	RDS 延長サポート開始 3 年目の価格設定日	RDS 延長サポートの終了日	RDS 延長サポートの対象となるマイナーバージョン
PostgreSQL 10 (廃止)	Aurora PostgreSQL 2 (廃止) PostgreSQL 10.17 以前のバージョンにのみ適用されます。10.18 以降のバージョンでは、Aurora バージョンは、PostgreSQL コミュニティバージョンの *major*.*min* バージョンと同じですが、*patch* の位置に 3 桁目	2022 年 11 月 10 日	2023 年 1 月 31 日	該当なし	該当なし	該当なし	該当なし

コミュニティメジャーバージョン	Aurora メジャーバージョン	コミュニティサポート終了日	Aurora 標準サポート終了日	RDS 延長サポート開始 1 年目の価格設定日	RDS 延長サポート開始 3 年目の価格設定日	RDS 延長サポートの終了日	RDS 延長サポートの対象となるマイナーバージョン
	があります。						

コミュニティメジャーバージョン	Aurora メジャーバージョン	コミュニティサポート終了日	Aurora 標準サポート終了日	RDS 延長サポート開始 1 年目の価格設定日	RDS 延長サポート開始 3 年目の価格設定日	RDS 延長サポートの終了日	RDS 延長サポートの対象となるマイナーバージョン
PostgreSQL 11	Aurora PostgreSQL 3。PostgreSQL 11.12 以前のバージョンにのみ適用されます。11.13 以降のバージョンでは、Aurora バージョンは、PostgreSQL コミュニティバージョンの *major*.*min* バージョンと同じですが、*patch* の場所に 3 桁目	2023 年 11 月	2024 年 2 月 29 日	2024 年 4 月 1 日	2026 年 4 月 1 日	2027 年 3 月 31 日	Aurora PostgreSQL 11.9、11.21

コミュニティメジャーバージョン	Aurora メジャーバージョン	コミュニティサポート終了日	Aurora 標準サポート終了日	RDS 延長サポート開始 1 年目の価格設定日	RDS 延長サポート開始 3 年目の価格設定日	RDS 延長サポートの終了日	RDS 延長サポートの対象となるマイナーバージョン
	を加えます。						

コミュニティメジャーバージョン	Aurora メジャーバージョン	コミュニティサポート終了日	Aurora 標準サポート終了日	RDS 延長サポート開始 1 年目の価格設定日	RDS 延長サポート開始 3 年目の価格設定日	RDS 延長サポートの終了日	RDS 延長サポートの対象となるマイナーバージョン
PostgreSQL 12	Aurora PostgreSQL 4。PostgreSQL 12.7 以前のバージョンにのみ適用されます。12.8 以降のバージョンでは、Aurora バージョンは、PostgreSQL コミュニティバージョンの *major*.*min* バージョンと同じですが、*patch* の場所に 3 桁目	2024 年 11 月	2025 年 2 月 28 日	2025 年 3 月 1 日	2027 年 3 月 1 日	2028 年 2 月 29 日	今後決定

コミュニティメジャーバージョン	Aurora メジャーバージョン	コミュニティサポート終了日	Aurora 標準サポート終了日	RDS 延長サポート開始 1 年目の価格設定日	RDS 延長サポート開始 3 年目の価格設定日	RDS 延長サポートの終了日	RDS 延長サポートの対象となるマイナーバージョン
	を加えます。						

コミュニティメジャーバージョン	Aurora メジャーバージョン	コミュニティサポート終了日	Aurora 標準サポート終了日	RDS 延長サポート開始 1 年目の価格設定日	RDS 延長サポート開始 3 年目の価格設定日	RDS 延長サポートの終了日	RDS 延長サポートの対象となるマイナーバージョン
PostgreSQL 13	Aurora PostgreSQL 13。13.3 以降のバージョンでは、Aurora バージョンは、PostgreSQL コミュニティバージョンの *major.min* バージョンと同じですが、Aurora のパッチがリリースされると、*patch* の位置に 3 桁目が追加されます。	2025 年 11 月	2026 年 2 月 28 日	2026 年 3 月 1 日	2028 年 3 月 1 日	2029 年 2 月 28 日	今後決定

コミュニティメジャーバージョン	Aurora メジャーバージョン	コミュニティサポート終了日	Aurora 標準サポート終了日	RDS 延長サポート開始 1 年目の価格設定日	RDS 延長サポート開始 3 年目の価格設定日	RDS 延長サポートの終了日	RDS 延長サポートの対象となるマイナーバージョン
PostgreSQL 14	Aurora PostgreSQL 14.3 以降。Aurora へのパッチがリリースされたとき、Aurora バージョンは、PostgreSQL コミュニティバージョンの *major.min* バージョンと同じですが、*patch* の場所に 3 桁目を加えます。	2026 年 11 月	2027 年 2 月 28 日	2027 年 3 月 1 日	2029 年 3 月 1 日	2030 年 2 月 28 日	今後決定

コミュニティメジャーバージョン	Aurora メジャーバージョン	コミュニティサポート終了日	Aurora 標準サポート終了日	RDS 延長サポート開始 1 年目の価格設定日	RDS 延長サポート開始 3 年目の価格設定日	RDS 延長サポートの終了日	RDS 延長サポートの対象となるマイナーバージョン
PostgreSQL 15	Aurora PostgreSQL 15.2 以降。Aurora へのパッチがリリースされたとき、Aurora バージョンは、PostgreSQL コミュニティバージョンの *major*.*min* バージョンと同じですが、*patch* の場所に 3 桁目を加えます。	2027 年 11 月	2028 年 2 月 29 日	2028 年 3 月 1 日	2030 年 3 月 1 日	2031 年 2 月 28 日	今後決定

コミュニティメジャーバージョン	Aurora メジャーバージョン	コミュニティサポート終了日	Aurora 標準サポート終了日	RDS 延長サポート開始 1 年目の価格設定日	RDS 延長サポート開始 3 年目の価格設定日	RDS 延長サポートの終了日	RDS 延長サポートの対象となるマイナーバージョン
PostgreSQL 16	Aurora PostgreSQL 16.1 以上。Aurora へのパッチがリリースされたとき、Aurora バージョンは、PostgreSQL コミュニティバージョンの *major.min* バージョンと同じですが、*patch* の場所に 3 桁目を加えます。	2028 年 11 月 9 日	2029 年 2 月 28 日	今後決定	今後決定	今後決定	今後決定

> **Note**
>
> Amazon RDS の Aurora MySQL バージョン 2 の延長サポートは 2024 年 11 月 1 日に開始されますが、2024 年 12 月 1 日まで課金されません。2024 年 11 月 1 日から 11 月 30 日までの間、すべての Aurora MySQL バージョン 2 DB クラスターは Amazon RDS 延長サポートの対象となります。
> Amazon RDS の PostgreSQL 11 の延長サポートは 2024 年 3 月 1 日に開始されますが、2024 年 4 月 1 日まで課金されません。2024 年 3 月 1 日から 3 月 31 日までの間、Aurora PostgreSQL バージョン 11 DB クラスターはすべて Amazon RDS 延長サポートの対象となります。

Amazon Aurora マイナーバージョン

Aurora バージョンでは、*major*.*minor*.*patch* スキームを使用します。Aurora マイナーバージョンは、新機能や修正など、コミュニティと Aurora のサービス固有の改善を段階的に提供します。

Amazon Aurora は、現在、以下の MySQL マイナーバージョンをサポートしています。

> **Note**
>
> マイナーバージョンでは Amazon RDS 延長サポートはご利用いただけません。

Aurora MySQL バージョン	Aurora MySQL リリース日	Aurora MySQL 標準サポート終了日
3.06 (Community MySQL 8.0.34 との互換性)	2024 年 3 月 7 日	2025 年 5 月 31 日
3.05 (Community MySQL 8.0.32 との互換性)	2023 年 10 月 25 日	2025 年 1 月 31 日
3.04[1] (Community MySQL 8.0.28 との互換性)	2023 年 7 月 31 日	2026 年 10 月 31 日
3.03 (Community MySQL 8.0.26 との互換性)	2023 年 3 月 1 日	2024 年 8 月 15 日

Aurora MySQL バージョン	Aurora MySQL リリース日	Aurora MySQL 標準サポート終了日
3.02 (Community MySQL 8.0.23 との互換性)	2022 年 4 月 20 日	2024 年 1 月 15 日
3.01 (Community MySQL 8.0.23 との互換性)	2021 年 11 月 18 日	2024 年 1 月 15 日
2.12[2] (Community MySQL 5.7.40 または 5.7.44[3] と互換)	2023 年 7 月 25 日	2024 年 10 月 31 日
2.11[2] (Community MySQL 5.7.12 との互換性)	2022 年 10 月 25 日	2024 年 10 月 31 日
2.07 (Community MySQL 5.7.12 との互換性)	2019 年 11 月 25 日	2024 年 4 月 30 日

[1] Aurora MySQL 長期サポート (LTS) バージョン 詳細については、「Aurora MySQL 長期サポート (LTS) リリース」を参照してください。

[2] このマイナーバージョンは、メジャーバージョンが Amazon RDS 延長サポート期間中でも引き続き利用できます。詳細については、「Amazon Aurora メジャーバージョン」を参照してください。

[3] Aurora MySQL バージョン 2.12 ~ 2.12.1 は MySQL バージョン 5.7.40 と、Aurora MySQL バージョン 2.12.2 以降は MySQL version 5.7.44 と互換性があります。

Amazon Aurora パッチバージョン

Aurora バージョンでは、*major*.*minor*.*patch* スキームを使用します。Aurora パッチバージョンには、初期リリース後にマイナーバージョンに追加された重要な修正 (例えば、Aurora MySQL 2.10.0、2.10.1、...、2.10.3 など) が含まれています。それぞれの新しいマイナーバージョンは、新しい Aurora 機能を提供するものですが、特定のマイナーバージョン内の新しいパッチバージョンは、主に重要な問題を修正するために提供されています。

パッチ適用の詳細については、「Amazon Aurora DB クラスターのメンテナンス」を参照してください。

Amazon Aurora の各バージョンの新機能

新しい Aurora の各バージョンには、そのバージョンに適用される新機能、修正、その他の機能強化などが記載されたリリースノートが付属しています。

Aurora MySQL リリースノートについては、「Aurora MySQL のリリースノート」を参照してください。Aurora PostgreSQL リリースノートについては、「Aurora PostgreSQL のリリースノート」を参照してください。

データベースクラスターのための Amazon Aurora データベースバージョンの指定

AWS Management Console、AWS CLI、または `CreateDBCluster` API の、[データベースを作成する] オペレーションを使用して新しい DB クラスターを作成する際に、現在利用可能な (メジャーおよびマイナーの) バージョンを指定できます。すべての Aurora データベースバージョンが、すべての AWS リージョンで利用可能な訳ではありません。

Aurora クラスターを作成する方法については、「Amazon Aurora DB クラスターの作成」を参照してください。既存の Aurora クラスターのバージョンを変更する方法については、「Amazon Aurora DB クラスターの変更」を参照してください。

Amazon Aurora のデフォルトバージョン

新しい Aurora マイナーバージョンに、以前のバージョンと比較して大幅な改善が含まれている場合、このバージョンが、新しい DB クラスターのデフォルトとしてマークされます。通常、メジャーバージョンごとに 2 つのデフォルトバージョンが、年ごとにリリースされます。

セキュリティと機能に関する最新の修正を利用するためには、DB クラスターを最新のデフォルトマイナーバージョンに維持することをお勧めします。

マイナーバージョンの自動アップグレード

Aurora マイナーバージョンを最新の状態に保つには、Aurora クラスターの各 DB インスタンスに対して、[自動マイナーバージョンアップグレード] をオンにします。Aurora は、クラスター内のすべての DB インスタンスでこの設定が有効になっている場合にのみ、自動アップグレードを実行します。マイナーバージョンの自動アップグレードは、デフォルトのマイナーバージョンについて実行されます。

通常、[自動マイナーバージョンアップグレード] に `Yes` が設定されている DB クラスターに対して、自動アップグレードが年に 2 回スケジュールされます。これらのアップグレードは、クラス

ターに指定したメンテナンス期間中に実施されます。詳細については、「Aurora DB クラスターのマイナーバージョン自動アップグレード」を参照してください。

マイナーバージョンの自動アップグレードは、カテゴリが maintenance で ID が RDS-EVENT-0156 の Amazon RDS DB クラスターイベントを通じて事前に通知されます。詳細については、「Amazon RDS のイベントカテゴリとイベントメッセージ」を参照してください。

Amazon Aurora メジャーバージョンが利用可能な期間

Amazon Aurora メジャーバージョンは、少なくとも対応するコミュニティバージョンのコミュニティが終了するまでの期間利用可能です。Aurora 標準サポートの終了日を使って、テストとアップグレードサイクルを計画することができます。これらの日付は、新しいバージョンへのアップグレードが必要になる可能性がある最も早い日付を表します。詳細については、「Amazon Aurora メジャーバージョン」を参照してください。

当社から新しいメジャーバージョンへのアップグレードをお願いする前に、計画策定に役立てていただくために、当社からは、少なくとも 12 ヶ月前にリマインダーを発信します。これは、詳細なアップグレードプロセスを通知するために送られます。詳細には、特定のマイルストーンのタイミング、DB クラスターへの影響、推奨されるアクションが含まれます。メジャーバージョンアップグレードを実行する前に、新しいデータベースバージョンに関するアプリケーションのテストを、完全に完了することをお勧めします。

メジャーバージョンの Aurora 標準サポートが終了すると、古いバージョンを実行している DB クラスターは、スケジュールされたメンテナンスウィンドウ中に自動的に延長サポートバージョンにアップグレードされます。延長サポート料金が適用される場合があります。Amazon RDS の延長サポートの詳細については、「Amazon RDS 延長サポートの使用」を参照してください。

Amazon Aurora マイナーバージョンがリリースされる頻度

通常、Amazon Aurora マイナーバージョンは四半期ごとにリリースされます。このリリーススケジュールは、追加の機能または修正を取り入れる目的で、変更される場合があります。

Amazon Aurora マイナーバージョンが利用可能な期間

当社では、特定のメジャーバージョン用の各 Amazon Aurora マイナーバージョンは、少なくとも 12 か月間利用可能にしています。この期間が終了すると、Aurora は、後続のデフォルトのマイナーバージョンに関し、自動マイナーバージョンアップグレードを適用する可能性があります。このアップグレードは、古いマイナーバージョンを実行しているクラスターで、スケジュールされたメンテナンスウィンドウ中にスタートされます。

セキュリティの問題などの重大な問題がある場合、またはメジャーバージョンの使用期間が終了した場合は、特定のメジャーバージョンのマイナーバージョンを、通常の 12 か月よりも早い時点で置き換えることがあります。

製品寿命が近づいているマイナーバージョンの自動アップグレードをスタートする場合は、通常 3 か月前にリマインダーを送信します。これは、詳細なアップグレードプロセスを通知するために送られます。詳細には、特定のマイルストーンのタイミング、DB クラスターへの影響、推奨されるアクションが含まれます。3 か月未満の通知は、セキュリティ上の問題など、より迅速な対応が必要な重大な問題がある場合に使用されます。

[マイナーバージョン自動アップグレード] 設定を有効にしていない場合、リマインダーは表示されますが、RDS イベントの通知は表示されません。アップグレードは、必須のアップグレード期限が過ぎた後のメンテナンス期間内に行われます。

[マイナーバージョン自動アップグレード] 設定を有効にしている場合、リマインダーとカテゴリーが `maintenance`、ID が `RDS-EVENT-0156` の Amazon RDS DB クラスターイベントが表示されます。アップグレードは次のメンテナンスウィンドウ中に行われます。

マイナーバージョン自動アップグレードの詳細については、「Aurora DB クラスターのマイナーバージョン自動アップグレード」を参照してください。

選択された Amazon Aurora マイナーバージョンの長期サポート

Aurora メジャーバージョンごとに、長期サポート (LTS) バージョンとして指定された特定のマイナーバージョンがあり、これらは、少なくとも 3 年間の利用が可能です。つまり、メジャーバージョンごとに少なくとも 1 つのマイナーバージョンが、通常の 12 か月よりも長く利用可能になります。通常、この期間の終了の 6 か月前にリマインダーを送信します。これは、詳細なアップグレードプロセスを通知するために送られます。詳細には、特定のマイルストーンのタイミング、DB クラスターへの影響、推奨されるアクションが含まれます。6 か月未満の通知は、セキュリティ上の問題など、より迅速な対応が必要な重大な問題がある場合に使用されます。

LTS マイナーバージョンには、(パッチバージョンによる) 重大な修正のみが含まれています。LTS バージョンには、導入後にリリースされた新機能は含まれていません。LTS マイナーバージョンで実行されている DB クラスターには、1 年に 1 回、LTS リリース用の最新パッチバージョンによるパッチ修正が行われます。このパッチ手法は、セキュリティと安定性に関する累積的な修正からのメリットを確実に適用するためのものです。セキュリティ関連など、適用すべき重要な修正が存在する場合には、より頻繁なパッチ修正を、LTS マイナーバージョンに対し行うことがあります。

> **Note**
>
> ライフサイクルの期間中、LTS マイナーバージョンを使用し続けたい場合は、DB インスタンスの [自動マイナーバージョンアップグレード] をオフにしてください。DB クラスターが LTS マイナーバージョンから自動的にアップグレードされないようにするには、Aurora クラスターの任意の DB インスタンスで、[自動マイナーバージョンアップグレード] に No を設定します。

すべての Aurora LTS バージョンのバージョン番号については、「Aurora MySQL 長期サポート (LTS) リリース」および「Aurora PostgreSQL 長期サポート (LTS) リリース」を参照してください。

選択した Aurora バージョンの Amazon RDS 延長サポート

Amazon RDS 延長サポートを利用すると、Aurora 標準サポート終了日以降も、データベースを以前のメジャーエンジンバージョンで引き続き実行できます (追加料金がかかります)。RDS 延長サポート期間中、Amazon RDS は、国家脆弱性データベース (NVD) の CVSS 重大度評価で定義されている重大および高 CVE のパッチを提供します。詳細については、「Amazon RDS 延長サポートの使用」を参照してください。

RDS 延長サポートは、特定の Aurora バージョンでのみ利用できます。詳細については、「Amazon Aurora メジャーバージョン」を参照してください。

データベースクラスターが新しいバージョンにアップグレードされるかどうか、および、そのタイミングを手動で制御する

マイナーバージョンの自動アップグレードは、デフォルトのマイナーバージョンについて実行されます。通常、[自動マイナーバージョンアップグレード] に Yes が設定されている DB クラスターに対して、自動アップグレードが年に 2 回スケジュールされます。これらのアップグレードは、お客様が指定したメンテナンスウィンドウ中にスタートされます。自動マイナーバージョンアップグレードをオフにするには、Aurora クラスター内の任意の DB インスタンスで、[自動マイナーバージョンアップグレード] に No を設定します。Aurora は、クラスター内のすべての DB インスタンスで設定が有効になっている場合にのみ、マイナーバージョンの自動アップグレードを実行します。

メジャーバージョンのアップグレードは互換性のリスクがあるため、自動では実行されません。前述の、メジャーバージョン廃止の場合を除き、これらの作業は、お客様にスタートしていただく必要が

あります。メジャーバージョンアップグレードを実行する前に、新しいデータベースバージョンに関するアプリケーションのテストを、完全に完了することをお勧めします。

DB クラスターを新しい Aurora メジャーバージョンにアップグレードする方法については、「Amazon Aurora MySQL DB クラスターのアップグレード」および「Amazon Aurora PostgreSQL DB クラスターのアップグレード」を参照してください。

Amazon Aurora の必須アップグレード

特定の重要な修正については、同じ LTS リリース内で、新しいパッチレベルへの管理アップグレードを実行する場合があります。これらの必須アップグレードは、[自動マイナーバージョンアップグレード] 設定がオフになっている場合でも実施されます。これを行う前に、アップグレードプロセスに関する詳細が当社から送られます。詳細には、特定のマイルストーンのタイミング、DB クラスターへの影響、推奨されるアクションが含まれます。これらの管理アップグレードは自動的に実行されます。各アップグレードは、クラスターのメンテナンスウィンドウ中にスタートされます。

アップグレードする前に新しい Aurora バージョンを使用して DB クラスターをテストする

アップグレードプロセスのテストや、新しいバージョンがアプリケーションとワークロードでどのように動作するかのテストを行うことができます。次のいずれかの方法を使用します。

- Amazon Aurora の高速データベースクローン機能を使用して、クラスターをクローンします。新しいクラスターについて、アップグレードプロセスと、アップグレード後のテストを実行します。
- クラスタースナップショットからの復元を利用して、新しい Aurora クラスターを作成します。クラスタースナップショットは、既存の Aurora クラスターから自分で作成できます。Aurora では、クラスターごとに、定期的なスナップショットも自動的に作成されます。その後、新しいクラスターに対するバージョンアップグレードをスタートします。元のクラスターをアップグレードするかどうかを決定する前に、アップグレードされたクラスターのコピーで試験することができます。

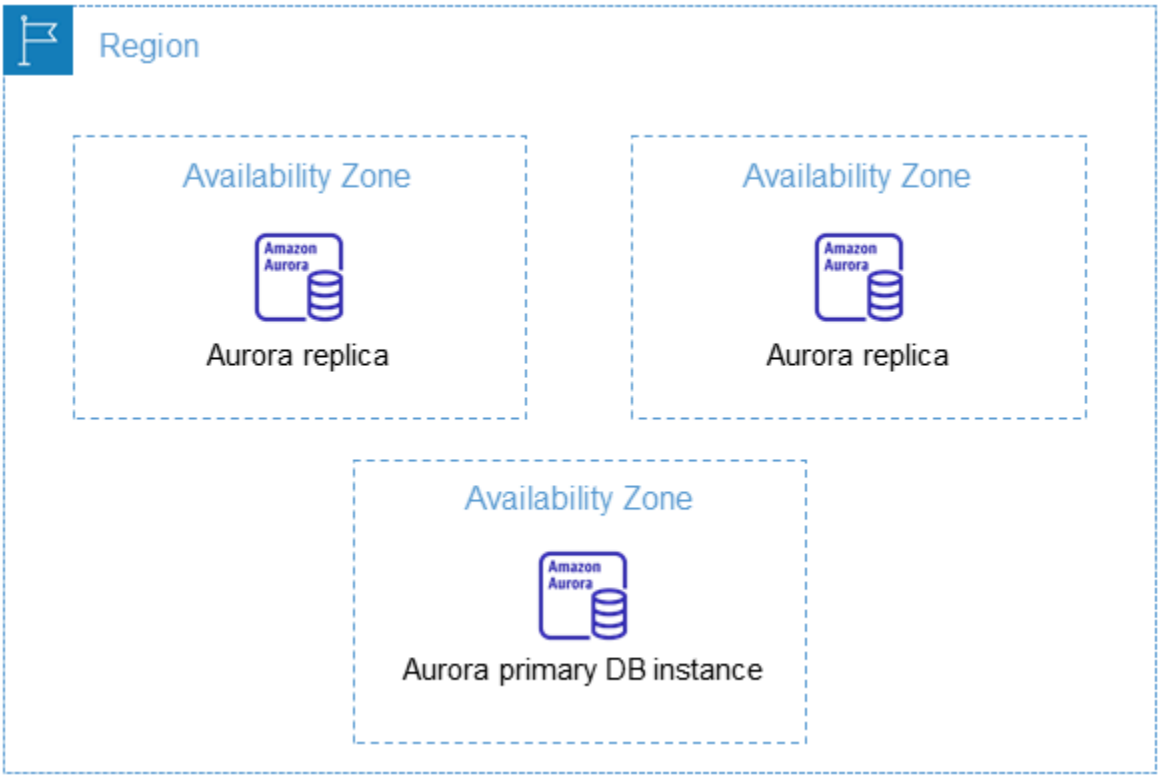
テスト用に新しいクラスターを作成するこれらの方法の詳細については、「Amazon Aurora DB クラスターのボリュームのクローン作成」および「DB クラスタースナップショットの作成」を参照してください。

リージョンとアベイラビリティーゾーン

Amazon クラウドコンピューティングリソースは、世界各地の多くの場所でホストされています。これらの場所は、AWS リージョンとアベイラビリティーゾーンから構成されています。AWS リージョンはそれぞれ、地理的に離れた領域です。AWS リージョンごとにアベイラビリティーゾーンと呼ばれる複数の独立した場所があります。

> **Note**
>
> AWS リージョンでのアベイラビリティーゾーンの検索については、Amazon EC2 ドキュメントの「Describe Your Availability Zones」(アベイラビリティーゾーンの詳細表示) を参照してください。

Amazon は、アベイラビリティーの高い最新のデータセンターを運用しています。ただし、非常にまれですが、同じ場所にある DB インスタンスすべての可用性に影響する障害が発生することもあります。すべての DB インスタンスを 1 か所でホストしている場合、そのような障害が起きた際に DB インスタンスがすべて利用できなくなります。



各 AWS リージョンは完全に独立していることを覚えておくことが重要です。スタートした Amazon RDS アクティビティ (例えば、データベースインスタンスの作成や使用可能なデータベースインスタンスの一覧表示など) は、現在のデフォルト AWSリージョンでのみ実行されます。デフォルトの AWS リージョンは、コンソールで変更するか、環境変数 `AWS_DEFAULT_REGION` を設定することにより変更できます。または、AWS Command Line Interface (AWS CLI) で `--region` パラメータを使用すると、リージョンを上書きできます。詳細については、「AWS Command Line Interface の設定」を参照し、特に環境可変とコマンドラインオプションのセクションに注目してください。

Amazon RDS では、AWS と呼ばれる特別な AWS GovCloud (US) リージョンをサポートしています。これらのリージョンは、米国政府機関および関係者が、より機密性の高いワークロードをクラウドに移行するができるように設計されたものです。AWS GovCloud (US) リージョンは、米国政府の特定の規制とコンプライアンスの要件に対応しています。詳細については、「AWS GovCloud (US) とは」を参照してください。

特定の AWS リージョンの Amazon RDS DB インスタンスを作成または操作するには、対応するリージョンのサービスエンドポイントを使用します。

> **Note**
>
> Aurora は Local Zones をサポートしていません。

AWS リージョン

各 AWS リージョンは、他の AWS リージョンと完全に分離されるように設計されています。この設計により、最大限の耐障害性と安定性が達成されます。

リソースを表示すると、指定した AWS リージョンに結び付けられているリソースのみが表示されます。これは、AWS リージョンが相互に分離されており、AWS リージョン間ではリソースが自動的にレプリケートされないためです。

利用可能なリージョン

コマンドラインインターフェイスまたは API オペレーションを使用して Aurora DB クラスターを操作する場合は、そのリージョンエンドポイントを指定する必要があります。

トピック

- Aurora MySQL が利用可能なリージョン
- Aurora PostgreSQL が利用可能なリージョン

Aurora MySQL が利用可能なリージョン

次の表は、現時点で Aurora MySQL を利用できる AWS リージョンとリージョン別のエンドポイントの一覧です。

リージョン名	リージョン	エンドポイント	プロトコル	
米国東部（オハイオ）	us-east-2	rds.us-east-2.amazonaws.com	HTTPS	
米国東部 (バージニア北部)	us-east-1	rds.us-east-1.amazonaws.com	HTTPS	
米国西部（北カリフォルニア）	us-west-1	rds.us-west-1.amazonaws.com	HTTPS	
米国西部（オレゴン）	us-west-2	rds.us-west-2.amazonaws.com	HTTPS	
アフリカ (ケープタウン)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS	
アジアパシフィック (香港)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS	
アジアパシフィック (ハイデラバード)	ap-south-2	rds.ap-south-2.amazonaws.com	HTTPS	

リージョン名	リージョン	エンドポイント	プロトコル	
アジアパシフィック (ジャカルタ)	ap-southeast-3	rds.ap-southeast-3.amazonaws.com	HTTPS	
アジアパシフィック (メルボルン)	ap-southeast-4	rds.ap-southeast-4.amazonaws.com	HTTPS	
アジアパシフィック (ムンバイ)	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS	
アジアパシフィック (大阪)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com	HTTPS	
アジアパシフィック (ソウル)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS	
アジアパシフィック (シンガポール)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS	
アジアパシフィック (シドニー)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS	

リージョン名	リージョン	エンドポイント	プロトコル	
アジアパシフィック (東京)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS	
カナダ (中部)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS	
カナダ西部 (カルガリー)	ca-west-1	rds.ca-west-1.amazonaws.com	HTTPS	
欧州 (フランクフルト)	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS	
欧州 (アイルランド)	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS	
欧州 (ロンドン)	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS	
欧州 (ミラノ)	eu-south-1	rds.eu-south-1.amazonaws.com	HTTPS	
欧州 (パリ)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS	
欧州 (スペイン)	eu-south-2	rds.eu-south-2.amazonaws.com	HTTPS	
欧州 (ストックホルム)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS	

リージョン名	リージョン	エンドポイント	プロトコル	
欧州 (チューリッヒ)	eu-central-2	rds.eu-central-2.amazonaws.com	HTTPS	
イスラエル (テルアビブ)	il-central-1	rds.il-central-1.amazonaws.com	HTTPS	
中東 (バーレーン)	me-south-1	rds.me-south-1.amazonaws.com	HTTPS	
中東 (アラブ首長国連邦)	me-central-1	rds.me-central-1.amazonaws.com	HTTPS	
南米 (サンパウロ)	sa-east-1	rds.sa-east-1.amazonaws.com	HTTPS	
AWS GovCloud (米国東部)	us-gov-east-1	rds.us-gov-east-1.amazonaws.com	HTTPS	
AWS GovCloud (米国西部)	us-gov-west-1	rds.us-gov-west-1.amazonaws.com	HTTPS	

Aurora PostgreSQL が利用可能なリージョン

次の表は、現時点で Aurora PostgreSQL を利用できる AWS リージョンとリージョン別のエンドポイントの一覧です。

リージョン名	リージョン	エンドポイント	プロトコル	
米国東部（オハイオ）	us-east-2	rds.us-east-2.amazonaws.com	HTTPS	
米国東部 (バージニア北部)	us-east-1	rds.us-east-1.amazonaws.com	HTTPS	
米国西部（北カリフォルニア）	us-west-1	rds.us-west-1.amazonaws.com	HTTPS	
米国西部（オレゴン）	us-west-2	rds.us-west-2.amazonaws.com	HTTPS	
アフリカ (ケープタウン)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS	
アジアパシフィック (香港)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS	
アジアパシフィック (ハイデラバード)	ap-south-2	rds.ap-south-2.amazonaws.com	HTTPS	
アジアパシフィック (ジャカルタ)	ap-southeast-3	rds.ap-southeast-3.amazonaws.com	HTTPS	

リージョン名	リージョン	エンドポイント	プロトコル	
アジアパシフィック (メルボルン)	ap-southeast-4	rds.ap-southeast-4.amazonaws.com	HTTPS	
アジアパシフィック (ムンバイ)	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS	
アジアパシフィック (大阪)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com	HTTPS	
アジアパシフィック (ソウル)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS	
アジアパシフィック (シンガポール)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS	
アジアパシフィック (シドニー)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS	
アジアパシフィック (東京)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS	
カナダ (中部)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS	

リージョン名	リージョン	エンドポイント	プロトコル	
カナダ西部 (カルガリー)	ca-west-1	rds.ca-west-1.amazonaws.com	HTTPS	
欧州 (フランクフルト)	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS	
欧州 (アイルランド)	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS	
欧州 (ロンドン)	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS	
欧州 (ミラノ)	eu-south-1	rds.eu-south-1.amazonaws.com	HTTPS	
欧州 (パリ)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS	
欧州 (スペイン)	eu-south-2	rds.eu-south-2.amazonaws.com	HTTPS	
欧州 (ストックホルム)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS	
欧州 (チューリッヒ)	eu-central-2	rds.eu-central-2.amazonaws.com	HTTPS	
イスラエル (テルアビブ)	il-central-1	rds.il-central-1.amazonaws.com	HTTPS	

リージョン名	リージョン	エンドポイント	プロトコル	
中東 (バーレーン)	me-south-1	rds.me-south-1.amazonaws.com	HTTPS	
中東 (アラブ首長国連邦)	me-central-1	rds.me-central-1.amazonaws.com	HTTPS	
南米 (サンパウロ)	sa-east-1	rds.sa-east-1.amazonaws.com	HTTPS	
AWS GovCloud (米国東部)	us-gov-east-1	rds.us-gov-east-1.amazonaws.com	HTTPS	
AWS GovCloud (米国西部)	us-gov-west-1	rds.us-gov-west-1.amazonaws.com	HTTPS	

アベイラビリティーゾーン

アベイラビリティーゾーンとは、特定の AWS リージョン 内の分離された口ケーションのことです。各リージョンには、リージョンで高可用性を実現するように設計された複数のアベイラビリティーゾーン (AZ) があります。AZ は、AWS リージョンコードとそれに続く文字識別子 (us-east-1a など) によって識別されます。デフォルトの VPC を使用せずに VPC とサブネットを作成する場合は、特定の AZ で各サブネットを定義します。Aurora DB クラスターを作成すると、Aurora は VPC の DB サブネットグループ内のサブネットの 1 つにプライマリインスタンスを作成します。これにより、そのインスタンスを Aurora が選択した特定の AZ に関連付けます。

各 Aurora DB クラスターは、DB サブネットグループの AZ から Aurora によって自動的に選択された 3 つの個別の AZ にストレージのコピーをホストします。クラスター内のすべての DB インスタンスは、これらの 3 つの AZ のいずれかに配置する必要があります。

クラスターに DB インスタンスを作成する際、AZ を指定しない場合、Aurora はそのインスタンスに適切な AZ を自動的に選択します。

次のように describe-availability-zones Amazon EC2 コマンドを使用して、アカウントで有効な、指定されたリージョン内のアベイラビリティーゾーンを記述します。

```
aws ec2 describe-availability-zones --region region-name
```

例えば、アカウントで有効になっている米国東部 (バージニア北部) リージョン (us-east-1) 内のアベイラビリティーゾーンを記述するには、次のコマンドを実行します。

```
aws ec2 describe-availability-zones --region us-east-1
```

クラスターの作成時またはクラスターへのインスタンスの追加時に AZ を指定する方法については、「DB クラスターのネットワークを設定する」を参照してください。

Amazon Aurora DB クラスターのローカルタイムゾーン

デフォルトでは、Amazon Aurora DB クラスターのタイムゾーンは協定世界時 (UTC) です。代わりに、DB クラスターのインスタンスのタイムゾーンをアプリケーションのローカルタイムゾーンに設定できます。

DB クラスターのローカルタイムゾーンを設定するには、タイムゾーンパラメータをサポートされている値のいずれかに設定します。このパラメータの値は、DB クラスターのクラスターパラメータグループで設定します。

- Aurora MySQL の場合、このパラメータの名前は`time_zone`です。`time_zone` パラメータの設定のベストプラクティスについては、「タイムスタンプ操作の最適化」を参照してください。
- Aurora PostgreSQL の場合、このパラメータの名前は`timezone`です。

DB クラスターのタイムゾーンパラメータを設定するときに、DB クラスターのすべてのインスタンスは、新しいローカルタイムゾーンを使用するように変更されます。他の Aurora DB クラスターが同じクラスターパラメータグループを使用している場合もあります。その場合は、それらの DB クラスター内のすべてのインスタンスも新しいローカルタイムゾーンを使用するように変更されます。クラスターレベルのパラメータの詳細については、「Amazon Aurora の DB クラスターパラメータと DB インスタンスパラメータ」を参照してください。

ローカルタイムゾーンを設定した後、データベースへのすべての新しい接続にその変更が反映されます。ローカルタイムゾーンを変更する際にデータベースへの接続を開いている場合もあります。その場合、その接続を閉じて新しい接続を開くまで、ローカルタイムゾーンの更新は表示されません。

AWS リージョン間でレプリケーションを実行する場合、レプリケーション元の DB クラスターとレプリカは、異なるパラメータグループを使用します。パラメータグループは、AWS リージョンに一意のものです。各インスタンスで同じローカルタイムゾーンを使用するには、レプリケーション元とレプリカの両方について、パラメータグループのタイムゾーンパラメータを設定する必要があります。

DB クラスタースナップショットから DB クラスターを復元すると、ローカルタイムゾーンが UTC に設定されます。復元が完了したら、タイムゾーンをローカルタイムゾーンに更新できます。DB クラスターを特定の時点の状態に復元する場合もあります。その場合、復元された DB クラスターのローカルタイムゾーンは、復元された DB クラスターのパラメータグループに設定されているタイムゾーンです。

次の表は、ローカルタイムゾーンを設定できる値の一部を示しています。使用可能なタイムゾーンをすべて一覧表示するには、次の SQL クエリを使用できます。

- Aurora MySQL: `select * from mysql.time_zone_name;`
- Aurora PostgreSQL: `select * from pg_timezone_names;`

> **Note**
>
> 表の注記のとおり、一部のタイムゾーンでは、特定の日付範囲の時間値が正しく報告さない場合があります。オーストラリアのタイムゾーンの場合、返されるタイムゾーンの略名は、表の注記のとおり古い値になります。

タイムゾーン	コメント
`Africa/Harare`	このタイムゾーン設定では、1903 年 2 月 28 日 21:49:40 GMT から 1903 年 2 月 28 日 21:55:48 GMT まで正しくない値を返す可能性があります。
`Africa/Monrovia`	

タイムゾーン	コメント
Africa/Nairobi	このタイムゾーン設定では、1939 年 12 月 31 日 21:30:00 GMT から 1959 年 12 月 31 日 21:15:15 GMT まで正しくない値を返す可能性があります。
Africa/Windhoek	
America/Bogota	このタイムゾーン設定では、1914 年 11 月 23 日 04:56:16 GMT から 1914 年 11 月 23 日 04:56:20 GMT まで正しくない値を返す可能性があります。
America/Caracas	
America/Chihuahua	
America/Cuiaba	
America/Denver	
America/Fortaleza	南米 (サンパウロ) リージョンにある DB クラスターでは、最近変更されたブラジルのタイムゾーンで時刻が正しく表示されない場合があります。その場合は、DB クラスターのタイムゾーンパラメータを America/Fortaleza にリセットします。
America/Guatemala	
America/Halifax	このタイムゾーン設定では、1918 年 10 月 27 日 05:00:00 GMT から 1918 年 10 月 31 日 05:00:00 GMT まで正しくない値を返す可能性があります。
America/Manaus	ご使用の DB クラスターが南米 (クイアバ) タイムゾーンに置かれており、最近変更されたブラジルのタイムゾーンで時刻が正しく想定どおりに表示されない場合は、その DB クラスターのタイムゾーンパラメータを America/Manaus にリセットします。

タイムゾーン	コメント
America/Matamoros	
America/Monterrey	
America/Montevideo	
America/Phoenix	
America/Tijuana	
Asia/Ashgabat	
Asia/Baghdad	
Asia/Baku	
Asia/Bangkok	
Asia/Beirut	
Asia/Calcutta	
Asia/Kabul	
Asia/Karachi	
Asia/Kathmandu	
Asia/Muscat	このタイムゾーン設定では、1919 年 12 月 31 日 20:05:36 GMT から 1919 年 12 月 31 日 20:05:40 GMT まで正しくない値を返す可能性があります。
Asia/Riyadh	このタイムゾーン設定では、1947 年 3 月 13 日 20:53:08 GMT から 1949 年 12 月 31 日 20:53:08 GMT まで正しくない値を返す可能性があります。

タイムゾーン	コメント
Asia/Seoul	このタイムゾーン設定では、1904 年 11 月 30 日 15:30:00 GMT から 1945 年 9 月 7 日 15:00:00 GMT まで正しくない値を返す可能性があります。
Asia/Shanghai	このタイムゾーン設定では、1927 年 12 月 31 日 15:54:08 GMT から 1940 年 6 月 2 日 16:00:00 GMT まで正しくない値を返す可能性があります。
Asia/Singapore	
Asia/Taipei	このタイムゾーン設定では、1937 年 9 月 30 日 16:00:00 GMT から 1979 年 9 月 29 日 15:00:00 GMT まで正しくない値を返す可能性があります。
Asia/Tehran	
Asia/Tokyo	このタイムゾーン設定では、1937 年 9 月 30 日 15:00:00 GMT から 1937 年 12 月 31 日 15:00:00 GMT まで正しくない値を返す可能性があります。
Asia/Ulaanbaatar	
Atlantic/Azores	このタイムゾーン設定では、1911 年 5 月 24 日 01:54:32 GMT から 1912 年 1 月 1 日 01:54:32 GMT まで正しくない値を返す可能性があります。
Australia/Adelaide	このタイムゾーンの略名は ACDT/ACST ではなく CST として返されます。
Australia/Brisbane	このタイムゾーンの略名は AEDT/AEST ではなく EST として返されます。
Australia/Darwin	このタイムゾーンの略名は ACDT/ACST ではなく CST として返されます。

タイムゾーン	コメント
Australia/Hobart	このタイムゾーンの略名は AEDT/AEST ではなく EST として返されます。
Australia/Perth	このタイムゾーンの略名は AWDT/AWST ではなく WST として返されます。
Australia/Sydney	このタイムゾーンの略名は AEDT/AEST ではなく EST として返されます。
Brazil/East	
Canada/Saskatchewan	このタイムゾーン設定では、1918 年 10 月 27 日 08:00:00 GMT から 1918 年 10 月 31 日 08:00:00 GMT まで正しくない値を返す可能性があります。
Europe/Amsterdam	
Europe/Athens	
Europe/Dublin	
Europe/Helsinki	このタイムゾーン設定では、1921 年 4 月 30 日 22:20:08 GMT から 1921 年 4 月 30 日 22:20:11 GMT まで正しくない値を返す可能性があります。
Europe/Paris	
Europe/Prague	
Europe/Sarajevo	
Pacific/Auckland	
Pacific/Guam	

タイムゾーン	コメント
Pacific/Honolulu	このタイムゾーン設定では、1933 年 5 月 21 日 11:30:00 GMT から 1945 年 9 月 30 日 11:30:00 GMT まで正しくない値を返す可能性があります。
Pacific/Samoa	このタイムゾーン設定では、1911 年 1 月 1 日 11:22:48 GMT から 1950 年 1 月 1 日 11:30:00 GMT まで正しくない値を返す可能性があります。
US/Alaska	
US/Central	
US/Eastern	
US/East-Indiana	
US/Pacific	
UTC	

AWS リージョン と Aurora DB エンジンにより Amazon Aurora でサポートされている機能

Aurora MySQL および PostgreSQL と互換性のあるデータベースエンジンは、いくつかの Amazon Aurora および Amazon RDS の機能とオプションをサポートしています。サポートは、各データベースエンジンの特定のバージョン、および AWS リージョン によって異なります。特定の AWS リージョン の機能での Aurora データベースエンジンで使用できるバージョンとそのサポートを確認するには、以下のセクションに従います。

下記の機能の中には、Aurora 以外では使用できない機能もあります。例えば、Aurora Serverless、Aurora Global Database、および AWS 機械学習サービスとの統合のサポートは、Amazon RDS ではサポートされていません。その他の機能、例えば Amazon RDS Proxy は、Amazon Aurora と Amazon RDS の両方でサポートされています。

サポートされているリージョンと DB エンジン

- テーブルの規則
- ブルー/グリーンデプロイでサポートされているリージョンと Aurora DB エンジン
- クラスターストレージ構成でサポートされているリージョンと Aurora DB エンジン
- データベースアクティビティストリームでサポートされているリージョンと Aurora DB エンジン
- Amazon S3 へのクラスターデータエクスポートでサポートされているリージョンと Aurora DB エンジン
- Amazon S3 へのスナップショットデータエクスポートでサポートされているリージョンと Aurora DB エンジン
- Aurora グローバルデータベースでサポートされているリージョンと DB エンジン
- IAM データベース認証でサポートされているリージョンと Aurora DB エンジン
- Kerberos 認証でサポートされているリージョンと Aurora DB エンジン
- Aurora 機械学習でサポートされているリージョンと DB エンジン
- Performance Insights でサポートされているリージョンと Aurora DB エンジン
- Amazon Redshift とのゼロ ETL 統合でサポートされているリージョンと Aurora DB エンジン
- Amazon RDS Proxy でサポートされているリージョンと Aurora DB エンジン
- Secrets Manager 統合でサポートされているリージョンと Aurora DB エンジン
- Aurora Serverless v2 でサポートされているリージョンと Aurora DB エンジン
- Aurora Serverless v1 でサポートされているリージョンと Aurora DB エンジン

- RDS Data API でサポートされているリージョンと Aurora DB エンジン
- ダウンタイムなしのパッチ適用 (ZDP) でサポートされているリージョンと Aurora DB エンジン
- Aurora エンジンのネイティブ機能でサポートされているリージョンと DB エンジン

テーブルの規則

機能セクションのテーブルでは、次のパターンを使ってバージョン番号とサポートレベルを指定しています。

- バージョン x.y - サポートされているのは明記されたバージョンのみです。
- バージョン x.y 以降 - 指定されているバージョン、およびそのメジャーバージョンのすべてのマイナーバージョンです。例えば「バージョン 10.11 以降」であれば、バージョン 10.11、10.11.1、10.12 がサポートされていることを意味します。
- - - この機能は、現在、該当の Aurora データベースエンジンの特定の Aurora 機能、または特定の AWS リージョン では使用できません。

ブルー/グリーンデプロイでサポートされているリージョンと Aurora DB エンジン

ブルー/グリーンデプロイでは、本稼働データベース環境を別の同期されたステージング環境にコピーします。Amazon RDS ブルー/グリーンデプロイを使用すると、本稼働環境に影響を与えずにステージング環境のデータベースに変更を加えることができます。例えば、DB エンジンのメジャーまたはマイナーバージョンのアップグレード、データベースパラメータの変更、スキーマの変更をステージング環境で行うことができます。準備ができたら、ステージング環境を新しい本稼働データベース環境に昇格できます。詳細については、「データベース更新のために Amazon RDS ブルー/グリーンデプロイを使用する」を参照してください。

Aurora MySQL によるブルー/グリーンデプロイ

ブルー/グリーンデプロイ機能は、すべての AWS リージョン において Aurora MySQL のすべてのバージョンで使用できます。

Aurora PostgreSQL によるブルー/グリーンデプロイ

Aurora PostgreSQL で ブルー/グリーンデプロイ が使用可能なリージョンとエンジンのバージョンは、以下のとおりです。

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
すべての AWS リージョン	バージョン 16.1 以上	バージョン 15.4 以降	バージョン 14.9 以降	バージョン 13.12 以降	バージョン 12.16 以上	バージョン 11.21 以上

クラスターストレージ構成でサポートされているリージョンと Aurora DB エンジン

Amazon Aurora には、Aurora I/O-Optimized と Aurora Standard の 2 つの DB クラスターストレージ設定があります。詳細については、「Amazon Aurora DB クラスターのストレージ設定」を参照してください。

Aurora I/O-Optimized

Aurora I/O-Optimized は、次の Amazon Aurora バージョンでは、すべての AWS リージョン で使用できます。

- Aurora MySQL バージョン 3.03.1 以降
- Aurora PostgreSQL バージョン 16.1 以降、15.2 以降、14.7 以降、および 13.10 以降

Aurora Standard

Aurora Standard を使用できるのは、すべての Aurora MySQL および Aurora PostgreSQL のバージョンのすべての AWS リージョン です。

データベースアクティビティストリームでサポートされているリージョンと Aurora DB エンジン

Aurora でデータベースアクティビティストリームを使用することで、Aurora データベースアクティビティをモニタリングし、アラームを設定できます。詳細については、「データベースアクティビティストリームを使用した Amazon Aurora のモニタリング」を参照してください。

データベースアクティビティストリームは、次の機能ではサポートされていません。

- Aurora Serverless v1
- Aurora Serverless v2
- Babelfish for Aurora PostgreSQL

トピック

- Aurora MySQL でデータベースアクティビティストリームを使用する
- Aurora PostgreSQL でデータベースアクティビティストリームを使用する

Aurora MySQL でデータベースアクティビティストリームを使用する

Aurora MySQL によるデータベースアクティビティストリームが使用可能なリージョンとエンジンのバージョンは以下のとおりです。

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
米国東部 (オハイオ)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
米国東部 (バージニア北部)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
米国西部 (北カリフォルニア)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
米国西部 (オレゴン)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
アフリカ (ケープタウン)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
アジアパシフィック (香港)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
アジアパシフィック (ハイデラバード)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
アジアパシフィック (ジャカルタ)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
アジアパシフィック (ムンバイ)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
アジアパシフィック (大阪)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
アジアパシフィック (ソウル)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
アジアパシフィック (シンガポール)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
アジアパシフィック (シドニー)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
アジアパシフィック (東京)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
カナダ (中部)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
カナダ西部 (カルガリー)	–	–
中国 (北京)	–	–
中国 (寧夏)	–	–
欧州 (フランクフルト)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
欧州 (アイルランド)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
欧州 (ロンドン)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
欧州 (ミラノ)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
欧州 (パリ)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
欧州 (スペイン)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
欧州 (ストックホルム)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
欧州 (チューリッヒ)	–	–
イスラエル (テルアビブ)	–	–
中東 (バーレーン)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
中東 (アラブ首長国連邦)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降
南米 (サンパウロ)	使用可能なすべてのバージョン	Aurora バージョン 2.11 以降

Aurora PostgreSQL でデータベースアクティビティストリームを使用する

Aurora PostgreSQL によるデータベースアクティビティストリームが使用可能なリージョンとエンジンのバージョンは以下のとおりです。

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
米国東部 (オハイオ)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 と

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
						バージョン 11.13 以降
米国東部 (バージニア北部)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降
米国西部 (北カリフォルニア)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降
米国西部 (オレゴン)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降
アフリカ (ケープタウン)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (香港)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (ハイデラバード)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
アジアパシフィック (ジャカルタ)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (メルボルン)	–	–	–	–	–	–
アジアパシフィック (ムンバイ)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (大阪)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (ソウル)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (シンガポール)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (シドニー)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
アジアパシフィック (東京)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降
カナダ (中部)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降
カナダ西部 (カルガリー)	–	–	–	–	–	–
中国 (北京)	–	–	–	–	–	–
中国 (寧夏)	–	–	–	–	–	–
欧州 (フランクフルト)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降
欧州 (アイルランド)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降
欧州 (ロンドン)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
欧州 (ミラノ)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降
欧州 (パリ)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降
欧州 (スペイン)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降
欧州 (ストックホルム)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降
欧州 (チューリッヒ)	–	–	–	–	–	–
イスラエル (テルアビブ)	–	–	–	–	–	–
中東 (バーレーン)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
中東 (アラブ首長国連邦)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降
南米 (サンパウロ)	バージョン 16.1 以上	バージョン 15.2 以降	使用可能なすべてのバージョン	使用可能なすべてのバージョン	使用可能なすべてのバージョン	バージョン 11.9 とバージョン 11.13 以降

Amazon S3 へのクラスターデータエクスポートでサポートされているリージョンと Aurora DB エンジン

Aurora DB クラスターデータを Amazon S3 バケットにエクスポートできます。データをエクスポートすると、Amazon Athena や Amazon Redshift Spectrum などのツールを使用して、エクスポートしたデータを直接分析できます。詳細については、「Amazon S3 への DB クラスターデータのエクスポート」を参照してください。

S3 へのクラスターデータのエクスポートは、以下の AWS リージョン で使用可能です。

- アジアパシフィック (香港)
- アジアパシフィック (ムンバイ)
- アジアパシフィック (大阪)
- アジアパシフィック (ソウル)
- アジアパシフィック (シンガポール)
- アジアパシフィック (シドニー)
- アジアパシフィック (東京)
- カナダ (中部)
- カナダ西部 (カルガリー)
- 中国 (寧夏)
- 欧州 (フランクフルト)

- 欧州 (アイルランド)
- 欧州 (ロンドン)
- 欧州 (パリ)
- 欧州 (ストックホルム)
- 南米 (サンパウロ)
- 米国東部 (バージニア北部)
- 米国東部 (オハイオ)
- 米国西部 (北カリフォルニア)
- 米国西部 (オレゴン)

トピック

- Aurora MySQL でクラスターデータを S3 にエクスポートする
- Aurora PostgreSQL でクラスターデータを S3 にエクスポートする

Aurora MySQL でクラスターデータを S3 にエクスポートする

現在使用可能な Aurora MySQL エンジンのすべてのバージョンは、DB クラスターデータの Amazon S3 へのエクスポートをサポートしています。バージョンの詳細については、「Aurora MySQL リリースノート」を参照してください。

Aurora PostgreSQL でクラスターデータを S3 にエクスポートする

現在使用可能な Aurora PostgreSQL エンジンのすべてのバージョンは、DB クラスターデータの Amazon S3 へのエクスポートをサポートしています。バージョンの詳細については、「Aurora PostgreSQL リリースノート」を参照してください。

Amazon S3 へのスナップショットデータエクスポートでサポートされているリージョンと Aurora DB エンジン

Aurora DB クラススナップショットデータを Amazon S3 バケットにエクスポートできます。手動スナップショットと自動システムスナップショットをエクスポートできます。データをエクスポートすると、Amazon Athena や Amazon Redshift Spectrum などのツールを使用して、エクスポートしたデータを直接分析できます。詳細については、「Amazon S3 への DB クラススナップショットデータのエクスポート」を参照してください。

S3 へのスナップショットのエクスポートは、以下を除くすべての AWS リージョン で使用可能です。

- アジアパシフィック (ハイデラバード)
- アジアパシフィック (ジャカルタ)
- アジアパシフィック (メルボルン)
- カナダ西部 (カルガリー)
- 欧州 (スペイン)
- 欧州 (チューリッヒ)
- イスラエル (テルアビブ)
- 中東 (アラブ首長国連邦)
- AWS GovCloud (米国東部)
- AWS GovCloud (米国西部)

トピック

- Aurora MySQL でスナップショットデータを S3 にエクスポートする
- Aurora PostgreSQL でスナップショットデータを S3 にエクスポートする

Aurora MySQL でスナップショットデータを S3 にエクスポートする

現在使用可能な Aurora MySQL エンジンのすべてのバージョンは、DB クラスタースナップショットデータの Amazon S3 へのエクスポートをサポートしています。バージョンの詳細については、「Aurora MySQL リリースノート」を参照してください。

Aurora PostgreSQL でスナップショットデータを S3 にエクスポートする

現在使用可能な Aurora PostgreSQL エンジンのすべてのバージョンは、DB クラスタースナップショットデータの Amazon S3 へのエクスポートをサポートしています。バージョンの詳細については、「Aurora PostgreSQL リリースノート」を参照してください。

Aurora グローバルデータベースでサポートされているリージョンと DB エンジン

Aurora グローバルデータベースは、低レイテンシーでグローバルな読み取り機能と、リージョン全域の機能停止時の災害対策機能を備えた、複数の AWS リージョン にまたがる単一のデータベース

です。DB インスタンスは、単一の AWS リージョン ではなく、複数のリージョンおよび異なるアベイラビリティーゾーンに依存しているので、デプロイには耐障害性が組み込まれています。詳細については、「Amazon Aurora Global Database の使用」を参照してください。

トピック

- Aurora MySQL を使用した Aurora グローバルデータベース
- Aurora PostgreSQL を使用した Aurora グローバルデータベース

Aurora MySQL を使用した Aurora グローバルデータベース

Aurora MySQL による Aurora グローバルデータベースが使用可能なリージョンとエンジンのバージョンは以下のとおりです。

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
米国東部 (オハイオ)	バージョン 3.01.0 以降	バージョン 2.07.0 以降
米国東部 (バージニア北部)	バージョン 3.01.0 以降	バージョン 2.07.0 以降
米国西部 (北カリフォルニア)	バージョン 3.01.0 以降	バージョン 2.07.0 以降
米国西部 (オレゴン)	バージョン 3.01.0 以降	バージョン 2.07.0 以降
アフリカ (ケープタウン)	バージョン 3.01.0 以降	バージョン 2.07.1 以降
アジアパシフィック (香港)	バージョン 3.01.0 以降	バージョン 2.07.1 以降
アジアパシフィック (ハイデラバード)	バージョン 3.02.0 以降	バージョン 2.11.2 以降
アジアパシフィック (ジャカルタ)	バージョン 3.01.0 以降	バージョン 2.07.6 以降
アジアパシフィック (メルボルン)	バージョン 3.03.0 以降	–
アジアパシフィック (ムンバイ)	バージョン 3.01.0 以降	バージョン 2.07.0 以降

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
アジアパシフィック (大阪)	バージョン 3.01.0 以降	バージョン 2.07.3 以降
アジアパシフィック (ソウル)	バージョン 3.01.0 以降	バージョン 2.07.0 以降
アジアパシフィック (シンガポール)	バージョン 3.01.0 以降	バージョン 2.07.0 以降
アジアパシフィック (シドニー)	バージョン 3.01.0 以降	バージョン 2.07.0 以降
アジアパシフィック (東京)	バージョン 3.01.0 以降	バージョン 2.07.0 以降
カナダ (中部)	バージョン 3.01.0 以降	バージョン 2.07.0 以降
カナダ西部 (カルガリー)	バージョン 3.01.0 以降	バージョン 2.07.0 以降
中国 (北京)	バージョン 3.01.0 以降	バージョン 2.07.2 以降
中国 (寧夏)	バージョン 3.01.0 以降	バージョン 2.07.2 以降
欧州 (フランクフルト)	バージョン 3.01.0 以降	バージョン 2.07.0 以降
欧州 (アイルランド)	バージョン 3.01.0 以降	バージョン 2.07.0 以降
欧州 (ロンドン)	バージョン 3.01.0 以降	バージョン 2.07.0 以降
欧州 (ミラノ)	バージョン 3.01.0 以降	バージョン 2.07.1 以降
欧州 (パリ)	バージョン 3.01.0 以降	バージョン 2.07.0 以降
欧州 (スペイン)	バージョン 3.02.0 以降	–
欧州 (ストックホルム)	バージョン 3.01.0 以降	バージョン 2.07.0 以降
欧州 (チューリッヒ)	バージョン 3.02.0 以降	–
イスラエル (テルアビブ)	–	–
中東 (バーレーン)	バージョン 3.01.0 以降	バージョン 2.07.1 以降

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
中東 (アラブ首長国連邦)	バージョン 3.02.0 以降	–
南米 (サンパウロ)	バージョン 3.01.0 以降	バージョン 2.07.1 以降
AWS GovCloud (米国東部)	バージョン 3.01.0 以降	バージョン 2.07.0 以降
AWS GovCloud (米国西部)	バージョン 3.01.0 以降	バージョン 2.07.0 以降

Aurora PostgreSQL を使用した Aurora グローバルデータベース

Aurora PostgreSQL による Aurora グローバルデータベースが使用可能なリージョンとエンジンのバージョンは以下のとおりです。

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
米国東部 (オハイオ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
米国東部 (バージニア北部)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
米国西部 (北カリフォルニア)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
米国西部 (オレゴン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
アフリカ (ケープタウン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (香港)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (ハイデラバード)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (ジャカルタ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (メルボルン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック（ムンバイ）	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (大阪)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
アジアパシフィック (ソウル)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (シンガポール)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (シドニー)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (東京)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
カナダ (中部)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
カナダ西部 (カルガリー)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
中国 (北京)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
中国 (寧夏)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
欧州 (フランクフルト)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
欧州 (アイルランド)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
欧州 (ロンドン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
欧州 (ミラノ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
欧州 (パリ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
欧州 (スペイン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
欧州 (ストックホルム)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
欧州 (チューリッヒ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
イスラエル (テルアビブ)	–	–	–	–	–	–
中東 (バーレーン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
中東 (アラブ首長国連邦)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
南米 (サンパウロ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
AWS GovCloud (米国東部)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
AWS GovCloud (米国西部)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降

IAM データベース認証でサポートされているリージョンと Aurora DB エンジン

Aurora の IAM データベース認証では、AWS Identity and Access Management (IAM) データベース認証を使用して、DB クラスターに認証できます。この認証方法では、DB クラスターに接続する際にパスワードを使用する必要はありません。代わりに、認証トークンを使用します。詳細については、「 の IAM データベース認証」を参照してください。

トピック

- Aurora MySQL で IAM データベース認証を使用する
- Aurora PostgreSQL で IAM データベース認証を使用する

Aurora MySQL で IAM データベース認証を使用する

Aurora MySQL で IAM データベース認証を使用する場合、以下のバージョンがすべてのリージョンで使用できます。

- Aurora MySQL 3 - 使用可能なすべてのバージョン
- Aurora MySQL 2 - 使用可能なすべてのバージョン

Aurora PostgreSQL で IAM データベース認証を使用する

Aurora PostgreSQL での IAM データベース認証は、以下のエンジンバージョンでは、すべてのリージョンで使用できます。

- Aurora PostgreSQL 16 - 使用可能なすべてのバージョン
- Aurora PostgreSQL 15 - 使用可能なすべてのバージョン

- Aurora PostgreSQL 14 - 使用可能なすべてのバージョン
- Aurora PostgreSQL 13 - 使用可能なすべてのバージョン
- Aurora PostgreSQL 12 - 使用可能なすべてのバージョン
- Aurora PostgreSQL 11 - 使用可能なすべてのバージョン

Kerberos 認証でサポートされているリージョンと Aurora DB エンジン

Aurora で Kerberos 認証を使用することで、Kerberos および Microsoft Active Directory を使用して、データベースユーザーの外部認証をサポートできます。Kerberos と Active Directory を使用することで、シングルサインオンとデータベースユーザーの一元化認証という利点が得られます。Kerberos と Active Directory は AWS Directory Service の機能の 1 つの AWS Directory Service for Microsoft Active Directory で利用できます。詳細については、「Kerberos 認証」を参照してください。

トピック

- Aurora MySQL で Kerberos 認証を使用する
- Aurora PostgreSQL で Kerberos 認証を使用する

Aurora MySQL で Kerberos 認証を使用する

Aurora MySQL での Kerberos 認証が使用可能なリージョンとエンジンのバージョンは以下のとおりです。

リージョン	Aurora MySQL バージョン 3
米国東部 (オハイオ)	バージョン 3.03.0 以降
米国東部 (バージニア北部)	バージョン 3.03.0 以降
米国西部 (北カリフォルニア)	バージョン 3.03.0 以降
米国西部 (オレゴン)	バージョン 3.03.0 以降
アフリカ (ケープタウン)	–
アジアパシフィック (香港)	–

リージョン	Aurora MySQL バージョン 3
アジアパシフィック (ジャカルタ)	–
アジアパシフィック (ムンバイ)	バージョン 3.03.0 以降
アジアパシフィック (大阪)	–
アジアパシフィック (ソウル)	バージョン 3.03.0 以降
アジアパシフィック (シンガポール)	バージョン 3.03.0 以降
アジアパシフィック (シドニー)	バージョン 3.03.0 以降
アジアパシフィック (東京)	バージョン 3.03.0 以降
カナダ (中部)	バージョン 3.03.0 以降
カナダ西部 (カルガリー)	–
中国 (北京)	バージョン 3.03.0 以降
中国 (寧夏)	バージョン 3.03.0 以降
欧州 (フランクフルト)	バージョン 3.03.0 以降
欧州 (アイルランド)	バージョン 3.03.0 以降
欧州 (ロンドン)	バージョン 3.03.0 以降
欧州 (ミラノ)	–
欧州 (パリ)	バージョン 3.03.0 以降
欧州 (スペイン)	–
欧州 (ストックホルム)	バージョン 3.03.0 以降
欧州 (チューリッヒ)	–
イスラエル (テルアビブ)	–

リージョン	Aurora MySQL バージョン 3
中東 (バーレーン)	–
中東 (アラブ首長国連邦)	–
南米 (サンパウロ)	バージョン 3.03.0 以降
AWS GovCloud (米国東部)	バージョン 3.03.0 以降
AWS GovCloud (米国西部)	バージョン 3.03.0 以降

Aurora PostgreSQL で Kerberos 認証を使用する

Aurora PostgreSQL での Kerberos 認証が使用可能なリージョンとエンジンのバージョンは以下のとおりです。

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
米国東部 (オハイオ)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
米国東部 (バージニア北部)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
米国西部 (北カリフォルニア)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
米国西部 (オレゴン)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
アフリカ (ケープタウン)	–	–	–	–	–	–
アジアパシフィック (香港)	–	–	–	–	–	–
アジアパシフィック (ハイデラバード)	–	–	–	–	–	–
アジアパシフィック (ジャカルタ)	–	–	–	–	–	–
アジアパシフィック (メルボルン)	–	–	–	–	–	–
アジアパシフィック (ムンバイ)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
アジアパシフィック (大阪)	–	–	–	–	–	–

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
アジアパシフィック (ソウル)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
アジアパシフィック (シンガポール)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
アジアパシフィック (シドニー)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
アジアパシフィック (東京)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
カナダ (中部)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
カナダ西部 (カルガリー)	–	–	–	–	–	–
中国 (北京)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
中国 (寧夏)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
欧州 (フランクフルト)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
欧州 (アイルランド)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
欧州 (ロンドン)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
欧州 (ミラノ)	–	–	–	–	–	–
欧州 (パリ)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
欧州 (スペイン)	–	–	–	–	–	–
欧州 (ストックホルム)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
欧州 (チューリッヒ)	–	–	–	–	–	–
イスラエル (テルアビブ)	–	–	–	–	–	–
中東 (バーレーン)	–	–	–	–	–	–
中東 (アラブ首長国連邦)	–	–	–	–	–	–

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
南米 (サンパウロ)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
AWS GovCloud (米国東部)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
AWS GovCloud (米国西部)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン

Aurora 機械学習でサポートされているリージョンと DB エンジン

Amazon Aurora 機械学習を使用することで、必要に応じて Aurora DB クラスターを Amazon Comprehend または Amazon SageMaker と統合できます。Amazon Comprehend と SageMaker はそれぞれ異なる機械学習のユースケースをサポートしています。Amazon Comprehend は、ドキュメントからインサイトを抽出するために使用される自然言語処理 (NLP) サービスです。Amazon Comprehend と Aurora 機械学習を使用することで、データベーステーブル内のテキストの感情を判断できます。SageMaker は、フルマネージド型の機械学習サービスです。データサイエンティストは Amazon SageMaker を使用して、不正の検出などのさまざまな推論タスク用の機械学習モデルの構築、トレーニング、テストを行っています。SageMaker で Aurora 機械学習を使用することで、データベース開発者は SQL コードで SageMaker 機能を呼び出すことができます。

すべての AWS リージョン が Amazon Comprehend と SageMaker の両方をサポートしているわけではなく、特定の AWS リージョン のみが Aurora 機械学習をサポートしています。そのため、Aurora DB クラスターからこれらのサービスにアクセスできます。また、Aurora 機械学習の統合プロセスはデータベースエンジンによって異なります。詳細については、「Amazon Aurora 機械学習の使用」を参照してください。

トピック

- Aurora MySQL を使用した Aurora Machine Learning
- Aurora PostgreSQL を使用した Aurora Machine Learning

Aurora MySQL を使用した Aurora Machine Learning

Aurora 機械学習は、以下の表に記載されている AWS リージョン の Aurora MySQL でサポートされています。ご使用のバージョンの Aurora MySQL を利用できるだけでなく、AWS リージョン で使用するサービスがサポートされている必要があります。Amazon SageMaker が利用可能な AWS リージョン の一覧については、Amazon Web Services 全般のリファレンス の「Amazon SageMaker エンドポイントとクォータ」を参照してください。Amazon Comprehend が利用可能な AWS リージョン の一覧については、Amazon Web Services 全般のリファレンス の「Amazon Comprehend エンドポイントとクォータ」を参照してください。

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
米国東部 (オハイオ)	バージョン 3.01.0 以降	バージョン 2.07 以降
米国東部 (バージニア北部)	バージョン 3.01.0 以降	バージョン 2.07 以降
米国西部 (北カリフォルニア)	バージョン 3.01.0 以降	バージョン 2.07 以降
米国西部 (オレゴン)	バージョン 3.01.0 以降	バージョン 2.07 以降
アフリカ (ケープタウン)	–	–
アジアパシフィック (香港)	バージョン 3.01.0 以降	バージョン 2.07 以降
アジアパシフィック (ハイデラバード)	バージョン 3.01.0 以降	バージョン 2.07 以降
アジアパシフィック (ジャカルタ)	バージョン 3.01.0 以降	バージョン 2.07 以降
アジアパシフィック (メルボルン)	バージョン 3.01.0 以降	バージョン 2.07 以降
アジアパシフィック (ムンバイ)	バージョン 3.01.0 以降	バージョン 2.07 以降
アジアパシフィック (大阪)	バージョン 3.01.0 以降	バージョン 2.07.3 以降
アジアパシフィック (ソウル)	バージョン 3.01.0 以降	バージョン 2.07 以降

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
アジアパシフィック (シンガポール)	バージョン 3.01.0 以降	バージョン 2.07 以降
アジアパシフィック (シドニー)	バージョン 3.01.0 以降	バージョン 2.07 以降
アジアパシフィック (東京)	バージョン 3.01.0 以降	バージョン 2.07 以降
カナダ (中部)	バージョン 3.01.0 以降	バージョン 2.07 以降
カナダ西部 (カルガリー)	バージョン 3.01.0 以降	バージョン 2.07 以降
中国 (北京)	バージョン 3.01.0 以降	バージョン 2.07 以降
中国 (寧夏)	バージョン 3.01.0 以降	バージョン 2.07 以降
欧州 (フランクフルト)	バージョン 3.01.0 以降	バージョン 2.07 以降
欧州 (アイルランド)	バージョン 3.01.0 以降	バージョン 2.07 以降
欧州 (ロンドン)	バージョン 3.01.0 以降	バージョン 2.07 以降
欧州 (ミラノ)	–	–
欧州 (パリ)	バージョン 3.01.0 以降	バージョン 2.07 以降
欧州 (スペイン)	バージョン 3.01.0 以降	バージョン 2.07 以降
欧州 (ストックホルム)	バージョン 3.01.0 以降	バージョン 2.07 以降
欧州 (チューリッヒ)	バージョン 3.01.0 以降	バージョン 2.07 以降
イスラエル (テルアビブ)	バージョン 3.01.0 以降	バージョン 2.07 以降
中東 (バーレーン)	バージョン 3.01.0 以降	バージョン 2.07 以降
中東 (アラブ首長国連邦)	バージョン 3.01.0 以降	バージョン 2.07 以降
南米 (サンパウロ)	バージョン 3.01.0 以降	バージョン 2.07 以降

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
AWS GovCloud (米国東部)	バージョン 3.01.0 以降	バージョン 2.07 以降
AWS GovCloud (米国西部)	バージョン 3.01.0 以降	バージョン 2.07 以降

Aurora PostgreSQL を使用した Aurora Machine Learning

Aurora 機械学習は、以下の表に記載されている AWS リージョン の Aurora PostgreSQL でサポートされています。ご使用のバージョンの Aurora PostgreSQL を利用できるだけでなく、AWS リージョン で使用するサービスがサポートされている必要があります。Amazon SageMaker が利用可能な AWS リージョン の一覧については、Amazon Web Services 全般のリファレンス の「Amazon SageMaker エンドポイントとクォータ」を参照してください。Amazon Comprehend が利用可能な AWS リージョン の一覧については、Amazon Web Services 全般のリファレンス の「Amazon Comprehend エンドポイントとクォータ」を参照してください。

Aurora PostgreSQL による Aurora 機械学習が使用可能なリージョンとエンジンのバージョンは以下のとおりです。

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
米国東部 (オハイオ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
米国東部 (バージニア北部)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
米国西部 (北カリフォルニア)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
米国西部 (オレゴン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
アフリカ (ケープタウン)	–	–	–	–	–	–
アジアパシフィック (香港)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
アジアパシフィック (ハイデラバード)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
アジアパシフィック (ジャカルタ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
アジアパシフィック (メルボルン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
アジアパシフィック (ムンバイ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
アジアパシフィック (大阪)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
アジアパシフィック (ソウル)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
アジアパシフィック (シンガポール)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
アジアパシフィック (シドニー)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
アジアパシフィック (東京)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
カナダ (中部)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
カナダ西部 (カルガリー)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
中国 (北京)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
中国 (寧夏)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
欧州 (フランクフルト)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
欧州 (アイルランド)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
欧州 (ロンドン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
欧州 (ミラノ)	–	–	–	–	–	–
欧州 (パリ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
欧州 (スペイン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
欧州 (ストックホルム)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
欧州 (チューリッヒ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
イスラエル (テルアビブ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
中東 (バーレーン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
中東 (アラブ首長国連邦)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
南米 (サンパウロ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
AWS GovCloud (米国東部)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降
AWS GovCloud (米国西部)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3	バージョン 13.3 以降	バージョン 12.4 以上	バージョン 11.9、11.12 以降

Performance Insights でサポートされているリージョンと Aurora DB エンジン

Performance Insights によって、既存の Amazon RDS モニタリング機能を拡張し、データベースのパフォーマンスを明確にして分析しやすくします。Performance Insights ダッシュボードを使用して Amazon RDS DB インスタンスのロードのデータベースロードを視覚化したり、待機ロード、SQL ステートメント、ホスト、ユーザー別にフィルタリングしたりできます。詳細については、「Amazon Aurora での Performance Insights の概要」を参照してください。

Performance Insights 機能のリージョン、DB エンジン、およびインスタンスクラスのサポート情報については、「Amazon Aurora DB エンジン、リージョン、およびインスタンスクラスでサポートされている Performance Insights 機能」を参照してください。

トピック

- Aurora MySQL で Performance Insights を使用する
- Aurora PostgreSQL で Performance Insights を使用する
- Aurora Serverless で Performance Insights を使用する

Aurora MySQL で Performance Insights を使用する

> **Note**
>
> パラレルクエリを有効にしている場合、Aurora MySQL で Performance Insights を使用する場合、サポートされているエンジンバージョンが異なります。パラレルロードの詳細については、「Amazon Aurora MySQL のパラレルクエリの使用」を参照してください。

トピック

- Aurora MySQL で Performance Insights を使用する (パラレルクエリがオフの場合)
- Aurora MySQL で Performance Insights を使用する (パラレルクエリがオンの場合)

Aurora MySQL で Performance Insights を使用する (パラレルクエリがオフの場合)

Aurora MySQL で並列クエリがオフのときに Performance Insights が使用可能なリージョンとエンジンのバージョンは、以下のとおりです。

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
米国東部 (オハイオ)	すべてのバージョン	すべてのバージョン
米国東部 (バージニア北部)	すべてのバージョン	すべてのバージョン
米国西部 (北カリフォルニア)	すべてのバージョン	すべてのバージョン
米国西部 (オレゴン)	すべてのバージョン	すべてのバージョン
アフリカ (ケープタウン)	すべてのバージョン	すべてのバージョン
アジアパシフィック (香港)	すべてのバージョン	すべてのバージョン

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
アジアパシフィック (ハイデラバード)	すべてのバージョン	すべてのバージョン
アジアパシフィック (ジャカルタ)	すべてのバージョン	すべてのバージョン
アジアパシフィック (メルボルン)	すべてのバージョン	すべてのバージョン
アジアパシフィック (ムンバイ)	すべてのバージョン	すべてのバージョン
アジアパシフィック (大阪)	すべてのバージョン	すべてのバージョン
アジアパシフィック (ソウル)	すべてのバージョン	すべてのバージョン
アジアパシフィック (シンガポール)	すべてのバージョン	すべてのバージョン
アジアパシフィック (シドニー)	すべてのバージョン	すべてのバージョン
アジアパシフィック (東京)	すべてのバージョン	すべてのバージョン
カナダ (中部)	すべてのバージョン	すべてのバージョン
カナダ西部 (カルガリー)	すべてのバージョン	すべてのバージョン
中国 (北京)	すべてのバージョン	すべてのバージョン
中国 (寧夏)	すべてのバージョン	すべてのバージョン
欧州 (フランクフルト)	すべてのバージョン	すべてのバージョン
欧州 (アイルランド)	すべてのバージョン	すべてのバージョン
欧州 (ロンドン)	すべてのバージョン	すべてのバージョン
欧州 (ミラノ)	すべてのバージョン	すべてのバージョン

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
欧州 (パリ)	すべてのバージョン	すべてのバージョン
欧州 (スペイン)	すべてのバージョン	すべてのバージョン
欧州 (ストックホルム)	すべてのバージョン	すべてのバージョン
欧州 (チューリッヒ)	すべてのバージョン	すべてのバージョン
イスラエル (テルアビブ)	すべてのバージョン	すべてのバージョン
中東 (バーレーン)	すべてのバージョン	すべてのバージョン
中東 (アラブ首長国連邦)	すべてのバージョン	すべてのバージョン
南米 (サンパウロ)	すべてのバージョン	すべてのバージョン
AWS GovCloud (米国東部)	すべてのバージョン	すべてのバージョン
AWS GovCloud (米国西部)	すべてのバージョン	すべてのバージョン

Aurora MySQL で Performance Insights を使用する (パラレルクエリがオンの場合)

Aurora MySQL で並列クエリがオンのときに Performance Insights が使用可能なリージョンとエンジンのバージョンは、以下のとおりです。

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
米国東部 (オハイオ)	–	バージョン 2.09.0 以降
米国東部 (バージニア北部)	–	バージョン 2.09.0 以降
米国西部 (北カリフォルニア)	–	バージョン 2.09.0 以降
米国西部 (オレゴン)	–	バージョン 2.09.0 以降
アフリカ (ケープタウン)	–	バージョン 2.09.0 以降

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
アジアパシフィック (香港)	–	バージョン 2.09.0 以降
アジアパシフィック (ハイデラバード)	–	すべてのバージョン
アジアパシフィック (ジャカルタ)	–	バージョン 2.09.0 以降
アジアパシフィック (メルボルン)	–	バージョン 2.09.0 以降
アジアパシフィック (ムンバイ)	–	バージョン 2.09.0 以降
アジアパシフィック (大阪)	–	バージョン 2.09.0 以降
アジアパシフィック (ソウル)	–	バージョン 2.09.0 以降
アジアパシフィック (シンガポール)	–	バージョン 2.09.0 以降
アジアパシフィック (シドニー)	–	バージョン 2.09.0 以降
アジアパシフィック (東京)	–	バージョン 2.09.0 以降
カナダ (中部)	–	バージョン 2.09.0 以降
カナダ西部 (カルガリー)	–	バージョン 2.09.0 以降
中国 (北京)	–	バージョン 2.09.0 以降
中国 (寧夏)	–	バージョン 2.09.0 以降
欧州 (フランクフルト)	–	バージョン 2.09.0 以降
欧州 (アイルランド)	–	バージョン 2.09.0 以降
欧州 (ロンドン)	–	バージョン 2.09.0 以降

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
欧州 (ミラノ)	–	バージョン 2.09.0 以降
欧州 (パリ)	–	バージョン 2.09.0 以降
欧州 (スペイン)	–	バージョン 2.09.0 以降
欧州 (ストックホルム)	–	バージョン 2.09.0 以降
欧州 (チューリッヒ)	–	バージョン 2.09.0 以降
イスラエル (テルアビブ)	–	バージョン 2.09.0 以降
中東 (バーレーン)	–	バージョン 2.09.0 以降
中東 (アラブ首長国連邦)	–	バージョン 2.09.0 以降
南米 (サンパウロ)	–	バージョン 2.09.0 以降
AWS GovCloud (米国東部)	–	バージョン 2.09.0 以降
AWS GovCloud (米国西部)	–	バージョン 2.09.0 以降

Aurora PostgreSQL で Performance Insights を使用する

Aurora PostgreSQL で Performance Insights が使用可能なリージョンとエンジンのバージョンは以下のとおりです。

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
米国東部 (オハイオ)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
米国東部 (バージニア北部)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
米国西部 (北カリフォルニア)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
米国西部 (オレゴン)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
アフリカ (ケープタウン)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
アジアパシフィック (香港)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
アジアパシフィック (ハイデラバード)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
アジアパシフィック (ジャカルタ)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
アジアパシフィック (メルボルン)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
アジアパシフィック (ムンバイ)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
アジアパシフィック (大阪)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
アジアパシフィック (ソウル)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
アジアパシフィック (シンガポール)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
アジアパシフィック (シドニー)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
アジアパシフィック (東京)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
カナダ (中部)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
カナダ西部 (カルガリー)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
中国 (北京)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
中国 (寧夏)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
欧州 (フランクフルト)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
欧州 (アイルランド)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
欧州 (ロンドン)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
欧州 (ミラノ)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
欧州 (パリ)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
欧州 (スペイン)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
欧州 (ストックホルム)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
欧州 (チューリッヒ)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
イスラエル (テルアビブ)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
中東 (バーレーン)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
中東 (アラブ首長国連邦)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
南米 (サンパウロ)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン
AWS GovCloud (米国東部)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
AWS GovCloud (米国西部)	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン	すべてのバージョン

Aurora Serverless で Performance Insights を使用する

Aurora Serverless v2 は、すべての MySQL 互換バージョンと PostgreSQL 互換バージョンの Performance Insights をサポートします。最小キャパシティは少なくとも 2 Aurora キャパシティユニット (ACU) に設定することをお勧めします。

Aurora Serverless v1 は Performance Insights をサポートしていません。

Amazon Redshift とのゼロ ETL 統合でサポートされているリージョンと Aurora DB エンジン

Amazon Redshift との Amazon Aurora ゼロ ETL 統合は、トランザクションデータを Aurora クラスターに書き込んだ後に Amazon Redshift で利用できるようにするためのフルマネージドソリューションです。詳細については、「ゼロ ETL 統合での作業」を参照してください。

Amazon Redshift とのゼロ ETL 統合は、以下のリージョンとエンジンバージョンで利用できます。

トピック

- Aurora MySQL ゼロ ETL 統合
- Aurora PostgreSQL ゼロ ETL 統合

Aurora MySQL ゼロ ETL 統合

リージョン	Aurora MySQL バージョン 3
米国東部 (バージニア北部)	バージョン 3.05.2 以上
米国東部 (オハイオ)	バージョン 3.05.2 以上

リージョン	Aurora MySQL バージョン 3
米国西部 (オレゴン)	バージョン 3.05.2 以上
米国西部 (北カリフォルニア)	バージョン 3.05.2 以上
アジアパシフィック (東京)	バージョン 3.05.2 以上
アジアパシフィック (シンガポール)	バージョン 3.05.2 以上
アジアパシフィック (ソウル)	バージョン 3.05.2 以上
アジアパシフィック (ムンバイ)	バージョン 3.05.2 以上
アジアパシフィック (香港)	バージョン 3.05.2 以上
アジアパシフィック (大阪)	バージョン 3.05.2 以上
アジアパシフィック (シドニー)	バージョン 3.05.2 以上
欧州 (フランクフルト)	バージョン 3.05.2 以上
欧州 (ストックホルム)	バージョン 3.05.2 以上
欧州 (アイルランド)	バージョン 3.05.2 以上
欧州 (パリ)	バージョン 3.05.2 以上
欧州 (ロンドン)	バージョン 3.05.2 以上
欧州 (ミラノ)	バージョン 3.05.2 以上
南米 (サンパウロ)	バージョン 3.05.2 以上
カナダ (中部)	バージョン 3.05.2 以上
中東 (バーレーン)	バージョン 3.05.2 以上
アフリカ (ケープタウン)	バージョン 3.05.2 以上
中国 (北京)	バージョン 3.05.2 以上

リージョン	Aurora MySQL バージョン 3
中国 (寧夏)	バージョン 3.05.2 以上

Aurora PostgreSQL ゼロ ETL 統合

Amazon Redshift との Aurora PostgreSQL ゼロ ETL 統合のプレビューリリースでは、米国東部 (オハイオ) (米国東部-2) AWS リージョン の Amazon RDS データベースプレビュー環境内に統合を作成する必要があります。プレビュー環境では、PostgreSQL データベースエンジンソフトウェアのベータ、リリース候補、および初期の本番バージョンをテストできます。

ソース DB クラスターは Aurora PostgreSQL (PostgreSQL 15.4 およびゼロ ETL サポートと互換性あり) を実行している必要があります。

Amazon RDS Proxy でサポートされているリージョンと Aurora DB エンジン

Amazon RDS Proxy は、確立済みのデータベース接続をプーリングし共有することでアプリケーションのスケーラビリティを高める、フルマネージドの高可用性データベースプロキシです。RDS Proxy の詳細については、「Amazon RDS Proxy for Aurora の使用」を参照してください。

トピック

- Aurora MySQL による Amazon RDS Proxy
- Aurora PostgreSQL による Amazon RDS Proxy

Aurora MySQL による Amazon RDS Proxy

Aurora MySQL で RDS Proxy が使用可能なリージョンとエンジンのバージョンは以下のとおりです。

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
米国東部 (オハイオ)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
米国東部 (バージニア北部)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
米国西部 (北カリフォルニア)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
米国西部 (オレゴン)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
アフリカ (ケープタウン)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
アジアパシフィック (香港)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
アジアパシフィック (ハイデラバード)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
アジアパシフィック (ジャカルタ)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
アジアパシフィック (メルボルン)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
アジアパシフィック (ムンバイ)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
アジアパシフィック (大阪)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
アジアパシフィック (ソウル)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
アジアパシフィック (シンガポール)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
アジアパシフィック (シドニー)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
アジアパシフィック (東京)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
カナダ (中部)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
カナダ西部 (カルガリー)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
中国 (北京)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
中国 (寧夏)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
欧州 (フランクフルト)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
欧州 (アイルランド)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
欧州 (ロンドン)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
欧州 (ミラノ)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
欧州 (パリ)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
欧州 (スペイン)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
欧州 (ストックホルム)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
欧州 (チューリッヒ)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
イスラエル (テルアビブ)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
中東 (バーレーン)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
中東 (アラブ首長国連邦)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
南米 (サンパウロ)	バージョン 3.01.0 以降	バージョン 2.07 とバージョン 2.11 以降
AWS GovCloud (米国東部)	–	–
AWS GovCloud (米国西部)	–	–

Aurora PostgreSQL による Amazon RDS Proxy

Aurora PostgreSQL で RDS Proxy が使用可能なリージョンとエンジンのバージョンは以下のとおりです。

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
米国東部 (オハイオ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
米国東部 (バージニア北部)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
米国西部 (北カリフォルニア)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
米国西部（オレゴン）	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
アフリカ (ケープタウン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (香港)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (ハイデラバード)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (ジャカルタ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (メルボルン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック（ムンバイ）	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
アジアパシフィック (大阪)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (ソウル)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (シンガポール)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (シドニー)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
アジアパシフィック (東京)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
カナダ (中部)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
カナダ西部 (カルガリー)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
中国 (北京)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
中国 (寧夏)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
欧州 (フランクフルト)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
欧州 (アイルランド)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
欧州 (ロンドン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
欧州 (ミラノ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
欧州 (パリ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
欧州 (スペイン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
欧州 (ストックホルム)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
欧州 (チューリッヒ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
イスラエル (テルアビブ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
中東 (バーレーン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
中東 (アラブ首長国連邦)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降
南米 (サンパウロ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.4 以降	バージョン 12.8 以降	バージョン 11.9 とバージョン 11.13 以降

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
AWS GovCloud (米国東部)	–	–	–	–	–	–
AWS GovCloud (米国西部)	–	–	–	–	–	–

Secrets Manager 統合でサポートされているリージョンと Aurora DB エンジン

AWS Secrets Manager を使用すると、コード内のハードコードされた認証情報 (データベースパスワードを含む) を Secrets Manager への API コールで置き換えて、プログラムでシークレットを取得することができます。Secrets Manager の詳細については、AWS Secrets Manager ユーザーガイドを参照してください。

Amazon Aurora がマスターユーザーパスワードを管理することを Aurora DB クラスターの Secrets Manager で指定できます。Aurora はパスワードを生成し、Secrets Manager に保存して、定期的にローテーションします。詳細については、「Amazon Aurora および AWS Secrets Manager によるパスワード管理」を参照してください。

Secrets Manager の統合は、すべての Aurora DB エンジンとすべてのバージョンで使用できます。

Secrets Manager の統合は、以下を除くすべての AWS リージョン で利用可能です。

- カナダ西部 (カルガリー)
- AWS GovCloud (米国東部)
- AWS GovCloud (米国西部)

Aurora Serverless v2 でサポートされているリージョンと Aurora DB エンジン

Aurora Serverless v2 は、オンデマンドの自動スケーリング機能です。Amazon Aurora で、断続的または予測不能なワークロードを実行する費用対効果の高いアプローチとなるように設計されています。アプリケーションのニーズに応じて、容量の拡大または縮小を行います。Aurora Serverless v1 により、スケーリングは、より速く、詳細になります。Aurora Serverless v2 では、各クラスターに 1 つのライター DB インスタンスと複数のリーダー DB インスタンスを含めることができます。同じクラスター内で、Aurora Serverless v2 と従来のプロビジョン済み DB インスタンスを組み合わせることができます。詳細については、「Aurora Serverless v2 を使用する」を参照してください。

トピック

- Aurora MySQL での Aurora Serverless v2
- Aurora PostgreSQL での Aurora Serverless v2

Aurora MySQL での Aurora Serverless v2

Aurora MySQL で Aurora Serverless v2 が使用可能なリージョンとエンジンのバージョンは、以下のとおりです。

リージョン	Aurora MySQL バージョン 3
米国東部 (オハイオ)	バージョン 3.02.0 以降
米国東部 (バージニア北部)	バージョン 3.02.0 以降
米国西部 (北カリフォルニア)	バージョン 3.02.0 以降
米国西部 (オレゴン)	バージョン 3.02.0 以降
アフリカ (ケープタウン)	バージョン 3.02.0 以降
アジアパシフィック (香港)	バージョン 3.02.0 以降
アジアパシフィック (ハイデラバード)	バージョン 3.02.3 以降
アジアパシフィック (ジャカルタ)	バージョン 3.02.0 以降

リージョン	Aurora MySQL バージョン 3
アジアパシフィック (メルボルン)	バージョン 3.02.3 以降
アジアパシフィック (ムンバイ)	バージョン 3.02.0 以降
アジアパシフィック (大阪)	バージョン 3.02.0 以降
アジアパシフィック (ソウル)	バージョン 3.02.0 以降
アジアパシフィック (シンガポール)	バージョン 3.02.0 以降
アジアパシフィック (シドニー)	バージョン 3.02.0 以降
アジアパシフィック (東京)	バージョン 3.02.0 以降
カナダ (中部)	バージョン 3.02.0 以降
カナダ西部 (カルガリー)	バージョン 3.04.0、3.04.1、3.05.0、3.05.1 以降
中国 (北京)	バージョン 3.02.2 以降
中国 (寧夏)	バージョン 3.02.2 以降
欧州 (フランクフルト)	バージョン 3.02.0 以降
欧州 (アイルランド)	バージョン 3.02.0 以降
欧州 (ロンドン)	バージョン 3.02.0 以降
欧州 (ミラノ)	バージョン 3.02.0 以降
欧州 (パリ)	バージョン 3.02.0 以降
欧州 (スペイン)	バージョン 3.02.3 以降
欧州 (ストックホルム)	バージョン 3.02.0 以降
欧州 (チューリッヒ)	バージョン 3.02.3 以降
イスラエル (テルアビブ)	バージョン 3.02.3 以降、3.03.1 以降

リージョン	Aurora MySQL バージョン 3
中東 (バーレーン)	バージョン 3.02.0 以降
中東 (アラブ首長国連邦)	バージョン 3.02.3 以降
南米 (サンパウロ)	バージョン 3.02.0 以降
AWS GovCloud (米国東部)	バージョン 3.02.2 以降
AWS GovCloud (米国西部)	バージョン 3.02.2 以降

Aurora PostgreSQL での Aurora Serverless v2

Aurora PostgreSQL で Aurora Serverless v2 が使用可能なリージョンとエンジンのバージョンは、以下のとおりです。

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
米国東部 (オハイオ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
米国東部 (バージニア北部)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
米国西部 (北カリフォルニア)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
米国西部 (オレゴン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
アフリカ (ケープタウン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
アジアパシフィック (香港)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
アジアパシフィック (ハイデラバード)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.6 以降	バージョン 13.9 以降
アジアパシフィック (ジャカルタ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
アジアパシフィック (メルボルン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.6 以降	バージョン 13.9 以降
アジアパシフィック (ムンバイ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
アジアパシフィック (大阪)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
アジアパシフィック (ソウル)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
アジアパシフィック (シンガポール)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
アジアパシフィック (シドニー)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
アジアパシフィック (東京)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
カナダ (中部)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
カナダ西部 (カルガリー)	バージョン 16.1 以上	バージョン 15.3 以降	バージョン 14.6、14.8 以降	バージョン 13.9、13.11 以降
中国 (北京)	–	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
中国 (寧夏)	–	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
欧州 (フランクフルト)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
欧州 (アイルランド)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
欧州 (ロンドン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
欧州 (ミラノ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
欧州 (パリ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
欧州 (スペイン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.6 以降	バージョン 13.9 以降
欧州 (ストックホルム)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
欧州 (チューリッヒ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.6 以降	バージョン 13.9 以降
イスラエル (テルアビブ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.6 以降	バージョン 13.9 以降

リージョン	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
中東 (バーレーン)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
中東 (アラブ首長国連邦)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.6 以降	バージョン 13.9 以降
南米 (サンパウロ)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
AWS GovCloud (米国東部)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降
AWS GovCloud (米国西部)	バージョン 16.1 以上	バージョン 15.2 以降	バージョン 14.3 以降	バージョン 13.6 以降

Aurora Serverless v1 でサポートされているリージョンと Aurora DB エンジン

Aurora Serverless v1 は、オンデマンドの自動スケーリング機能です。Amazon Aurora で、断続的または予測不能なワークロードを実行する費用対効果の高いアプローチとなるように設計されています。各クラスターで単一の DB インスタンスを使用し、アプリケーションのニーズに応じて自動的に起動、シャットダウン、容量の増減を行います。詳細については、「Amazon Aurora Serverless v1 の使用」を参照してください。

トピック

- Aurora MySQL での Aurora Serverless v1
- Aurora PostgreSQL での Aurora Serverless v1

Aurora MySQL での Aurora Serverless v1

Aurora MySQL で Aurora Serverless v1 が使用可能なリージョンとエンジンのバージョンは、以下のとおりです。

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
米国東部 (オハイオ)	–	バージョン: 2.11.4
米国東部 (バージニア北部)	–	バージョン: 2.11.4
米国西部 (北カリフォルニア)	–	バージョン: 2.11.4
米国西部 (オレゴン)	–	バージョン: 2.11.4
アフリカ (ケープタウン)	–	–
アジアパシフィック (香港)	–	–
アジアパシフィック (ハイデラバード)	–	–
アジアパシフィック (ジャカルタ)	–	–
アジアパシフィック (メルボルン)	–	–
アジアパシフィック (ムンバイ)	–	バージョン: 2.11.4
アジアパシフィック (大阪)	–	–
アジアパシフィック (ソウル)	–	バージョン: 2.11.4
アジアパシフィック (シンガポール)	–	バージョン: 2.11.4
アジアパシフィック (シドニー)	–	バージョン: 2.11.4
アジアパシフィック (東京)	–	バージョン: 2.11.4
カナダ (中部)	–	バージョン: 2.11.4

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
カナダ西部 (カルガリー)	–	–
中国 (北京)	–	–
中国 (寧夏)	–	バージョン: 2.11.4
欧州 (フランクフルト)	–	バージョン: 2.11.4
欧州 (アイルランド)	–	バージョン: 2.11.4
欧州 (ロンドン)	–	バージョン: 2.11.4
欧州 (ミラノ)	–	–
欧州 (パリ)	–	バージョン: 2.11.4
欧州 (スペイン)	–	–
欧州 (ストックホルム)	–	–
欧州 (チューリッヒ)	–	–
イスラエル (テルアビブ)	–	–
中東 (バーレーン)	–	–
中東 (アラブ首長国連邦)	–	–
南米 (サンパウロ)	–	–
AWS GovCloud (米国東部)	–	–
AWS GovCloud (米国西部)	–	–

Aurora PostgreSQL での Aurora Serverless v1

Aurora PostgreSQL で Aurora Serverless v1 が使用可能なリージョンとエンジンのバージョンは、以下のとおりです。

リージョン	Aurora PostgreSQL 13
米国東部 (オハイオ)	バージョン 13.12
米国東部 (バージニア北部)	バージョン 13.12
米国西部 (北カリフォルニア)	バージョン 13.12
米国西部 (オレゴン)	バージョン 13.12
アフリカ (ケープタウン)	–
アジアパシフィック (香港)	–
アジアパシフィック (ハイデラバード)	–
アジアパシフィック (ジャカルタ)	–
アジアパシフィック (メルボルン)	–
アジアパシフィック (ムンバイ)	バージョン 13.12
アジアパシフィック (大阪)	–
アジアパシフィック (ソウル)	バージョン 13.12
アジアパシフィック (シンガポール)	バージョン 13.12
アジアパシフィック (シドニー)	バージョン 13.12
アジアパシフィック (東京)	バージョン 13.12
カナダ (中部)	バージョン 13.12
カナダ西部 (カルガリー)	–
中国 (北京)	–
中国 (寧夏)	–
欧州 (フランクフルト)	バージョン 13.12

リージョン	Aurora PostgreSQL 13
欧州 (アイルランド)	バージョン 13.12
欧州 (ロンドン)	バージョン 13.12
欧州 (ミラノ)	–
欧州 (パリ)	バージョン 13.12
欧州 (スペイン)	–
欧州 (ストックホルム)	–
欧州 (チューリッヒ)	–
イスラエル (テルアビブ)	–
中東 (バーレーン)	–
中東 (アラブ首長国連邦)	–
南米 (サンパウロ)	–
AWS GovCloud (米国東部)	–
AWS GovCloud (米国西部)	–

RDS Data API でサポートされているリージョンと Aurora DB エンジン

RDS Data API (Data API) は、Amazon Aurora DB クラスターへのウェブサービスインターフェイスを提供します。クライアントアプリケーションからデータベース接続を管理する代わりに、SQL コマンドを HTTPS エンドポイントに対して 実行できます。詳細については、「RDS Data API の使用」を参照してください。

Aurora MySQL では、Data API は Aurora Serverless v2 または のプロビジョニングされた DB クラスターではサポートされていません。

トピック

- Aurora MySQL Serverless v1 で Data API を使用する

- Aurora PostgreSQL Serverless v2 とプロビジョニングされた Data API
- Aurora PostgreSQL Serverless v1 を使用する Data API

Aurora MySQL Serverless v1 で Data API を使用する

Aurora MySQL Serverless v1 による Data API が使用可能なリージョンとエンジンのバージョンは以下のとおりです。

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
米国東部 (オハイオ)	–	バージョン 2.11.3
米国東部 (バージニア北部)	–	バージョン 2.11.3
米国西部 (北カリフォルニア)	–	バージョン 2.11.3
米国西部 (オレゴン)	–	バージョン 2.11.3
アフリカ (ケープタウン)	–	–
アジアパシフィック (香港)	–	–
アジアパシフィック (ハイデラバード)	–	–
アジアパシフィック (ジャカルタ)	–	–
アジアパシフィック (メルボルン)	–	–
アジアパシフィック (ムンバイ)	–	バージョン 2.11.3
アジアパシフィック (大阪)	–	–
アジアパシフィック (ソウル)	–	バージョン 2.11.3

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
アジアパシフィック (シンガポール)	–	バージョン 2.11.3
アジアパシフィック (シドニー)	–	バージョン 2.11.3
アジアパシフィック (東京)	–	バージョン 2.11.3
カナダ (中部)	–	バージョン 2.11.3
カナダ西部 (カルガリー)	–	–
中国 (北京)	–	–
中国 (寧夏)	–	バージョン 2.11.3
欧州 (フランクフルト)	–	バージョン 2.11.3
欧州 (アイルランド)	–	バージョン 2.11.3
欧州 (ロンドン)	–	バージョン 2.11.3
欧州 (ミラノ)	–	–
欧州 (パリ)	–	バージョン 2.11.3
欧州 (スペイン)	–	–
欧州 (ストックホルム)	–	–
欧州 (チューリッヒ)	–	–
イスラエル (テルアビブ)	–	–
中東 (バーレーン)	–	–
中東 (アラブ首長国連邦)	–	–
南米 (サンパウロ)	–	–

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
AWS GovCloud (米国東部)	–	–
AWS GovCloud (米国西部)	–	–

Aurora PostgreSQL Serverless v2 とプロビジョニングされた Data API

Aurora PostgreSQL Serverless v2 とプロビジョニングされた DB クラスターによって Data API が使用可能なリージョンとエンジンのバージョンは以下のとおりです。

リージョン	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
米国東部 (オハイオ)	–	–	–
米国東部 (バージニア北部)	バージョン 15.3 以降	バージョン 14.8 以降	バージョン 13.11 以降
米国西部 (北カリフォルニア)	–	–	–
米国西部 (オレゴン)	バージョン 15.3 以降	バージョン 14.8 以降	バージョン 13.11 以降
アフリカ (ケープタウン)	–	–	–
アジアパシフィック (香港)	–	–	–
アジアパシフィック (ハイデラバード)	–	–	–
アジアパシフィック (ジャカルタ)	–	–	–

リージョン	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
アジアパシフィック (メルボルン)	–	–	–
アジアパシフィック (ムンバイ)	–	–	–
アジアパシフィック (大阪)	–	–	–
アジアパシフィック (ソウル)	–	–	–
アジアパシフィック (シンガポール)	–	–	–
アジアパシフィック (シドニー)	–	–	–
アジアパシフィック (東京)	バージョン 15.3 以降	バージョン 14.8 以降	バージョン 13.11 以降
カナダ (中部)	–	–	–
カナダ西部 (カルガリー)	–	–	–
中国 (北京)	–	–	–
中国 (寧夏)	–	–	–
欧州 (フランクフルト)	バージョン 15.3 以降	バージョン 14.8 以降	バージョン 13.11 以降
欧州 (アイルランド)	–	–	–
欧州 (ロンドン)	–	–	–

リージョン	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
欧州 (ミラノ)	–	–	–
欧州 (パリ)	–	–	–
欧州 (スペイン)	–	–	–
欧州 (ストックホルム)	–	–	–
欧州 (チューリッヒ)	–	–	–
イスラエル (テルアビブ)	–	–	–
中東 (バーレーン)	–	–	–
中東 (アラブ首長国連邦)	–	–	–
南米 (サンパウロ)	–	–	–
AWS GovCloud (米国東部)	–	–	–
AWS GovCloud (米国西部)	–	–	–

Aurora PostgreSQL Serverless v1 を使用する Data API

Aurora PostgreSQL Serverless v1 による Data API が使用可能なリージョンとエンジンのバージョンは以下のとおりです。

リージョン	Aurora PostgreSQL 13	Aurora PostgreSQL 11
米国東部 (オハイオ)	バージョン 13.9	バージョン 11.18
米国東部 (バージニア北部)	バージョン 13.9	バージョン 11.18

リージョン	Aurora PostgreSQL 13	Aurora PostgreSQL 11
米国西部 (北カリフォルニア)	バージョン 13.9	バージョン 11.18
米国西部 (オレゴン)	バージョン 13.9	バージョン 11.18
アフリカ (ケープタウン)	–	–
アジアパシフィック (香港)	–	–
アジアパシフィック (ハイデラバード)	–	–
アジアパシフィック (ジャカルタ)	–	–
アジアパシフィック (メルボルン)	–	–
アジアパシフィック (ムンバイ)	バージョン 13.9	バージョン 11.18
アジアパシフィック (大阪)	–	–
アジアパシフィック (ソウル)	バージョン 13.9	バージョン 11.18
アジアパシフィック (シンガポール)	バージョン 13.9	バージョン 11.18
アジアパシフィック (シドニー)	バージョン 13.9	バージョン 11.18
アジアパシフィック (東京)	バージョン 13.9	バージョン 11.18
カナダ (中部)	バージョン 13.9	バージョン 11.18
中国 (北京)	–	–
中国 (寧夏)	–	–

リージョン	Aurora PostgreSQL 13	Aurora PostgreSQL 11
欧州 (フランクフルト)	バージョン 13.9	バージョン 11.18
欧州 (アイルランド)	バージョン 13.9	バージョン 11.18
欧州 (ロンドン)	バージョン 13.9	バージョン 11.18
欧州 (ミラノ)	–	–
欧州 (パリ)	バージョン 13.9	バージョン 11.18
欧州 (スペイン)	–	–
欧州 (ストックホルム)	–	–
欧州 (チューリッヒ)	–	–
イスラエル (テルアビブ)	–	–
中東 (バーレーン)	–	–
中東 (アラブ首長国連邦)	–	–
南米 (サンパウロ)	–	–
AWS GovCloud (米国東部)	–	–
AWS GovCloud (米国西部)	–	–

ダウンタイムなしのパッチ適用 (ZDP) でサポートされているリージョンと Aurora DB エンジン

Aurora DB クラスターのアップグレードを実行する場合、データベースがシャットダウンしているときにアップグレードすると、停止する可能性があります。デフォルトでは、データベースがビジー状態のときにアップグレードをスタートすると、DB クラスターが処理しているすべての接続とトランザクションが失われます。アップグレードを実行するためにデータベースがアイドル状態になるまで待機する場合は、長時間待機しなければならない場合があります。

ダウンタイムのないパッチ適用 (ZDP) 機能では、ベストエフォートに基づいて、Aurora アップグレード中のクライアント接続を維持するよう試みます。ZDP が正常に完了されると、アプリケーションのセッションが保持され、アップグレードの進行中にデータベースエンジンが再起動します。データベースエンジンの再起動により、数秒から約 1 分間スループットが低下する可能性があります。

Aurora MySQL のアップグレードで ZDP を利用できる条件とエンジンバージョンの詳細については、「ダウンタイムのないパッチ適用の使用」を参照してください。

Aurora PostgreSQL のアップグレードで ZDP を利用できる条件とエンジンバージョンの詳細については、「マイナーリリースのアップグレードとダウンタイムなしのパッチ適用プロセス」を参照してください。

Aurora エンジンのネイティブ機能でサポートされているリージョンと DB エンジン

また、Aurora データベースエンジンには、Aurora 専用の追加機能がサポートしています。エンジンネイティブの機能には、特定の Aurora DB のエンジン、バージョン、リージョンに対するサポートや特権に制限があるものがあります。

トピック

- Aurora MySQL のエンジンネイティブの機能
- Aurora PostgreSQL のエンジンネイティブの機能

Aurora MySQL のエンジンネイティブの機能

Aurora MySQL のエンジンネイティブの機能には次のようなものがあります。

- 高度な監査
- バックトラック
- 障害挿入クエリ
- クラスター内書き込み転送
- パラレルクエリ

Aurora PostgreSQL のエンジンネイティブの機能

Aurora PostgreSQL のエンジンネイティブの機能には次のようなものがあります。

- Babelfish
- 障害挿入クエリ
- クエリプラン管理

Amazon Aurora 接続管理

通常、Amazon Aurora は単一の DB インスタンスではなく、DB インスタンスのクラスターを必要とします。各接続は特定の DB インスタンスで処理されます。Aurora クラスターに接続すると、指定したホスト名とポートがエンドポイントと呼ばれる中間ハンドラーをポイントします。Aurora は、エンドポイントメカニズムを使用してこれらの接続を抽象化します。したがって、一部の DB インスタンスが使用できないときに、すべてのホスト名をハードコードしたり、接続のロードバランシングやルート再設定を行うために独自のロジックを記述したりする必要はありません。

Aurora の特定のタスクでは、インスタンスやインスタンスグループごとに異なるロールを実行します。例えば、プライマリインスタンスはすべてのデータ定義言語 (DDL) とデータ操作言語 (DML) のステートメントを処理します。最大 15 の Aurora レプリカで読み取り専用のクエリトラフィックを処理します。

エンドポイントを使用すると、ユースケースに基づいて各接続を対応するインスタンスまたはインスタンスグループにマッピングできます。例えば、DDL ステートメントを実行するには、プライマリインスタンスであるいずれのインスタンスにも接続できます。クエリを実行するには、リーダーエンドポイントに接続できます。Aurora はすべての Aurora レプリカ間で自動的にロードバランシングを実行します。容量や設定が異なる DB インスタンスで構成されるクラスターの場合は、DB インスタンスのサブセット別に関連付けたカスタムエンドポイントに接続できます。診断またはチューニングの場合は、特定のインスタンスエンドポイントに接続して、特定の DB インスタンスに関する詳細を調査できます。

トピック

- Aurora エンドポイントのタイプ
- Aurora クラスターのエンドポイントの表示
- クラスターエンドポイントの使用
- リーダーエンドポイントの使用
- カスタムエンドポイントの使用
- カスタムエンドポイントの作成
- カスタムエンドポイントの表示

- カスタムエンドポイントの編集
- カスタムエンドポイントの削除
- カスタムエンドポイントのエンドツーエンド AWS CLI の例
- インスタンスエンドポイントの使用
- Aurora エンドポイントでの高可用性の使用

Aurora エンドポイントのタイプ

エンドポイントは、ホストアドレスとポートを含む Aurora 固有の URL として表されます。Aurora DB クラスターでは、以下のタイプのエンドポイントを使用できます。

クラスターエンドポイント

Aurora DB クラスターのクラスターエンドポイント (またはライターエンドポイント) は、その DB クラスターの現在のプライマリ DB インスタンスに接続します。このエンドポイントは、DDL ステートメントなどの書き込みオペレーションを実行できる唯一のエンドポイントです。このため、初期にクラスターを設定する場合や、クラスターに含まれている DB インスタンスが 1 つのみである場合は、クラスターエンドポイントに接続します。

Aurora DB クラスターごとに 1 つのクラスターエンドポイントと 1 つのプライマリ DB インスタンスがあります。

クラスターエンドポイントは、DB クラスターに対するすべての書き込みオペレーション (挿入、更新、削除、DDL の変更など) で使用します。クラスターエンドポイントは、クエリなどの読み取りオペレーションでも使用できます。

クラスターエンドポイントは、DB クラスターへの読み取り/書き込み接続のフェイルオーバーサポートを提供します。DB クラスターの現在のプライマリ DB インスタンスが失敗した場合、Aurora は新しいプライマリ DB インスタンスに自動的にフェイルオーバーします。フェイルオーバー中、DB クラスターは、新しいプライマリ DB インスタンスからクラスターエンドポイントへの接続リクエストに継続して対応し、サービスの中断は最小限に抑えられます。

次の例では、Aurora MySQL DB クラスターのエンドポイントを示します。

```
mydbcluster.cluster-c7tj4example.us-east-1.rds.amazonaws.com:3306
```

リーダーエンドポイント

Aurora DB クラスターのリーダーエンドポイントは、その DB クラスターへの読み取り専用接続を負荷分散します。リーダーエンドポイントは、クエリなどの読み取りオペレーションで使用し

ます。これらのステートメントを読み取り専用の Aurora レプリカで処理することにより、このエンドポイントはプライマリインスタンスのオーバーヘッドを削減します。また、クラスター内の Aurora レプリカの数に合わせて、同時 SELECT クエリを処理する容量をスケールすることもできます。Aurora DB クラスターごとに 1 つのリーダーエンドポイントがあります。

クラスターに 1 つ以上の Aurora レプリカが含まれている場合、リーダーエンドポイントは Aurora レプリカ間で各接続リクエストを負荷分散します。その場合、そのセッションでは SELECT などの読み取り専用ステートメントのみを実行できます。クラスターにプライマリインスタンスのみが含まれていて、Aurora レプリカが含まれていない場合、リーダーエンドポイントはプライマリインスタンスに接続します。その場合は、エンドポイントを介して書き込みオペレ–ションを実行できます。

次の例では、Aurora MySQL DB クラスターのリーダーエンドポイントを示します。

```
mydbcluster.cluster-ro-c7tj4example.us-east-1.rds.amazonaws.com:3306
```

カスタムエンドポイント

Aurora クラスターのカスタムエンドポイントは、選択した DB インスタンスのセットを表します。カスタムエンドポイントに接続すると、Aurora は負荷分散を行い、グループ内のいずれかのインスタンスを選択して接続を処理します。ユーザーは、このエンドポイントで参照するインスタンスを定義し、エンドポイントの用途を決めます。

Aurora DB クラスターには、ユーザーが作成するまで、カスタムエンドポイントは存在しません。Aurora のプロビジョニングされたクラスターまたは Aurora Serverless v2 クラスターごとに最大 5 つのカスタムエンドポイントを作成できます。Aurora Serverless v1 クラスターにカスタムエンドポイントを使用することはできません。

カスタムエンドポイントでは、DB インスタンスの読み取り専用機能や読み取り/書き込み機能とは異なる基準に従ってデータベース接続を負荷分散できます。例えば、特定の AWS インスタンスクラスや特定の DB パラメータグループを使用するインスタンスに接続するカスタムエンドポイントを定義できます。次に、このカスタムエンドポイントを特定のユーザーグループに知らせることができます。例えば、社内ユーザーをレポート生成やアドホック (1 回だけの) クエリ実行用の低容量インスタンスに振り向けたり、本番稼働用トラフィックを高容量インスタンスに振り向けたりすることができます。

カスタムエンドポイントが関連付けられているいずれの DB インスタンスも接続先となるため、グループ内のすべての DB インスタンス間で同様の特性を共有することをお勧めします。これにより、このエンドポイントに接続するすべてのユーザー間で、パフォーマンス、メモリ容量などが一貫したものになります。

この機能は、クラスター内のすべての Aurora レプリカを同一に保つことが現実的ではない特種なワークロードを扱う上級ユーザー向けです。カスタムエンドポイントを使用すると、各接続に使用される DB インスタンスの容量を予測できます。カスタムエンドポイントを使用する場合は、通常、そのクラスターのリーダーエンドポイントは使用しません。

次の例は、Aurora MySQL DB クラスター内の DB インスタンスのカスタムエンドポイントを示しています。

```
myendpoint.cluster-custom-c7tj4example.us-east-1.rds.amazonaws.com:3306
```

インスタンスエンドポイント

インスタンスエンドポイントは、Aurora クラスター内の特定の DB インスタンスに接続します。DB クラスターの各 DB インスタンスには、独自のインスタンスエンドポイントがあります。したがって、DB クラスター内の現在のプライマリ DB インスタンスに 1 つのインスタンスエンドポイントがあり、DB クラスター内の Aurora レプリカごとに 1 つのインスタンスエンドポイントがあります。

インスタンスエンドポイントは、クラスターエンドポイントやリーダーエンドポイントの使用が適切でないシナリオ向けに、DB クラスターへの接続の直接制御を提供します。例えば、ワークロードタイプに基づき、さらにきめ細かいロードバランシングがアプリケーションに必要になる場合があります。この場合、DB クラスター内の別の Aurora レプリカに接続するように複数のクライアントを設定して、読み取りワークロードを分散させることができます。Aurora PostgreSQL のフェイルオーバー後に接続速度を向上させるインスタンスエンドポイントを使用する例については、「Amazon Aurora PostgreSQL による高速フェイルオーバー」を参照してください。Aurora MySQL のフェイルオーバー後にインスタンスエンドポイントを使用して接続速度を向上させる例については、「MariaDB Connector/J failover support - case Amazon Aurora」を参照してください。

次の例では、Aurora MySQL DB クラスターの DB インスタンスのインスタンスエンドポイントを示します。

```
mydbinstance.c7tj4example.us-east-1.rds.amazonaws.com:3306
```

Aurora クラスターのエンドポイントの表示

AWS Management Console では、クラスターエンドポイント、リーダーエンドポイント、およびカスタムエンドポイントが、各クラスターの詳細ページに表示されます。インスタンスエンドポイントは、各インスタンスの詳細ページに表示されます。接続する場合、この詳細ページに表示されているエンドポイント名の後にコロン (:) および該当するポート番号を付加する必要があります。

AWS CLI では、ライターエンドポイント、リーダーエンドポイント、およびカスタムエンドポイントが describe-db-clusters コマンドの出力に表示されます。例えば、次のコマンドは、現在の AWS リージョンにあるすべてのクラスターのエンドポイント属性を表示します。

```
aws rds describe-db-clusters --query '*[].
{Endpoint:Endpoint,ReaderEndpoint:ReaderEndpoint,CustomEndpoints:CustomEndpoints}'
```

Amazon RDS API では、DescribeDBClusterEndpoints 関数を呼び出してエンドポイントを取得します。

クラスターエンドポイントの使用

Aurora クラスターごとに組み込まれている単一のクラスターエンドポイントの名前と他の属性は Aurora で管理されるため、この種のエンドポイントをユーザーが作成、削除、または変更することはできません。

クラスターエンドポイントは、クラスターの管理、抽出/変換/ロード (ETL) オペレーションの実行、またはアプリケーションの開発やテストに使用します。クラスターエンドポイントはクラスターのプライマリインスタンスに接続します。プライマリインスタンスは、テーブルとインデックスを作成し、INSERT ステートメントや他の DDL/DML オペレーションを実行できる唯一の DB インスタンスです。

フェイルオーバー機構によって別の DB インスタンスがクラスターの読み取り/書き込みのプライマリインスタンスに昇格されると、クラスターエンドポイントがポイントする物理 IP アドレスは変わります。何らかの形式の接続プールや他の多重化を使用している場合は、キャッシュされた DNS 情報の有効時間をフラッシュまたは削減する必要があります。これにより、フェイルオーバー後に使用不可または読み取り専用になった DB インスタンスに読み取り/書き込み接続を試行できないようにします。

リーダーエンドポイントの使用

リーダーエンドポイントは、Aurora クラスターの読み取り専用接続に使用します。このエンドポイントでは、ロードバランシング機構を使用して、クラスターによるクエリが集中的に行われるワークロードの処理を促進します。リーダーエンドポイントは、クラスターに対してレポートや他の読み取り専用のオペレーションを実行するアプリケーションに指定します。

リーダーエンドポイントは、Aurora DB クラスター内の利用可能な Aurora レプリカへの接続を負荷分散します。個別のクエリは負荷分散しません。各クエリを負荷分散して DB クラスターの読み取りワークロードを配信する場合は、クエリごとにリーダーエンドポイントへの新しい接続を開きます。

Aurora クラスターごとに組み込まれている単一のリーダーエンドポイントの名前や他の属性は Aurora で管理されます。ユーザーが、この種のエンドポイントを作成、削除、または変更することはできません。

クラスターにプライマリインスタンスのみが含まれていて、Aurora レプリカが含まれていない場合、リーダーエンドポイントはプライマリインスタンスに接続します。その場合、このエンドポイントを介して書き込みオペレーションを実行できます。

> **Tip**
>
> RDS Proxy を使用して、Aurora クラスターの追加の読み取り専用エンドポイントを作成できます。このようなエンドポイントは、Aurora リーダーエンドポイントと同じ種類の負荷分散を実行します。リーダーインスタンスが使用できなくなった場合、アプリケーションは Aurora リーダーエンドポイントよりもプロキシエンドポイントにすばやく再接続できます。プロキシエンドポイントは、多重化などの他のプロキシ機能を利用することもできます。詳細については、「Aurora クラスターでのリーダーエンドポイントの使用」を参照してください。

カスタムエンドポイントの使用

カスタムエンドポイントでは、容量や構成設定が異なる DB インスタンスがクラスターに含まれている場合に接続管理を簡素化します。

以前は、CNAMES 機構を使用して各自のドメインからドメインネームサービス (DNS) のエイリアスをセットアップして同様の結果を達成していました。カスタムエンドポイントを使用すると、クラスターの増大や縮小に伴って CNAME レコードを更新する必要がなくなります。カスタムエンドポイントでは、暗号化された Transport Layer Security/Secure Sockets Layer (TLS/SSL) 接続を使用することもできます。

異なる目的ごとに 1 つの DB インスタンスを使用し、そのインスタンスエンドポイントに接続する代わりに、複数の専用の DB インスタンスを持つことができます。この場合、グループごとに独自のカスタムエンドポイントを使用します。これにより、Aurora は、レポートの作成、本番稼働用クエリや内部クエリの処理など、タスク別のすべてのインスタンスをまたいでロードバランシングを実行できます。カスタムエンドポイントは、クラスター内の DB インスタンスのグループごとにロードバランシングと高可用性を提供します。グループ内のいずれかの DB インスタンスが使用不可になると、Aurora は以降のカスタムエンドポイント接続を、同じエンドポイントに関連付けられている他のいずれかの DB インスタンスに振り向けます。

トピック

- カスタムエンドポイントのプロパティの指定
- カスタムエンドポイントのメンバーシップルール
- カスタムエンドポイントの管理

カスタムエンドポイントのプロパティの指定

カスタムエンドポイント名の最大長は 63 文字です。名前の形式は次のとおりです。

```
endpoint_name.cluster-custom-customer_DNS_identifier.AWS_Region.rds.amazonaws.com
```

同じカスタムエンドポイント名を同じ AWS リージョン の複数のクラスターで再利用することはできません。顧客の DNS 識別子は、特定の AWS リージョン の AWS アカウント に関連付けられた一意識別子です。

各カスタムエンドポイントの対応するタイプにより、このエンドポイントに関連付けることができる DB インスタンスが決まります。現在、タイプは READER、WRITER、ANY のいずれかです。カスタムエンドポイントのタイプには、以下の考慮事項が適用されます。

- カスタムエンドポイントタイプを AWS Management Console で選択することはできません。AWS Management Console で作成するすべてのカスタムエンドポイントは ANY タイプになります。

  AWS CLI または Amazon RDS API を使用してカスタムエンドポイントタイプを設定および変更できます。
- リーダー DB インスタンスのみが、READER カスタムエンドポイントの一部となることができます。
- リーダー DB インスタンスとライター DB インスタンスの両方が、ANY カスタムエンドポイントの一部となることができます。Aurora はクラスターエンドポイントに接続を指示し、ANY のタイプを関連付けられた DB インスタンスのいずれかに同じ確率で割り当てます。ANY タイプは、任意のレプリケーショントポロジを使用するクラスターに適用されます。
- 作成しようとしているカスタムエンドポイントのタイプがクラスターのレプリケーション設定に適切なものではない場合、Aurora はエラーを返します。

カスタムエンドポイントのメンバーシップルール

DB インスタンスをカスタムエンドポイントに追加したり、カスタムエンドポイントから削除したりすると、この DB インスタンスへの既存の接続はアクティブのまま残ります。

カスタムエンドポイントに含めたり、カスタムエンドポイントから除外したりする DB インスタンスのリストを定義できます。これらのリストは、それぞれ静的リストおよび除外リストと呼ばれます。包含/除外機構を使用して、さらに DB インスタンスのグループを細分化したり、カスタムエンドポイントのセットがクラスターのすべての DB インスタンスをカバーしていることを確認したりできます。各カスタムエンドポイントには、これらのリストタイプの 1 つのみを含めることができます。

AWS Management Console で次の操作を行います。

- 選択は、[Attach future instances added to this cluster] (今後追加されるインスタンスをこのクラスターにアタッチする) チェックボックスによって表されます。このチェックボックスをオフのままにすると、カスタムエンドポイントはページで指定された DB インスタンスのみを含む静的リストを使用します。このチェックボックスをオンにすると、カスタムエンドポイントは除外リストを使用します。この場合、カスタムエンドポイントは、ページで選択されなかったものを除いて、クラスター内のすべての DB インスタンス (今後追加するものも含む) を表します。
- コンソールではエンドポイントタイプを指定できません。コンソールを使用して作成されたカスタムエンドポイントは、すべて `ANY` タイプです。

  そのため、フェイルオーバーやプロモーションにより DB インスタンスのロールがライターとリーダーの間で変更されても、Aurora はカスタムエンドポイントのメンバーシップを変更しません。

AWS CLI および Amazon RDS API では:

- エンドポイントタイプを指定できます。そのため、エンドポイントタイプを `READER` または `WRITER` に設定すると、フェイルオーバーやプロモーション時にエンドポイントのメンバーシップが自動的に調整されます。

  例えば、タイプ `READER` のカスタムエンドポイントに Aurora レプリカが含まれ、その後、ライターインスタンスに昇格されたとします。新しいライターインスタンスは、カスタムエンドポイントの一部ではなくなります。
- ロールを変更した後に、個々のメンバーをリストに追加したり、リストから削除したりできます。modify-db-cluster-endpoint AWS CLI コマンドまたは ModifyDBClusterEndpoint API オペレーションを使用してください。

DB インスタンスは、複数のカスタムエンドポイントに関連付けることができます。例えば、ユーザーが新しい DB インスタンスをクラスターに追加するか、Aurora が Autoscaling 機構を通じて DB インスタンスを自動的に追加するとします。このような場合、DB インスタンスは該当するすべてのカスタムエンドポイントに追加されます。どのエンドポイントに DB インスタンスが追加されるかは、カスタムエンドポイントのタイプである `READER`、`WRITER`、または `ANY` と、各エンドポイントに定義されている静的リストや除外リストに基づきます。例えば、エンドポイントに DB インスタンスの静的リストが含まれている場合、新しく追加された Aurora レプリカはこのエンドポイントに追加されません。逆に、エンドポイントに除外リストが含まれている場合、新しく追加された Aurora レプリカは、この除外リストに名前が存在せず、ロールがカスタムエンドポイントのタイプと一致すれば、エンドポイントに追加されます。

Aurora レプリカは、使用不可になっても、カスタムエンドポイントに関連付けられたままになります。例えば、異常、停止、再起動などに伴って、カスタムエンドポイントの一部として残ります。ただし、レプリカが再度使用可能になるまでは、これらのエンドポイントを通じてレプリカに接続することはできません。

カスタムエンドポイントの管理

新しく作成された Aurora クラスターにはカスタムエンドポイントがないため、これらのオブジェクトを自分で作成して管理する必要があります。これを行うには、AWS Management Console、AWS CLI、または Amazon RDS API を使用します。

> **Note**
>
> スナップショットから復元した Aurora クラスターのカスタムエンドポイントも作成して管理する必要があります。カスタムエンドポイントはスナップショットに含まれていません。カスタムエンドポイントは、復元後に再度作成します。復元したクラスターが元のクラスターと同じリージョン内にある場合は、新しいエンドポイント名を選択します。

AWS Management Console からカスタムエンドポイントを使用するには、Aurora クラスターの詳細ページに移動し、[カスタムエンドポイント] セクションの制御を使用します。

AWS CLI からカスタムエンドポイントを使用するには、以下のオペレーションを使用できます。

- create-db-cluster-endpoint
- describe-db-cluster-endpoints
- modify-db-cluster-endpoint

- delete-db-cluster-endpoint

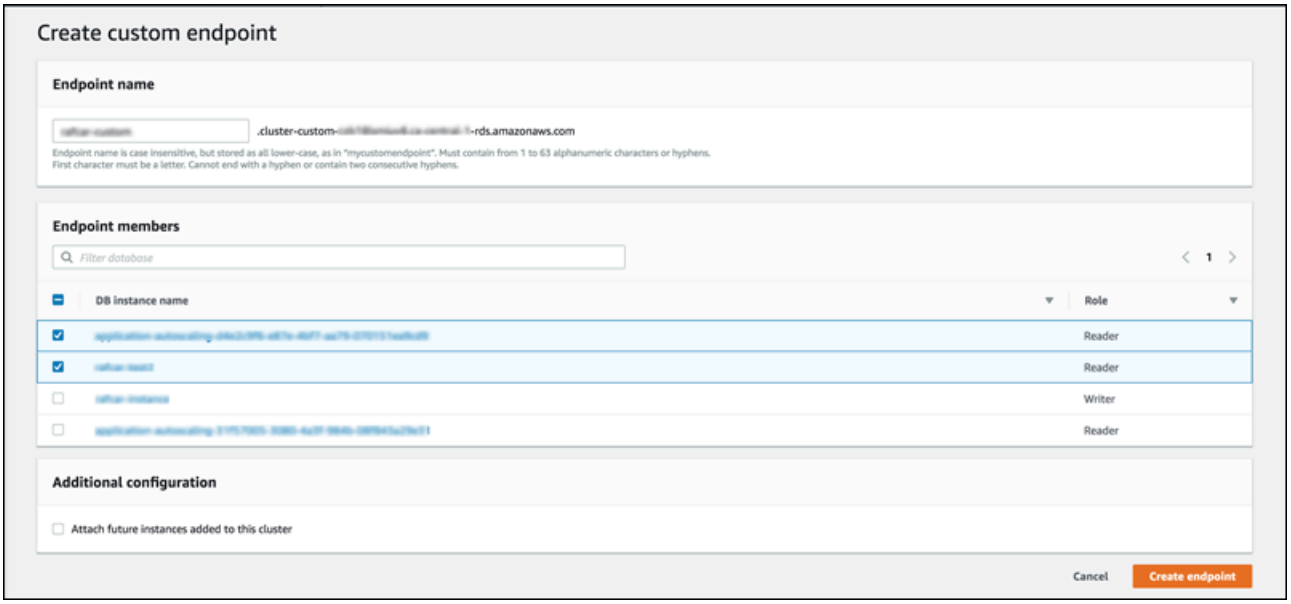
Amazon RDS API からカスタムエンドポイントを使用するには、以下の関数を使用できます。

- CreateDBClusterEndpoint
- DescribeDBClusterEndpoints
- ModifyDBClusterEndpoint
- DeleteDBClusterEndpoint

カスタムエンドポイントの作成

コンソール

AWS Management Console でカスタムエンドポイントを作成するには、クラスターの詳細ページに移動し、[エンドポイント] セクションの [Create custom endpoint] アクションを選択します。カスタムエンドポイントの名前を選択します。名前はユーザー ID およびリージョンで一意なものを使用します。クラスターが拡張しても変更されない DB インスタンスのリストを選択するには、[Attach future instances added to this cluster (今後追加されるインスタンスをこのクラスターにアタッチ)] をオフのままにします。このチェックボックスをオンにすると、カスタムエンドポイントは、クラスターに新しく追加されたインスタンスを動的に追加します。



カスタムエンドポイントタイプの ANY または READER を AWS Management Console で選択することはできません。AWS Management Console で作成するすべてのカスタムエンドポイントは ANY タイプになります。

AWS CLI

AWS CLI でカスタムエンドポイントを作成するには、create-db-cluster-endpoint コマンドを実行します。

次のコマンドは、特定のクラスターにアタッチされるカスタムエンドポイントを作成します。当初、エンドポイントはクラスター内のすべての Aurora レプリカインスタンスに関連付けられます。後続のコマンドで、エンドポイントをクラスター内の特定の DB インスタンスのセットに関連付けます。

Linux、macOS、Unix の場合:

```
aws rds create-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample \
  --endpoint-type reader \
  --db-cluster-identifier cluster_id

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample \
  --static-members instance_name_1 instance_name_2
```

Windows の場合:

```
aws rds create-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample ^
  --endpoint-type reader ^
  --db-cluster-identifier cluster_id

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier custom-endpoint-
doc-sample ^
  --static-members instance_name_1 instance_name_2
```

RDS API

RDS API でカスタムエンドポイントを作成するには、CreateDBClusterEndpoint オペレーションを実行します。

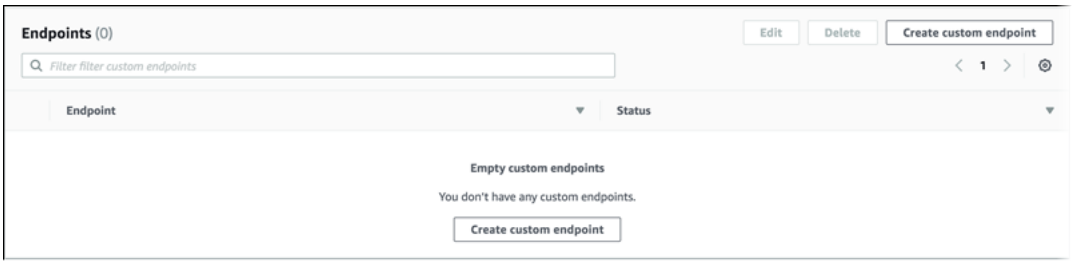
カスタムエンドポイントの表示

コンソール

AWS Management Console でカスタムエンドポイントを表示するには、クラスターの詳細ページに移動し、[エンドポイント] セクションを参照します。このセクションには、カスタムエンドポイント

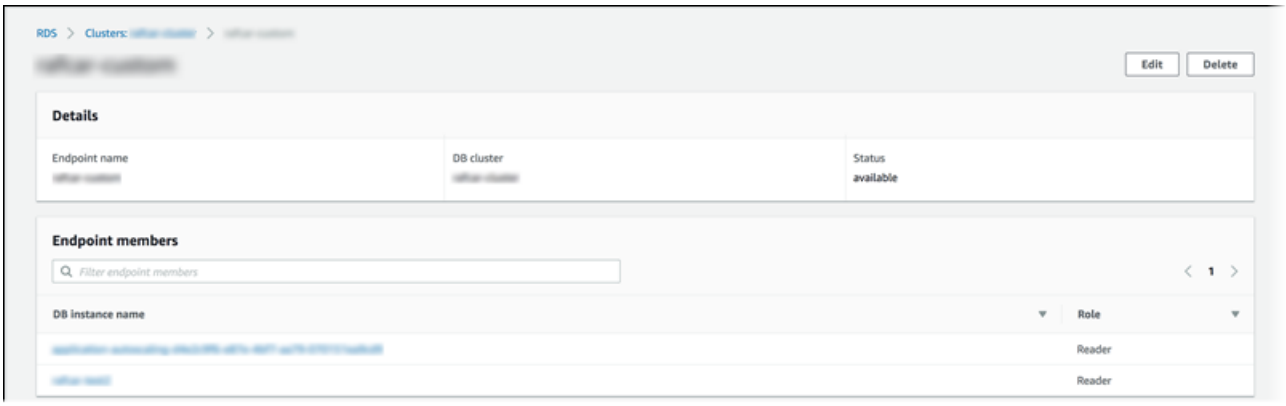
に関する情報のみが表示されます。組み込みエンドポイントに関する詳細は、メインの [詳細] セクションに表示されます。特定のカスタムエンドポイントに関する詳細を表示するには、その名前を選択してエンドポイントの詳細ページを表示します。

次のスクリーンショットは、Aurora クラスターのカスタムエンドポイントのリストが当初は空であることを示しています。



クラスターのカスタムエンドポイントを作成すると、[Endpoints (エンドポイント)] セクションに表示されます。

クリックを繰り返して詳細ページまで移動すると、エンドポイントに現在関連付けられている DB インスタンスが表示されます。



クラスターに追加された新しい DB インスタンスがエンドポイントにも自動的に追加されるかどうかの詳細を表示するには、エンドポイントの [Edit] (編集) ページを開きます。

AWS CLI

AWS CLI でカスタムエンドポイントを表示するには、describe-db-cluster-endpoints コマンドを実行します。

次のコマンドは、特定のリージョンの特定のクラスターに関連付けられているカスタムエンドポイントを表示します。出力には、組み込みエンドポイントとカスタムエンドポイントの両方が含まれています。

Linux、macOS、Unix の場合:

```
aws rds describe-db-cluster-endpoints --region region_name \
  --db-cluster-identifier cluster_id
```

Windows の場合:

```
aws rds describe-db-cluster-endpoints --region region_name ^
  --db-cluster-identifier cluster_id
```

describe-db-cluster-endpoints コマンドの出力例は以下のとおりです。EndpointType の WRITER または READER は、クラスターの組み込みの読み取り/書き込みエンドポイントと読み取り専用エンドポイントを示します。EndpointType の CUSTOM は、ユーザーが作成し、関連する DB インスタンスを選択した、エンドポイントを示します。エンドポイントの 1 つに含まれている空ではない StaticMembers フィールドは、エンドポイントが DB インスタンスの厳密なセットに関連付けられることを示します。他のエンドポイントの空ではない ExcludedMembers フィールドは、ExcludedMembers にリストアップされているものを除いたすべての DB インスタンスにエンドポイントが関連付けられることを示します。この 2 番目の種類のカスタムエンドポイントは、クラスターに追加された新しいインスタンスを包含して増大します。

```
{
 "DBClusterEndpoints": [
  {
   "Endpoint": "custom-endpoint-demo.cluster-c7tj4example.ca-
central-1.rds.amazonaws.com",
   "Status": "available",
   "DBClusterIdentifier": "custom-endpoint-demo",
   "EndpointType": "WRITER"
  },
  {
   "Endpoint": "custom-endpoint-demo.cluster-ro-c7tj4example.ca-
central-1.rds.amazonaws.com",
   "Status": "available",
   "DBClusterIdentifier": "custom-endpoint-demo",
   "EndpointType": "READER"
  },
  {
```

```
   "CustomEndpointType": "ANY",
   "DBClusterEndpointIdentifier": "powers-of-2",
   "ExcludedMembers": [],
   "DBClusterIdentifier": "custom-endpoint-demo",
   "Status": "available",
   "EndpointType": "CUSTOM",
   "Endpoint": "powers-of-2.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
   "StaticMembers": [
     "custom-endpoint-demo-04",
     "custom-endpoint-demo-08",
     "custom-endpoint-demo-01",
     "custom-endpoint-demo-02"
   ],
   "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
   "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:powers-of-2"
  },
  {
   "CustomEndpointType": "ANY",
   "DBClusterEndpointIdentifier": "eight-and-higher",
   "ExcludedMembers": [
     "custom-endpoint-demo-04",
     "custom-endpoint-demo-02",
     "custom-endpoint-demo-07",
     "custom-endpoint-demo-05",
     "custom-endpoint-demo-03",
     "custom-endpoint-demo-06",
     "custom-endpoint-demo-01"
   ],
   "DBClusterIdentifier": "custom-endpoint-demo",
   "Status": "available",
   "EndpointType": "CUSTOM",
   "Endpoint": "eight-and-higher.cluster-custom-123456789012.ca-
central-1.rds.amazonaws.com",
   "StaticMembers": [],
   "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHYQKFU6J6NV5FHU",
   "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:eight-and-higher"
  }
 ]
```

```
}
```

RDS API

RDS API でカスタムエンドポイントを表示するには、DescribeDBClusterEndpoints.html オペレーションを実行します。

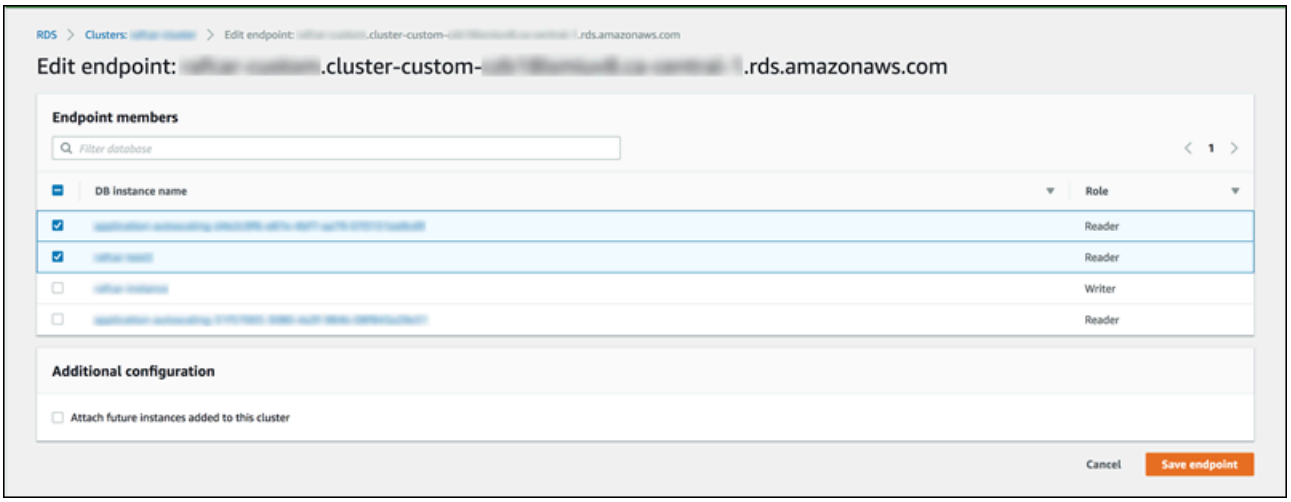
カスタムエンドポイントの編集

カスタムエンドポイントのプロパティを編集して、エンドポイントに関連付けられている DB インスタンスを変更できます。静的リストと除外リストの間でエンドポイントを変更することもできます。これらのエンドポイントプロパティの詳細については、「カスタムエンドポイントのメンバーシップルール」を参照してください。

編集アクションによる変更の進行中も、カスタムエンドポイントへの接続やカスタムエンドポイントの使用を続行できます。

コンソール

AWS Management Console でカスタムエンドポイントを編集するには、クラスターの詳細ページでエンドポイントを選択するか、エンドポイントの詳細ページを表示して、[編集] アクションを選択します。



AWS CLI

AWS CLI でカスタムエンドポイントを編集するには、modify-db-cluster-endpoint コマンドを実行します。

以下のコマンドは、カスタムエンドポイントに適用される DB インスタンスのセットを変更し、必要に応じて静的リストまたは除外リストの動作間を切り替えます。`--static-members` パラメータと

`--excluded-members` パラメータは、スペースで区切られた DB インスタンス識別子のリストを使用します。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 \
  --static-members db-instance-id-1 db-instance-id-2 db-instance-id-3 \
  --region region_name

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 \
  --excluded-members db-instance-id-4 db-instance-id-5 \
  --region region_name
```

Windows の場合:

```
aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 ^
  --static-members db-instance-id-1 db-instance-id-2 db-instance-id-3 ^
  --region region_name

aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier my-custom-endpoint
 ^
  --excluded-members db-instance-id-4 db-instance-id-5 ^
  --region region_name
```

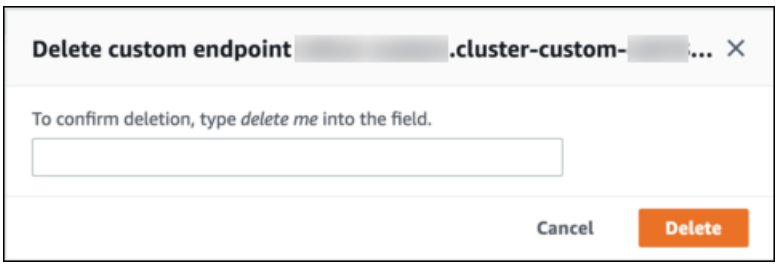
RDS API

RDS API でカスタムエンドポイントを編集するには、ModifyDBClusterEndpoint.html オペレーションを実行します。

カスタムエンドポイントの削除

コンソール

AWS Management Console でカスタムエンドポイントを削除するには、クラスターの詳細ページに移動し、該当するカスタムエンドポイントを選択して、[削除] アクションを選択します。



AWS CLI

AWS CLI でカスタムエンドポイントを削除するには、delete-db-cluster-endpoint コマンドを実行します。

次のコマンドは、カスタムエンドポイントを削除します。関連付けられたクラスターを指定する必要はありませんが、リージョンを指定する必要があります。

Linux、macOS、Unix の場合:

```
aws rds delete-db-cluster-endpoint --db-cluster-endpoint-identifier custom-end-point-id
 \
  --region region_name
```

Windows の場合:

```
aws rds delete-db-cluster-endpoint --db-cluster-endpoint-identifier custom-end-point-id
 ^
  --region region_name
```

RDS API

RDS API でカスタムエンドポイントを削除するには、DeleteDBClusterEndpoint オペレーションを実行します。

カスタムエンドポイントのエンドツーエンド AWS CLI の例

次のチュートリアルでは、AWS CLI による Unix シェル構文の例を使用して、いくつかの「小さい」DB インスタンスと少数の「大きい」DB インスタンスで構成されたクラスターを定義し、カスタムエンドポイントを作成して DB インスタンスの各セットに接続できることを示します。自分のシステムで同様のコマンドを実行するには、リージョン、サブネットグループ、VPC セキュリティグループなどのパラメータに独自の値を指定するために、Aurora クラスターと AWS CLI の基本的な使用法に精通している必要があります。

この例では、初期のセットアップステップである、Aurora クラスターの作成とクラスターへの DB インスタンスの追加を示します。これは異種クラスターです。つまり、すべての DB インスタンスが同じ容量を持っているわけではありません。ほとんどのインスタンスは AWS インスタンスクラス db.r4.4xlarge を使用しますが、最後の 2 つの DB インスタンスは db.r4.16xlarge を使用します。以下の各 create-db-instance コマンドでは、出力を画面に表示し、JSON のコピーをファイルに保存して後で検査できるようにします。

```
aws rds create-db-cluster --db-cluster-identifier custom-endpoint-demo --engine aurora-
mysql \
     --engine-version 8.0.mysql_aurora.3.02.0 --master-username $MASTER_USER --manage-
master-user-password \
     --db-subnet-group-name $SUBNET_GROUP  --vpc-security-group-ids $VPC_SECURITY_GROUP
 \
     --region $REGION

for i in 01 02 03 04 05 06 07 08
do
  aws rds create-db-instance --db-instance-identifier custom-endpoint-demo-${i} \
     --engine aurora --db-cluster-identifier custom-endpoint-demo --db-instance-class
 db.r4.4xlarge \
     --region $REGION \
     | tee custom-endpoint-demo-${i}.json
done

for i in 09 10
do
  aws rds create-db-instance --db-instance-identifier custom-endpoint-demo-${i} \
     --engine aurora --db-cluster-identifier custom-endpoint-demo --db-instance-class
 db.r4.16xlarge \
     --region $REGION \
     | tee custom-endpoint-demo-${i}.json
done
```

より大きいインスタンスは特種なレポートクエリ用に予約されています。これらのインスタンスがプライマリインスタンスに昇格されないように、次の例では昇格階層を最低の優先度に変更します。この例では、マスターユーザーパスワードを生成して Secrets Manager で管理する --manage-master-user-password オプションを指定しています。詳細については、「Amazon Aurora および AWS Secrets Manager によるパスワード管理」を参照してください。または、--master-password オプションを使用して、自分でパスワードを指定して管理することもできます。

```
for i in 09 10
```

```
do
  aws rds modify-db-instance --db-instance-identifier custom-endpoint-demo-${i} \
    --region $REGION --promotion-tier 15
done
```

例えば、2 つの「大きい」インスタンスを、リソースを最も集中的に使用するクエリ専用にするとします。これを行うには、初期にカスタムの読み取り専用エンドポイントを作成します。次に、このエンドポイントから上記の DB インスタンスにのみ接続するように、メンバーの静的なリストを追加できます。これらのインスタンスは、既に最低の昇格階層にあるため、いずれのインスタンスもプライマリインスタンスに昇格される可能性はありません。いずれかがプライマリインスタンスに昇格されたとしても、タイプとして READER ではなく ANY を指定しているため、このエンドポイントを通じてインスタンスに到達することはできなくなります。

次の例では、エンドポイントの作成コマンドと変更コマンドを示します。JSON 出力例は、カスタムエンドポイントの当初の状態と変更後の状態を示しています。

```
$ aws rds create-db-cluster-endpoint --region $REGION \
    --db-cluster-identifier custom-endpoint-demo \
    --db-cluster-endpoint-identifier big-instances --endpoint-type reader
{
    "EndpointType": "CUSTOM",
    "Endpoint": "big-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "DBClusterEndpointIdentifier": "big-instances",
    "DBClusterIdentifier": "custom-endpoint-demo",
    "StaticMembers": [],
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
    "ExcludedMembers": [],
    "CustomEndpointType": "READER",
    "Status": "creating",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:big-instances"
}

$ aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier big-instances \
    --static-members custom-endpoint-demo-09 custom-endpoint-demo-10 --region $REGION
{
    "EndpointType": "CUSTOM",
    "ExcludedMembers": [],
    "DBClusterEndpointIdentifier": "big-instances",
```

```
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
    "CustomEndpointType": "READER",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:big-instances",
    "StaticMembers": [
        "custom-endpoint-demo-10",
        "custom-endpoint-demo-09"
    ],
    "Status": "modifying",
    "Endpoint": "big-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "DBClusterIdentifier": "custom-endpoint-demo"
}
```

クラスターのデフォルトの READER エンドポイントは、「小さい」DB インスタンスまたは「大きい」DB インスタンスに接続できます。クラスターがビジー状態になると、クエリのパフォーマンスとスケーラビリティは予測不可能になります。DB インスタンスのセット間でワークロードを明確に分けるには、デフォルトの READER エンドポイントを無視して、他のすべての DB インスタンスに接続する 2 番目のカスタムエンドポイントを作成できます。次の例では、これを行うために、カスタムエンドポイントを作成して除外リストを追加します。後でクラスターに追加した他の DB インスタンスは、このエンドポイントに自動的に追加されます。ANY タイプでは、このエンドポイントに合計 8 個のインスタンス (プライマリインスタンスと 7 個の Aurora レプリカ) が関連付けられます。この例で READER タイプを使用したとすると、カスタムエンドポイントは 7 個のみの Aurora レプリカに関連付けられます。

```
$ aws rds create-db-cluster-endpoint --region $REGION --db-cluster-identifier custom-
endpoint-demo \
    --db-cluster-endpoint-identifier small-instances --endpoint-type any
{
    "Status": "creating",
    "DBClusterEndpointIdentifier": "small-instances",
    "CustomEndpointType": "ANY",
    "EndpointType": "CUSTOM",
    "Endpoint": "small-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "StaticMembers": [],
    "ExcludedMembers": [],
    "DBClusterIdentifier": "custom-endpoint-demo",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:small-instances",
```

```
    "DBClusterEndpointResourceIdentifier": "cluster-
endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY"
}

$ aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier small-instances \
    --excluded-members custom-endpoint-demo-09 custom-endpoint-demo-10 --region $REGION
{
    "DBClusterEndpointIdentifier": "small-instances",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:c7tj4example:cluster-
endpoint:small-instances",
    "DBClusterEndpointResourceIdentifier": "cluster-
endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY",
    "CustomEndpointType": "ANY",
    "Endpoint": "small-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "EndpointType": "CUSTOM",
    "ExcludedMembers": [
        "custom-endpoint-demo-09",
        "custom-endpoint-demo-10"
    ],
    "StaticMembers": [],
    "DBClusterIdentifier": "custom-endpoint-demo",
    "Status": "modifying"
}
```

次の例では、このクラスターのエンドポイントの状態をチェックします。クラスターには、まだ元のクラスターエンドポイントがあります。EndPointType の WRITER は、引き続き管理、ETL、その他の書き込みオペレーションに使用することにします。元の READER エンドポイントは、「小さい」DB インスタンスまたは「大きい」DB インスタンスに振り向けられる可能性があるため、使用しないことにします。カスタムエンドポイントでは、この動作が予測可能になり、指定したエンドポイントに基づいて「小さい」DB インスタンスまたは「大きい」DB インスタンスのいずれかが接続で確実に使用されます。

```
$ aws rds describe-db-cluster-endpoints --region $REGION
{
    "DBClusterEndpoints": [
        {
            "EndpointType": "WRITER",
            "Endpoint": "custom-endpoint-demo.cluster-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "Status": "available",
            "DBClusterIdentifier": "custom-endpoint-demo"
```

```
        },
        {
            "EndpointType": "READER",
            "Endpoint": "custom-endpoint-demo.cluster-ro-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "Status": "available",
            "DBClusterIdentifier": "custom-endpoint-demo"
        },
        {
            "Endpoint": "small-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "CustomEndpointType": "ANY",
            "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:small-instances",
            "ExcludedMembers": [
                "custom-endpoint-demo-09",
                "custom-endpoint-demo-10"
            ],
            "DBClusterEndpointResourceIdentifier": "cluster-
endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY",
            "DBClusterIdentifier": "custom-endpoint-demo",
            "StaticMembers": [],
            "EndpointType": "CUSTOM",
            "DBClusterEndpointIdentifier": "small-instances",
            "Status": "modifying"
        },
        {
            "Endpoint": "big-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "CustomEndpointType": "READER",
            "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:big-instances",
            "ExcludedMembers": [],
            "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
            "DBClusterIdentifier": "custom-endpoint-demo",
            "StaticMembers": [
                "custom-endpoint-demo-10",
                "custom-endpoint-demo-09"
            ],
            "EndpointType": "CUSTOM",
            "DBClusterEndpointIdentifier": "big-instances",
            "Status": "available"
        }
```

```
    ]
}
```

最後の例では、カスタムエンドポイントへの以降のデータベース接続が Aurora クラスターのさまざまな DB インスタンスに接続されることを示します。small-instances エンドポイントは、db.r4.4xlarge DB インスタンス (このクラスターの下位数字のホスト) に常に接続します。

```
$ mysql -h small-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -
u $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id       |
+--------------------------+
| custom-endpoint-demo-02 |
+--------------------------+

$ mysql -h small-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -
u $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id       |
+--------------------------+
| custom-endpoint-demo-07 |
+--------------------------+

$ mysql -h small-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -
u $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id       |
+--------------------------+
| custom-endpoint-demo-01 |
+--------------------------+
```

big-instances エンドポイントは、db.r4.16xlarge DB インスタンス (このクラスターの最上位数字のホスト 2 つ) に常に接続します。

```
$ mysql -h big-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -u
 $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id       |
```

```
+--------------------------+
| custom-endpoint-demo-10 |
+--------------------------+

$ mysql -h big-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -u
 $MYUSER -p
mysql> select @@aurora_server_id;
+--------------------------+
| @@aurora_server_id      |
+--------------------------+
| custom-endpoint-demo-09 |
+--------------------------+
```

インスタンスエンドポイントの使用

Aurora クラスターの DB インスタンスごとに個別に組み込まれているインスタンスエンドポイントの名前や他の属性は Aurora で管理されます。ユーザーが、この種のエンドポイントを作成、削除、または変更することはできません。Amazon RDS を使用している場合は、インスタンスエンドポイントをよく使用している可能性があります。ただし、Aurora では通常、インスタンスエンドポイントよりもライターエンドポイントとリーダーエンドポイントをよく使用します。

日常的な Aurora オペレーションでインスタンスエンドポイントを主に使用するのは、Aurora クラスター内の特定のインスタンスに影響している容量やパフォーマンスの問題を診断する場合です。特定のインスタンスに接続しているときに、そのステータス可変、メトリクスなどを調査できます。これを行うことで、クラスターの他のインスタンスとは異なり、この特定のインスタンスだけに起きている現象を判断できます。

高度なユースケースでは、一部の DB インスタンスを他のインスタンスとは異なるように設定することもできます。この場合は、インスタンスエンドポイントを使用して他のインスタンスと比べて、より小さいインスタンス、より大きいインスタンス、または特性が異なるインスタンスに直接接続します。また、この特定の DB インスタンスがプライマリインスタンスに昇格される順位が最後になるように、フェイルオーバーの優先度を設定します。このような場合は、インスタンスエンドポイントではなく、カスタムエンドポイントを使用することをお勧めします。カスタムエンドポイントを使用すると、クラスターに追加する DB インスタンスが増えたときの接続の管理と高可用性の確保が容易になります。

Aurora エンドポイントでの高可用性の使用

高可用性が重要であるクラスターでは、ライターエンドポイントを読み取り/書き込み接続や汎用接続に使用し、リーダーエンドポイントを読み取り専用接続に使用します。ライターエンドポイント

とリーダーエンドポイントは、インスタンスエンドポイントよりも DB インスタンスのフェイルオーバーを適切に管理します。インスタンスエンドポイントとは異なり、ライターエンドポイントとリーダーエンドポイントは、クラスター内の DB インスタンスが利用できなくなった場合に、接続先の DB インスタンスを自動的に変更します。

DB クラスターのプライマリ DB インスタンスが失敗した場合、Aurora は新しいプライマリ DB インスタンスに自動的にフェイルオーバーします。そのために、既存の Aurora レプリカが新しいプライマリ DB インスタンスに昇格されるか、新しいプライマリ DB インスタンスが作成されます。フェイルオーバーが発生した場合、ライターエンドポイントを使用して新しく昇格させたプライマリインスタンスや新規作成したプライマリ DB インスタンスに接続するか、リーダーエンドポイントを使用して DB クラスター内のいずれかの Aurora レプリカに接続できます。フェイルオーバーの際に、Aurora レプリカが新しいプライマリ DB インスタンスに昇格されると、リーダーエンドポイントは接続を DB クラスターの新しいプライマリ DB インスタンスに短時間だけ振り向ける場合があります。

インスタンスエンドポイントへの接続を管理するように独自のアプリケーションロジックを設計する場合は、DB クラスター内の使用可能な DB インスタンスの結果セットを手動またはプログラムで検出できます。describe-db-clustersAWS CLI コマンドまたは DescribeDBClusters RDS API オペレーションを使用して、DB クラスターとリーダーエンドポイント、DB インスタンス、DB インスタンスがリーダーか、DB インスタンスの昇格階層を検索します。次に、フェイルオーバー後のインスタンスクラスを確認し、適切なインスタンスエンドポイントに接続できます。

フェイルオーバーについての詳細は、「Aurora DB クラスターの耐障害性」を参照してください。

Aurora DB インスタンスクラス

DB インスタンスクラスによって、 Amazon Aurora DB インスタンスの計算とメモリの容量を決定します。必要な DB インスタンスクラスは、処理能力とメモリの要件によって異なります。

DB インスタンスクラスは、DB インスタンスクラスタイプとサイズの両方で構成されます。例えば、db.r6g は AWS Graviton2 プロセッサを搭載したメモリ最適化DB インスタンスクラスタイプです。db.r6g インスタンスクラスタイプ内の db.r6g.2xlarge は DB インスタンスクラスです。このクラスのサイズは 2xlarge です。

インスタンスクラスの料金の詳細については、「Amazon RDS の料金」を参照してください。

トピック

DB インスタンスクラスタイプ

Amazon Aurora は、以下のタイプの DB インスタンスクラスをサポートしています。

- Aurora Serverless v2
- メモリ最適化
- バースト可能パフォーマンス
- Optimized Reads

Amazon EC2 インスタンスタイプの詳細については、Amazon EC2 ドキュメントの「インスタンスタイプ」を参照してください。

Aurora Serverless v2 インスタンスクラスのタイプ

次の Aurora Serverless v2 タイプが使用可能です。

- db.serverless — Aurora Serverless v2 によって使用される特別な DB インスタンスクラスタイプ。Aurora は、ワークロードの変化に応じて、コンピューティング、メモリ、およびネットワー

クリソースを動的に調整します。使用方法の詳細については、「Aurora Serverless v2 を使用する」を参照してください。

メモリ最適化インスタンスクラスタイプ

メモリ最適化 X ファミリーは、以下のインスタンスクラスをサポートします。

- db.x2g - メモリを大量に消費するアプリケーション用に最適化され、AWS Graviton2 プロセッサを搭載したインスタンスクラス。これらのインスタンスクラスは、メモリの GiB あたりのコストを削減します。

  AWS Graviton2 プロセッサを搭載した DB インスタンスクラスの 1 つを使用するように DB インスタンスを変更できます。これを行うには、他の DB インスタンスを変更する場合と同じ手順を実行します。

メモリ最適化 R ファミリーは、次のインスタンスクラスタイプをサポートします。

- db.r7g - AWS Graviton3 プロセッサを搭載したインスタンスクラス。これらのインスタンスクラスは、メモリ消費の高いワークロードを実行するのに最適です。

  AWS Graviton3 プロセッサを搭載した DB インスタンスクラスの 1 つを使用するように DB インスタンスを変更できます。これを行うには、他の DB インスタンスを変更する場合と同じ手順を実行します。
- db.r6g - AWSGraviton2 プロセッサを搭載したインスタンスクラス。これらのインスタンスクラスは、

  AWS Graviton2 プロセッサを搭載した DB インスタンスクラスの 1 つを使用するように DB インスタンスを変更できます。これを行うには、他の DB インスタンスを変更する場合と同じ手順を実行します。
- db.r6i — 第 3 世代インテル Xeon スケーラブルプロセッサを搭載したインスタンスクラス。これらのインスタンスクラスは SAP 認定であり、MySQL や PostgreSQL などのオープンソースデータベースでメモリ消費の高いワークロードを実行するのに最適です。
- db.r4 — これらのインスタンスクラスは Aurora PostgreSQL 11 および 12 バージョンでのみサポートされます。db.r4 DB インスタンスクラスを使用しているすべての Aurora PostgreSQL DB クラスターは、できるだけ早期により高い世代のインスタンスクラスにアップグレードすることをお勧めします。

  db.r4 インスタンスクラスは、Aurora I/O-Optimizedクラスターストレージ構成に使用できません。

- db.r3 - メモリの最適化を提供するインスタンスクラス。

  Amazon Aurora は、アップグレードの推奨事項を含め、以下のスケジュールで db.r3 DB インスタンスクラスを終了するプロセスを開始しています。db.r3 DB インスタンスクラスを使用しているすべての Aurora MySQL DB クラスターは、できるだけ早く db.r5 以降の DB インスタンスクラスにアップグレードすることをお勧めします。

アクションまたは推奨事項	日付
db.r3 DB インスタンスクラスを使用する Aurora MySQL DB クラスターを作成できなくなりました。	現在
Amazon Aurora は、db.r3 DB インスタンスクラスを使用する Aurora MySQL DB クラスターの同等の db.r5 以降の DB インスタンスクラスへの自動アップグレードを開始しました。	2023 年 1 月 31 日

バースト可能パフォーマンスインスタンスクラスタイプ

以下のバースト可能パフォーマンス DB インスタンスクラスタイプが使用可能です。

- db.t4g – ARM ベースの AWS Graviton2 プロセッサを搭載した汎用インスタンスクラス。これらのインスタンスクラスでは、幅広いバースト汎用ワークロードに対して、以前のバーストパフォーマンス DB インスタンスクラスよりも優れた料金パフォーマンスを提供します。Amazon RDS db.t4g インスタンスは、Unlimited モードに設定されています。したがって、追加料金を支払えば、24 時間にわたり、ベースラインを超えてバーストできることになります。

  AWS Graviton2 プロセッサを搭載した DB インスタンスクラスの 1 つを使用するように DB インスタンスを変更できます。これを行うには、他の DB インスタンスを変更する場合と同じ手順を実行します。
- db.t3 – ベースラインのパフォーマンスレベルを提供しながら、CPU の最大使用率までバーストすることも可能なインスタンスクラス。db.t3 インスタンスは、Unlimited モードに設定されています。これらのインスタンスクラスには、前世代の db.t2 インスタンスクラスよりも多くのコンピューティング容量を備えています。専用ハードウェアと軽量ハイパーバイザーが組み合わされた AWS Nitro System を使用します。これらのインスタンスクラスは、開発/テストサーバーなどの本稼働用以外のサーバーでのみ使用することをお勧めします。

- db.t2 – ベースラインのパフォーマンスレベルを提供しながら、CPU の最大使用率までバーストすることも可能なインスタンスクラス。db.t2 インスタンスは、Unlimited モードに設定されています。これらのインスタンスクラスは、開発/テストサーバーなどの本稼働用以外のサーバーでのみ使用することをお勧めします。

  db.12 インスタンスクラスは、Aurora I/O-Optimizedクラスターストレージ構成に使用できません。

> **Note**
>
> T DB インスタンスクラスは、開発、テスト、またはその他の本稼働以外のサーバーにのみ使用することをお勧めします。T インスタンスクラスの詳細な推奨事項については、「開発やテストのための T インスタンスクラスの使用」を参照してください。

DB インスタンスクラスのハードウェア仕様については、「Aurora 用の DB インスタンスクラスのハードウェア仕様」を参照してください。

Optimized Reads インスタンスクラスタイプ

次の Optimized Reads インスタンスクラスタイプが利用可能です。

- db.r6gd – AWS Graviton2 プロセッサを搭載したインスタンスクラス。これらのインスタンスクラスは、メモリ負荷の高いワークロードの実行に最適で、高速で低レイテンシーのローカルストレージを必要とするアプリケーションに、ローカル NVMe ベースの SSD ブロックレベルストレージを提供します。
- db.r6id – 第 3 世代 Intel Xeon スケーラブルプロセッサを搭載したインスタンスクラス。これらのインスタンスクラスは SAP 認定であり、メモリ負荷の高いワークロードに最適です。最大 1 TiB のメモリと、最大 7.6 TB の直接アタッチされた NVMe ベースの SSD ストレージを備えています。

DB インスタンスクラスでサポートされている DB エンジン

Aurora DB エンジンでサポートされている Amazon Aurora DB インスタンスに関する詳細を以下の表に示します。

インスタンスクラス	Aurora MySQL	Aurora PostgreSQL
db.serverless — 自動容量スケーリング機能を備えた Aurora Serverless v2 インスタンスクラス		
db.serverless	「Aurora Serverless v2 でサポートされているリージョンと Aurora DB エンジン」を参照してください。	「Aurora Serverless v2 でサポートされているリージョンと Aurora DB エンジン」を参照してください。
db.x2g – AWS Graviton2 プロセッサを搭載したメモリ最適化インスタンスクラス		
db.x2g.16xlarge	2.09.2 以降、2.10.0 以降、3.01.0 以降	15.2 以降、14.3 以降、13.3 以降、12.8 以降、11.9、11.12 以降
db.x2g.12xlarge	2.09.2 以降、2.10.0 以降、3.01.0 以降	15.2 以降、14.3 以降、13.3 以降、12.8 以降、11.9、11.12 以降
db.x2g.8xlarge	2.09.2 以降、2.10.0 以降、3.01.0 以降	15.2 以降、14.3 以降、13.3 以降、12.8 以降、11.9、11.12 以降
db.x2g.4xlarge	2.09.2 以降、2.10.0 以降、3.01.0 以降	15.2 以降、14.3 以降、13.3 以降、12.8 以降、11.9、11.12 以降
db.x2g.2xlarge	2.09.2 以降、2.10.0 以降、3.01.0 以降	15.2 以降、14.3 以降、13.3 以降、12.8 以降、11.9、11.12 以降
db.x2g.xlarge	2.09.2 以降、2.10.0 以降、3.01.0 以降	15.2 以降、14.3 以降、13.3 以降、12.8 以降、11.9、11.12 以降
db.x2g.large	2.09.2 以降、2.10.0 以降、3.01.0 以降	15.2 以降、14.3 以降、13.3 以降、12.8 以降、11.9、11.12 以降
db.r6gd – AWS Graviton2 プロセッサを搭載した Optimized Reads インスタンスクラス		
db.r6gd.16xlarge	いいえ	15.4 以降、14.9 以降
db.r6gd.12xlarge	いいえ	15.4 以降、14.9 以降
db.r6gd.8xlarge	いいえ	15.4 以降、14.9 以降

インスタンスクラス	Aurora MySQL	Aurora PostgreSQL
db.r6gd.4xlarge	いいえ	15.4 以降、14.9 以降
db.r6gd.2xlarge	いいえ	15.4 以降、14.9 以降
db.r6gd.xlarge	いいえ	15.4 以降、14.9 以降
db.r6i – Optimized Reads インスタンスクラス		
db.r6id.32xlarge	いいえ	15.4 以降、14.9 以降
db.r6id.24xlarge	いいえ	15.4 以降、14.9 以降
db.r7g – AWS Graviton3 プロセッサを搭載したメモリ最適化インスタンスクラス		
db.r7g.16xlarge	2.12.0 以降、3.03.1 以降	15.2 以降、14.7 以降、13.10 以降
db.r7g.12xlarge	2.12.0 以降、3.03.1 以降	15.2 以降、14.7 以降、13.10 以降
db.r7g.8xlarge	2.12.0 以降、3.03.1 以降	15.2 以降、14.7 以降、13.10 以降
db.r7g.4xlarge	2.12.0 以降、3.03.1 以降	15.2 以降、14.7 以降、13.10 以降
db.r7g.2xlarge	2.12.0 以降、3.03.1 以降	15.2 以降、14.7 以降、13.10 以降
db.r7g.xlarge	2.12.0 以降、3.03.1 以降	15.2 以降、14.7 以降、13.10 以降
db.r7g.large	2.12.0 以降、3.03.1 以降	15.2 以降、14.7 以降、13.10 以降
db.r6g – AWS Graviton2 プロセッサを搭載したメモリ最適化インスタンスクラス		
db.r6g.16xlarge	2.09.2 以降、2.10.0 以降、3.01.0 以降	15.2 以降、14.3 以降、13.3 以降、12.8 以降、11.9、11.12 以降
db.r6g.12xlarge	2.09.2 以降、2.10.0 以降、3.01.0 以降	15.2 以降、14.3 以降、13.3 以降、12.8 以降、11.9、11.12 以降
db.r6g.8xlarge	2.09.2 以降、2.10.0 以降、3.01.0 以降	15.2 以降、14.3 以降、13.3 以降、12.8 以降、11.9、11.12 以降

インスタンスクラス	Aurora MySQL	Aurora PostgreSQL
db.r6g.4xlarge	2.09.2 以降、2.10.0 以降、3.01.0 以降	15.2 以降、14.3 以降、13.3 以降、12.8 以降、11.9、11.12 以降
db.r6g.2xlarge	2.09.2 以降、2.10.0 以降、3.01.0 以降	15.2 以降、14.3 以降、13.3 以降、12.8 以降、11.9、11.12 以降
db.r6g.xlarge	2.09.2 以降、2.10.0 以降、3.01.0 以降	15.2 以降、14.3 以降、13.3 以降、12.8 以降、11.9、11.12 以降
db.r6g.large	2.09.2 以降、2.10.0 以降、3.01.0 以降	15.2 以降、14.3 以降、13.3 以降、12.8 以降、11.9、11.12 以降
db.r6i – メモリ最適化インスタンスクラス		
db.r6i.32xlarge	2.11.0 以降、3.02.1 以降	15.2 以降、14.3 以降、13.5以降、12.9 以降、
db.r6i.24xlarge	2.11.0 以降、3.02.1 以降	15.2 以降、14.3 以降、13.5以降、12.9 以降、
db.r6i.16xlarge	2.11.0 以降、3.02.1 以降	15.2 以降、14.3 以降、13.5以降、12.9 以降、
db.r6i.12xlarge	2.11.0 以降、3.02.1 以降	15.2 以降、14.3 以降、13.5以降、12.9 以降、
db.r6i.8xlarge	2.11.0 以降、3.02.1 以降	15.2 以降、14.3 以降、13.5以降、12.9 以降、
db.r6i.4xlarge	2.11.0 以降、3.02.1 以降	15.2 以降、14.3 以降、13.5以降、12.9 以降、
db.r6i.2xlarge	2.11.0 以降、3.02.1 以降	15.2 以降、14.3 以降、13.5以降、12.9 以降、
db.r6i.xlarge	2.11.0 以降、3.02.1 以降	15.2 以降、14.3 以降、13.5以降、12.9 以降、

インスタンスクラス	Aurora MySQL	Aurora PostgreSQL
db.r6i.large	2.11.0 以降、3.02.1 以降	15.2 以降、14.3 以降、13.5以降、12.9 以降、
db.r5 – メモリ最適化インスタンスクラス		
db.r5.24xlarge	すべての 2.x バージョン、3.01.0 以降	現在利用可能なすべてのバージョン
db.r5.16xlarge	すべての 2.x バージョン、3.01.0 以降	現在利用可能なすべてのバージョン
db.r5.12xlarge	すべての 2.x バージョン、3.01.0 以降	現在利用可能なすべてのバージョン
db.r5.8xlarge	すべての 2.x バージョン、3.01.0 以降	現在利用可能なすべてのバージョン
db.r5.4xlarge	すべての 2.x バージョン、3.01.0 以降	現在利用可能なすべてのバージョン
db.r5.2xlarge	すべての 2.x バージョン、3.01.0 以降	現在利用可能なすべてのバージョン
db.r5.xlarge	すべての 2.x バージョン、3.01.0 以降	現在利用可能なすべてのバージョン
db.r5.large	すべての 2.x バージョン、3.01.0 以降	現在利用可能なすべてのバージョン
db.r4 – メモリ最適化インスタンスクラス		
db.r4.16xlarge	すべての 2.x バージョン、3.01.0 以降ではサポートされていません	いいえ
db.r4.8xlarge	すべての 2.x バージョン、3.01.0 以降ではサポートされていません	いいえ

インスタンスクラス	Aurora MySQL	Aurora PostgreSQL
db.r4.4xlarge	すべての 2.x バージョン、3.01.0 以降ではサポートされていません	いいえ
db.r4.2xlarge	すべての 2.x バージョン、3.01.0 以降ではサポートされていません	いいえ
db.r4.xlarge	すべての 2.x バージョン、3.01.0 以降ではサポートされていません	いいえ
db.r4.large	すべての 2.x バージョン、3.01.0 以降ではサポートされていません	いいえ
db.t4g – AWS Graviton2 プロセッサを搭載したバーストパフォーマンスインスタンスクラス		
db.t4g.2xlarge	いいえ	いいえ
db.t4g.xlarge	いいえ	いいえ
db.t4g.large	2.11.1 以降、3.01.0 以降	15.2 以降、14.3 以降、13.3 以降、12.7 以降、11.12 以降、
db.t4g.medium	2.11.1 以降、3.01.0 以降	15.2 以降、14.3 以降、13.3 以降、12.7 以降、11.12 以降、
db.t4g.small	いいえ	いいえ
db.t3 – バーストパフォーマンスインスタンスクラス		
db.t3.2xlarge	いいえ	いいえ
db.t3.xlarge	いいえ	いいえ
db.t3.large	2.11.1 以降、3.01.0 以降	15.2 以降、14.3 以降、13.3 以降、12.7 以降、11.12 以降、
db.t3.medium	すべての 2.x バージョン、3.01.0 以降	15.2 以降、14.3 以降、13.3 以降、12.7 以降、11.12 以降、

インスタンスクラス	Aurora MySQL	Aurora PostgreSQL
db.t3.small	すべての 2.x バージョン、3.01.0 以降ではサポートされていません	いいえ
db.t3.micro	いいえ	いいえ
db.t2 – バーストパフォーマンスインスタンスクラス		
db.t2.medium	すべての 2.x バージョン、3.01.0 以降ではサポートされていません	いいえ
db.t2.small	すべての 2.x バージョン、3.01.0 以降ではサポートされていません	いいえ

AWS リージョン での DB インスタンスクラスのサポートを決定する

特定の AWS リージョン で各 DB エンジンがサポートしている DB インスタンスクラスを決定するため、複数のアプローチの中から 1 つを選択できます。AWS Management Console で、「Amazon RDS 料金表」ページ、または describe-orderable-db-instance-options AWS CLI コマンドを使用できます。

> **Note**
>
> AWS Management Console でオペレーションを実行すると、特定の DB エンジン、DB エンジンバージョン、および AWS リージョン でサポートされる DB インスタンスクラスが自動的に表示されます。実行できるオペレーションの例には、DB クラスターの作成や変更などが含まれます。

目次

- AWS リージョン 内で特定の DB インスタンスクラスをサポートする DB エンジンのバージョンの一覧表示

Amazon RDS 料金ページを使用して、AWS リージョン での DB インスタンスクラスのサポートを決定する

「Amazon Aurora の料金」ページを使用して、特定の AWS リージョン内で各 DB エンジンがサポートしている DB インスタンスクラスを確認できます。

料金ページを使用して、リージョンの各エンジンでサポートされる DB インスタンスクラスを決定するには

1. [Amazon Aurora の料金] に移動します。
2. [AWS 料金見積りツール] セクションで Amazon Aurora エンジンを選択します。
3. [リージョンを選択] で、[AWS リージョン] を選択します。
4. [クラスター設定オプション] で、設定オプションを選択します。
5. 互換性のあるインスタンスのセクションを使用して、サポートされている DB インスタンスクラスを確認します。
6. (オプション) 見積りツールで他のオプションを選択し、[概要を保存して表示] または [サービスを保存して追加] を選択します。

AWS CLI を使用して、AWS リージョン 内での DB インスタンスクラスのサポートを決定する

AWS CLI を使用して、AWS リージョン 内の特定の DB エンジンおよび DB エンジンバージョンでサポートされる DB インスタンスクラスを決定できます。

次の AWS CLI 例を使用する場合は、DB エンジン、DB エンジンバージョン、DB インスタンスクラス、AWS リージョン の有効な値を入力してください。次の表は、有効な DB エンジンの値を示しています。

エンジン名	CLI コマンドのエンジン値	バージョンの詳細
MySQL 5.7 互換、8.0 互換 Aurora	`aurora-mysql`	Aurora MySQL のリリースノートの Amazon Aurora MySQL バージョン 2 のデータベースエ

エンジン名	CLI コマンドのエンジン値	バージョンの詳細
		ンジンの更新と Amazon Aurora MySQL バージョン 3 のデータベースエンジンの更新
Aurora PostgreSQL	`aurora-postgresql`	Aurora PostgreSQL リリースノート

AWS リージョン 名については、「AWS リージョン」を参照してください。

次の例は、describe-orderable-db-instance-options AWS CLI コマンドを使用して、AWS リージョンでの DB インスタンスクラスのサポートを決定する方法を示しています。

トピック

- AWS リージョン 内の特定の DB エンジンバージョンでサポートされている DB インスタンスクラスの一覧表示
- AWS リージョン 内で特定の DB インスタンスクラスをサポートする DB エンジンのバージョンの一覧表示

AWS リージョン 内の特定の DB エンジンバージョンでサポートされている DB インスタンスクラスの一覧表示

AWS リージョン 内の特定の DB エンジンバージョンでサポートされている DB インスタンスクラスを一覧表示するには、次のコマンドを実行します。

Linux、macOS、Unix の場合:

```
aws rds describe-orderable-db-instance-options --engine engine --engine-version version
 \
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region region
```

Windows の場合:

```
aws rds describe-orderable-db-instance-options --engine engine --engine-version version
 ^
```

```
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" ^
    --output table ^
    --region region
```

出力には、各 DB インスタンスクラスでサポートされているエンジンモードも表示されます。

例えば、次のコマンドは、米国東部 (バージニア北部) の Aurora PostgreSQL DB エンジンのバージョン 13.6 でサポートされている DB インスタンスクラスを一覧表示します。

Linux、macOS、Unix の場合:

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --engine-
version 15.3 \
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region us-east-1
```

Windows の場合:

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --engine-
version 15.3 ^
    --query "OrderableDBInstanceOptions[].
{DBInstanceClass:DBInstanceClass,SupportedEngineModes:SupportedEngineModes[0]}"  ^
    --output table ^
    --region us-east-1
```

AWS リージョン 内で特定の DB インスタンスクラスをサポートする DB エンジンのバージョンの一覧表示

AWS リージョン 内で特定の DB インスタンスクラスをサポートしている DB エンジンのバージョンを一覧表示するには、次のコマンドを実行します。

Linux、macOS、Unix の場合:

```
aws rds describe-orderable-db-instance-options --engine engine --db-instance-
class DB_instance_class \
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
```

```
    --region region
```

Windows の場合:

```
aws rds describe-orderable-db-instance-options --engine engine --db-instance-
class DB_instance_class ^
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" ^
    --output table ^
    --region region
```

出力には、各 DB エンジンバージョンでサポートされているエンジンモードも表示されます。

例えば、次のコマンドは、US East (N. Virginia) で db.r5.large DB インスタンスクラスをサポートする Aurora PostgreSQL DB エンジンの DB エンジンのバージョンを一覧表示します。

Linux、macOS、Unix の場合:

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-
instance-class db.r7g.large \
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" \
    --output table \
    --region us-east-1
```

Windows の場合:

```
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-
instance-class db.r7g.large ^
    --query "OrderableDBInstanceOptions[].
{EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes[0]}" ^
    --output table ^
    --region us-east-1
```

Aurora 用の DB インスタンスクラスのハードウェア仕様

以下の用語を使用して、DB インスタンスクラスのハードウェア仕様について説明します。

vCPU

仮想中央演算装置 (CPU) の数。仮想 CPU は、DB インスタンスクラスの比較に使用できる容量の単位です。特定のプロセッサを購入またはリースして数か月から数年間使用する代わりに、

時間単位で処理能力をレンタルすることができます。私たちの目標は、実際の基盤となるハードウェアの範囲内で、一貫して特定の容量の CPU 能力を使用できるようにすることです。

ECU

Amazon EC2 インスタンスの整数処理能力の相対的測定基準。異なるインスタンスクラス間で開発者が簡単に CPU 能力値を比較できるように、Amazon EC2 コンピュート単位が定義されています。特定のインスタンスに配分されている CPU 量は、これらの EC2 コンピュート単位で明示されます。現在のところ、1 つの ECU で、1.0 - 1.2 GHz 2007 Opteron または 2007 Xeon プロセッサと同等の CPU 能力が提供されます。

メモリ (GiB)

DB インスタンスに割り当てられる RAM (ギガバイナリバイト単位)。通常、メモリと vCPU の比率は一定です。例として、db.r4 インスタンスクラスを使用します。このインスタンスクラスのメモリと vCPU の比率は db.r5 インスタンスクラスと同じですが、db.r5 インスタンスクラスのパフォーマンスは、ほとんどのユースケースで db.r4 インスタンスクラスより安定して優れています。

最大 EBS 帯域幅 (Mbps)

最大 EBS 帯域幅 (メガビット/秒)。8 で割ると、メガバイト/秒でのスループットが得られます。

> **Note**
>
> この図は、DB インスタンス内のローカルストレージの I/O 帯域幅を示しています。Aurora クラスターボリュームとの通信には適用されません。

ネットワーク帯域幅

他の DB インスタンスクラスとの相対的なネットワーク速度。

Aurora 対応の Amazon RDS DB インスタンスクラスに関するハードウェアの詳細を以下の表に示します。

DB インスタンスクラスごとの Aurora DB エンジンサポートについては、「DB インスタンスクラスでサポートされている DB エンジン」を参照してください。

インスタンスクラス	vCPU	ECU	メモリ (GiB)	ローカルストレージの最大帯域幅 (Mbps)	ネットワークパフォーマンス (Gbps)
db.x2g – メモリ最適化インスタンスクラス					
db.x2g.16xlarge	64	—	1024	19,000	25
db.x2g.12xlarge	48	—	768	14,250	20
db.x2g.8xlarge	32	—	512	9,500	12
db.x2g.4xlarge	16	—	256	4,750	最大 10
db.x2g.2xlarge	8	—	128	最大 4,750	最大 10
db.x2g.xlarge	4	—	64	最大 4,750	最大 10
db.x2g.large	2	—	32	最大 4,750	最大 10
db.r7g – AWS Graviton3 プロセッサを搭載したメモリ最適化インスタンスクラス					
db.r7g.16xlarge	64	—	512	20,000	30
db.r7g.12xlarge	48	—	384	15,000	22.5
db.r7g.8xlarge	32	—	256	10,000	15
db.r7g.4xlarge	16	—	128	最大 10,000	最大 15
db.r7g.2xlarge	8	—	64	最大 10,000	最大 15
db.r7g.xlarge	4	—	32	最大 10,000	最大 12.5
db.r7g.large	2	—	16	最大 10,000	最大 12.5
db.r6g – AWS Graviton2 プロセッサを搭載したメモリ最適化インスタンスクラス					
db.r6g.16xlarge	64	—	512	19,000	25
db.r6g.12xlarge	48	—	384	13,500	20

インスタンスクラス	vCPU	ECU	メモリ (GiB)	ローカルストレージの最大帯域幅 (Mbps)	ネットワークパフォーマンス (Gbps)
db.r6g.8xlarge	32	—	256	9,000	12
db.r6g.4xlarge	16	—	128	4,750	最大 10
db.r6g.2xlarge	8	—	64	最大 4,750	最大 10
db.r6g.xlarge	4	—	32	最大 4,750	最大 10
db.r6g.large	2	—	16	最大 4,750	最大 10
db.r6i – メモリ最適化インスタンスクラス					
db.r6i.32xlarge	128	—	1,024	40,000	50
db.r6i.24xlarge	96	—	768	30,000	37.5
db.r6i.16xlarge	64	—	512	20,000	25
db.r6i.12xlarge	48	—	384	15,000	18.75
db.r6i.8xlarge	32	—	256	10,000	12.5
db.r6i.4xlarge	16	—	128	最大 10,000	最大 12.5
db.r6i.2xlarge	8	—	64	最大 10,000	最大 12.5
db.r6i.xlarge	4	—	32	最大 10,000	最大 12.5
db.r6i.large	2	—	16	最大 10,000	最大 12.5
db.r5 – メモリ最適化インスタンスクラス					
db.r5.24xlarge	96	347	768	19,000	25
db.r5.16xlarge	64	264	512	13,600	20
db.r5.12xlarge	48	173	384	9,500	12

インスタンスクラス	vCPU	ECU	メモリ (GiB)	ローカルストレージの最大帯域幅 (Mbps)	ネットワークパフォーマンス (Gbps)
db.r5.8xlarge	32	132	256	6,800	10
db.r5.4xlarge	16	71	128	4,750	最大 10
db.r5.2xlarge	8	38	64	最大 4,750	最大 10
db.r5.xlarge	4	19	32	最大 4,750	最大 10
db.r5.large	2	10	16	最大 4,750	最大 10
db.r4 – メモリ最適化インスタンスクラス					
db.r4.16xlarge	64	195	488	14,000	25
db.r4.8xlarge	32	99	244	7,000	10
db.r4.4xlarge	16	53	122	3,500	最大 10
db.r4.2xlarge	8	27	61	1,700	最大 10
db.r4.xlarge	4	13.5	30.5	850	最大 10
db.r4.large	2	7	15.25	425	最大 10
db.t4g – バーストパフォーマンスインスタンスクラス					
db.t4g.large	2	—	8	最大 2,780	最大 5
db.t4g.medium	2	—	4	最大 2,085	最大 5
db.t3 – バーストパフォーマンスインスタンスクラス					
db.t3.large	2	可変	8	最大 2,048	最大 5
db.t3.medium	2	可変	4	最大 1,536	最大 5
db.t3.small	2	可変	2	最大 1,536	最大 5

インスタンスクラス	vCPU	ECU	メモリ (GiB)	ローカルストレージの最大帯域幅 (Mbps)	ネットワークパフォーマンス (Gbps)
db.t2 – バーストパフォーマンスインスタンスクラス					
db.t2.medium	2	可変	4	—	中
db.t2.small	1	可変	2	—	低

Amazon Aurora ストレージと信頼性

次に、Aurora ストレージサブシステムについて説明します。Aurora の分散型の共有ストレージアーキテクチャは、Aurora クラスターのパフォーマンス、スケーラビリティ、および信頼性に重要な影響を与えます。

トピック

- Amazon Aurora ストレージの概要
- クラスターボリュームの内容
- Amazon Aurora DB クラスターのストレージ設定
- Aurora ストレージのサイズを自動的に変更する方法
- Aurora データストレージに対する請求方法
- Amazon Aurora の信頼性

Amazon Aurora ストレージの概要

Aurora のデータはクラスターボリュームに保存されます。これは、SSD (Solid State Drive) を使用する単一の仮想ボリュームです。クラスターボリュームは、単一の AWS リージョンの 3 つのアベイラビリティーゾーン間のデータのコピーで構成されます。データはアベイラビリティーゾーン間で自動的にレプリケートされるため、データ損失の可能性は低く、耐久性の高いは非常に高くなります。また、このレプリケーションにより、フェイルオーバー時のデータベースの可用性が高くなります。データのコピーは他のアベイラビリティーゾーンに既に存在するため、DB クラスター内の DB インスタンスに対するデータ要求は継続して処理されます。レプリケーションの数は、クラスター内の DB インスタンスの数とは関係ありません。

Aurora は、非永続的な一時ファイル用に、分離したローカルストレージを使用します。これには、クエリ処理中の大きなデータセットのソートや、インデックスの作成などの目的に使用するファイルが含まれます。詳細については、「Aurora MySQL 用の一時ストレージの制限」および「Aurora PostgreSQL 用の一時ストレージの制限」を参照してください。

クラスターボリュームの内容

Aurora クラスターボリュームには、すべてのユーザーデータ、スキーマオブジェクト、および内部メタデータ (システムテーブルやバイナリログなど) が入ります。例えば、Aurora は Aurora クラスター内のすべてのテーブル、インデックス、バイナリラージオブジェクト (BLOB)、ストアドプロシージャなどをクラスターボリュームに保存します。

Aurora の共有ストレージアーキテクチャでは、クラスター内の DB インスタンスとは別個にデータが処理されます。例えば、Aurora はテーブルデータの新しいコピーを作成しないため、DB インスタンスをすばやく追加できます。代わりに、DB インスタンスから、すべてのデータが既に含まれている共有ボリュームに接続します。クラスターから DB インスタンスを削除しても、元になるデータはクラスターから削除されません。クラスター全体を削除した場合にのみ、Aurora でデータが削除されます。

Amazon Aurora DB クラスターのストレージ設定

Amazon Aurora には、2 つの DB クラスターストレージ設定があります:

- Aurora I/O-Optimized— I/O 集約型のアプリケーションの価格性能と予測可能性が向上しました。お支払いいただくのは DB クラスターの使用量とストレージに対してのみで、読み取りと書き込み I/O オペレーションに追加料金はかかりません。

  Aurora I/O-Optimized は、I/O 支出が Aurora データベースの総支出の 25% 以上である場合に最適な選択肢です。

  Aurora I/O-Optimized クラスター設定をサポートする DB エンジンバージョンの DB クラスターを作成または変更する場合は、Aurora I/O-Optimized を選択できます。Aurora I/O-Optimized から Aurora Standard にはいつでも切り替えることができます。
- Aurora Standard — I/ O使用量が中程度の多くのアプリケーション向けの費用対効果の高い価格設定。DB クラスターの使用量とストレージに加えて、I/O オペレーションの 100 万件のリクエストごとに標準料金をお支払いいただきます。

  Aurora Standard は、I/O 支出が Aurora データベースの総支出の 25% 未満である場合に最適な選択肢です。

30 日に 1 回 Aurora Standard から Aurora I/O-Optimized に切り替えることができます。Aurora Standard から Aurora I/O-Optimized、または Aurora I/O-Optimized から Aurora Standard に切り替えてもダウンタイムはありません。

AWS リージョン に関する詳細とバージョンのサポートについては、「クラスターストレージ構成でサポートされているリージョンと Aurora DB エンジン」を参照してください。

Amazon Aurora ストレージ設定の料金情報の詳細については、「Amazon Aurora の料金」を参照してください。

DB クラスターを作成する際のストレージ構成の選択については、「DB クラスターの作成」を参照してください。DB クラスターのストレージ構成の変更については、「Amazon Aurora の設定」を参照してください。

Aurora ストレージのサイズを自動的に変更する方法

Aurora クラスターボリュームは、データベースのデータ量が増えるにつれて自動的に増加します。Aurora クラスターボリュームの最大サイズは、DB エンジンのバージョンに応じて、128 テビバイト (TiB) または 64 TiB のどちらかです。特定のバージョンの最大サイズは、「Amazon Aurora サイズ制限」でご確認ください。この自動ストレージスケーリングは、高性能で高度に分散されたストレージサブシステムと組み合わされています。これにより、信頼性と高可用性を主目的とする重要なエンタープライズデータには Aurora が適切な選択となります。

ボリュームステータスを確認するには、Aurora MySQL DB クラスターのボリュームステータスの表示 または Aurora PostgreSQL DB クラスターのボリュームステータスの表示 をご覧ください。ストレージコストと他の優先順位のバランスをとる方法について、ストレージのスケーリング が Amazon Aurora メトリクス `AuroraVolumeBytesLeftTotal` および `VolumeBytesUsed` を CloudWatch でモニタリングする方法について説明しています。

Aurora のデータを削除すると、そのデータに割り当てられていた領域が解放されます。データの削除の例としては、テーブルの削除や切り捨てなどがあります。このストレージ使用量の自動削減により、ストレージ料金を最小限に抑えることができます。

> **Note**
>
> ここで説明するストレージ制限と動的なサイズ変更の動作は、クラスターボリュームに保存されている永続テーブルやその他のデータに適用されます。

Aurora PostgreSQL の場合、臨時テーブルデータがローカル DB インスタンスに保存されます。
Aurora MySQL バージョン 2 では、一時テーブルデータは、デフォルトで、ライターインスタンスの場合はクラスターボリュームに、リーダーインスタンスの場合はローカルストレージに保存されます。詳細については、「オンディスク一時テーブルのストレージエンジン」を参照してください。
Aurora MySQL バージョン 3 では、一時テーブルデータはローカル DB インスタンスまたはクラスターボリュームに保存されます。詳細については、「Aurora MySQL バージョン 3 での新しい一時テーブルの動作」を参照してください。
ローカルストレージでの一時テーブルの最大サイズは、DB インスタンスの最大ローカルストレージサイズによって制限されます。ローカルストレージのサイズは、使用するインスタンスクラスによって異なります。詳細については、「Aurora MySQL 用の一時ストレージの制限」および「Aurora PostgreSQL 用の一時ストレージの制限」を参照してください。

クラスターボリュームの最大サイズやデータ削除時の自動サイズ変更など、一部のストレージ機能は、クラスターの Aurora バージョンによって異なります。詳細については、「ストレージのスケーリング」を参照してください。また、ストレージの問題を回避する方法と、クラスター内の割り当てられたストレージと空き領域をモニタリングする方法についても確認できます。

Aurora データストレージに対する請求方法

Aurora クラスターボリュームは 128 tebibytes (TiB) まで拡張できますが、請求対象となるのは Aurora クラスターボリュームで使用した分の領域のみです。Aurora の以前のバージョンでは、データを削除したときに解放された領域をクラスターボリュームで再利用できましたが、割り当てられたストレージ領域が縮小されることはありませんでした。現在では、テーブルやデータベースの削除などによって Aurora データが削除されると、割り当てられた領域全体が相応に減少します。したがって、不要になったテーブル、インデックス、データベースなどを削除することで、ストレージ料金を削減できます。

Tip

動的サイズ変更機能のない以前のバージョンの場合、クラスターのストレージ使用量をリセットするには、論理的なダンプを実行して新しいクラスターに復元する必要がありました。このオペレーションは、データが大量にある場合、長い時間がかかることがあります。このような状況が発生した場合は、クラスターを、動的なボリュームのサイズ変更をサポートするバージョンにアップグレードすることを検討してください。

動的なサイズ変更をサポートする Aurora のバージョンと、クラスターのストレージ使用量をモニタリングしてストレージ料金を最小限に抑える方法については、「ストレージのスケーリング」を参照してください。Aurora バックアップストレージの請求については、「Amazon Aurora バックアップストレージの使用状況を確認する」を参照してください。Aurora データストレージの料金情報については、「Amazon RDS for Aurora の料金」を参照してください。

Amazon Aurora の信頼性

Aurora は、信頼性、耐久性の高い、および耐障害性を持つように設計されています。Aurora DB クラスターは、Aurora レプリカの追加や複数のアベイラビリティーゾーンへの配置などを行うことで可用性を向上させるように設計できます。さらに Aurora には、信頼できるデータベースソリューションとして使用できるさまざまな自動機能があります。

トピック

- ストレージの自動修復
- 存続できるページキャッシュ
- 計画外の再起動からの復旧

ストレージの自動修復

Aurora では、データの複数のコピーを 3 つのアベイラビリティーゾーンに保持するため、ディスク障害によってデータが失われる可能性が最小限に抑えられます。Aurora は、クラスターボリュームを構成するディスクボリュームの障害を自動的に検出します。ディスクボリュームのセグメントで障害が発生すると、Aurora はすぐにそのセグメントを修復します。Aurora はディスクセグメントを修復するときに、クラスターボリュームを構成する他のボリューム内のデータを使用して、修復されるセグメントのデータが最新であるようにします。その結果、Aurora ではデータ損失が回避され、ディスク障害から回復するためのポイントインタイム復元を実行する必要性が低減します。

存続できるページキャッシュ

Aurora では、各 DB インスタンスのページキャッシュはデータベースとは別のプロセスで管理されるため、ページキャッシュはデータベースとは無関係に存続できます。(ページキャッシュは Aurora MySQL では InnoDB バッファプール、Aurora PostgreSQL ではバッファキャッシュとも呼ばれます。)

まれにデータベース障害が発生した場合でも、ページキャッシュはメモリに残るため、データベースの起動時に現在のデータページがページキャッシュに「ウォームアップ」されます。これにより、

ページキャッシュを「ウォームアップ」するために読み取り I/O 操作を実行するための初期クエリの必要性がバイパスされ、パフォーマンスが向上します。

Aurora MySQL では、再起動およびフェイルオーバー時のページキャッシュの動作は次のとおりです。

- 2.10 より前のバージョン — ライター DB インスタンスが再起動すると、ライターインスタンスのページキャッシュは存続しますが、リーダー DB インスタンスのページキャッシュは失われます。
- バージョン 2.10 以降 – リーダーインスタンスを再起動せずに、ライターインスタンスを再起動できます。
  - ライターインスタンスの再起動時にリーダーインスタンスが再起動しなくても、ページキャッシュは失われません。
  - ライターインスタンスの再起動時にリーダーインスタンスが再起動した場合、ページキャッシュが失われます。
- リーダーインスタンスが再起動すると、ライターインスタンスとリーダーインスタンスのページキャッシュが両方とも存続します。
- DB クラスターがフェイルオーバーした場合、その効果はライターインスタンスが再起動したときと似ています。新しいライターインスタンス (以前のリーダーインスタンス) ではページキャッシュは存続しますが、リーダーインスタンス (以前のライターインスタンス) ではページキャッシュは存続しません。

Aurora PostgreSQL の場合、クラスターキャッシュ管理を使用して、フェイルオーバー後にライターインスタンスになる指定されたリーダーインスタンスのページキャッシュを保持できます。詳細については、「Aurora PostgreSQL のクラスターキャッシュ管理によるフェイルオーバー後の高速リカバリ」を参照してください。

計画外の再起動からの復旧

Aurora は、計画外の再起動からほぼ瞬時に復旧し、バイナリログなしでアプリケーションデータを提供し続けるように設計されています。Aurora は、パラレルスレッドで非同期に復旧します。これにより、計画外の再起動の直後にデータベースを開き、使用できるようにします。

詳細については、「Aurora DB クラスターの耐障害性」および「データベースの再起動時間を短縮するための最適化」を参照してください。

以下に示しているのは、Aurora MySQL のバイナリログ記録と計画外の再起動からの復旧に関する考慮事項です。

- Aurora でバイナリログ記録を有効にすると、計画外の再起動後の復旧時間に直接影響を与えます。これは、DB インスタンスで強制的にバイナリログ復旧が実行されるためです。
- 使用するバイナリログ記録のタイプは、ログ記録のサイズと効率に影響を与えます。データベースアクティビティの量が同じでも、バイナリログの形式によって記録される情報の量が異なります。`binlog_format` パラメータの以下の設定により、ログデータの量が異なることになります。
  - `ROW` - ログデータの量が最も多い
  - `STATEMENT` - ログデータの量が最も少ない
  - `MIXED` - ログデータの量が中程度。通常、データ整合性とパフォーマンスのバランスが最も良くなります

  バイナリログデータの量は復旧時間に影響を与えます。バイナリログに記録されているデータが多いほど、DB インスタンスが復旧中に処理するデータが多くなり、復旧時間が長くなります。
- バイナリログ記録を使用して計算オーバーヘッドを削減し、復元時間を短縮するために、拡張バイナリログを使用できます。拡張バイナリログにより、データベースの復元時間が最大 99% 向上します。詳細については、「拡張バイナリログ記録の設定」を参照してください。
- Aurora では、DB クラスター内でデータをレプリケートしたり、ポイントインタイムリカバリ (PITR) を実行したりするために、バイナリログは不要です。
- 外部レプリケーション (または外部バイナリログストリーミング) にバイナリログが不要な場合は、`binlog_format` パラメータを `OFF` に設定して、バイナリログ記録を無効にすることをお勧めします。これにより、復旧時間が短くなります。

Aurora バイナリログ記録とレプリケーションの詳細については、「Amazon Aurora でのレプリケーション」を参照してください。さまざまな MySQL レプリケーションタイプの影響の詳細については、MySQL ドキュメントの「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」を参照してください。

Amazon Aurora のセキュリティ

Amazon Aurora のセキュリティは次の 3 つのレベルで管理されます。

- Aurora DB クラスターと DB インスタンスに対する Amazon RDS 管理アクションを実行できるユーザーを制御するには、AWS Identity and Access Management (IAM) を使用します。IAM 認証情報を使用して AWS に接続するとき、AWS アカウントには、Amazon RDS の管理オペレーションを実行するためのアクセス許可を付与する IAM ポリシーが必要です。詳細については、「Amazon Aurora での Identity and Access Management」を参照してください。

IAM を使用して Amazon RDS コンソールにアクセスする場合は、まず、ユーザー認証情報で AWS Management Console にログオンし、次に Amazon RDS コンソール (https://console.aws.amazon.com/rds) に移動する必要があります。

- Aurora DB クラスターは、Amazon VPC サービスに基づいて Virtual Private Cloud (VPC) で作成する必要があります。VPC 内の Aurora DB クラスター用の DB インスタンスのエンドポイントとポートに対して接続を開くことができるデバイスと Amazon EC2 インスタンスを制御するには、VPC セキュリティグループを使用します。これらのエンドポイントおよびポートの接続には TLS/SSL (Transport Layer Security/Secure Sockets Layer) を使用できます。さらに、会社のファイアウォールルールでも、社内のいずれのデバイスが DB インスタンスへの接続を開くことができるかを制御できます。VPC の詳細については、「Amazon VPC VPC とAmazon Aurora」を参照してください。
- Amazon Aurora DB クラスターに対するログインとアクセス権限を認証するには、次の方法のいずれか、または組み合わせて使用します。
  - MySQL または PostgreSQL のスタンドアロン DB インスタンスと同じ方法を使用することもできます。

    SQL コマンドの使用やデータベーススキーマテーブルの変更など、MySQL や PostgreSQL のスタンドアロン DB インスタンスのログインとアクセス許可を認証するための技法も、Aurora で使用できます。詳細については、「Amazon Aurora MySQL でのセキュリティ」または「Amazon Aurora PostgreSQL でのセキュリティ」を参照してください。
  - IAM データベース認証を使用できます。

    IAM データベース認証を使用する場合は、ユーザーまたは IAM ロールと認証トークンを使用して、Aurora DB クラスターに対して認証を行います。認証トークン は、署名バージョン 4 の署名プロセスを使用して生成されている一意の値です。IAM データベース認証では、同一の認証情報を使用して AWS リソースおよびデータベースへのアクセスを制御できます。詳細については、「 の IAM データベース認証」を参照してください。
  - Aurora PostgreSQL および Aurora MySQL に Kerberos 認証を使用できます。

    ユーザーが Aurora PostgreSQL DB および Aurora MySQLDB クラスターに接続する場合、Kerberos を使用してそのユーザーを認証できます。この場合、Kerberos 認証を有効にするために、DB クラスターは AWS Directory Service for Microsoft Active Directory を使用できます。AWS Directory Service for Microsoft Active Directory は AWS Managed Microsoft AD とも呼ばれます。同じディレクトリにすべての認証情報を保持することで時間と労力を節約できます。複数の DB クラスターの認証情報を一元的に保存および管理できます。また、ディレクトリを使用することで、セキュリティプロファイル全体を向上できます。詳細については、Aurora

PostgreSQL で Kerberos 認証を使用する および Aurora MySQL での Kerberos 認証の使用 を参照してください。

セキュリティ設定の詳細については、「Amazon Aurora でのセキュリティ」を参照してください。

Aurora DB クラスターでの SSL の使用

Amazon Aurora DB クラスターは、アプリケーションからの Secure Sockets Layer (SSL) 接続を、Amazon RDS DB インスタンスと同じプロセスと公開鍵を使用してサポートします。詳細については「Amazon Aurora MySQL でのセキュリティ」、「Amazon Aurora PostgreSQL でのセキュリティ」または「Aurora Serverless v1 での TLS/SSL の使用」を参照してください。

Amazon Aurora の高可用性

Amazon Aurora アーキテクチャでは、ストレージとコンピューティングが分離されます。Aurora では、高可用性のためのいくつかの機能が、DB クラスターのデータに対し適用されます。クラスター内の DB インスタンスの一部またはすべてが使用できなくなっても、データは安全に維持されます。その他の高可用性機能は、DB インスタンスに適用されます。これらの機能により、1 つまたはそれ以上の DB インスタンスがアプリケーションからのデータベースリクエストを処理できます。

トピック

- Aurora データの高可用性
- Aurora DB インスタンスの高可用性
- Aurora グローバルデータベースを使用した AWS リージョン間での高可用性
- Aurora DB クラスターの耐障害性
- Amazon RDS Proxy における高可用性

Aurora データの高可用性

Aurora は、単一の AWS リージョン 内の複数のアベイラビリティーゾーンにまたがる DB クラスターにデータのコピーを保存します。Aurora は、DB クラスターのインスタンスが複数のアベイラビリティーゾーンにまたがっているかどうかにかかわらず、これらのコピーを作成します。Aurora の詳細については、「Amazon Aurora DB クラスターの管理」を参照してください。

データがプライマリ DB インスタンスに書き込まれると、Aurora によりアベイラビリティーゾーン全体で、クラスターボリュームに関連付けられた 6 つのストレージノードにデータが同期的に複製

されます。これにより、データの冗長性が確保されて I/O のフリーズが回避され、システムバックアップ時のレイテンシー急上昇が最小限に抑えられます。高可用性を備えた DB インスタンスを実行すると、計画されたシステムメンテナンス中の可用性が向上し、障害とアベイラビリティーゾーンの中断からデータベースを保護できます。アベイラビリティーゾーンの詳細については、「リージョンとアベイラビリティーゾーン」を参照してください。

Aurora DB インスタンスの高可用性

プライマリ (ライター) インスタンスを作成した後、最大 15 の読み取り専用 Aurora レプリカを作成できます。Aurora レプリカはリーダーインスタンスとも呼ばれます。

日常的な操作で、読み取りを多用するアプリケーションの作業の一部をオフロードするには、リーダーインスタンスを使用して SELECT クエリを処理します。問題がプライマリインスタンスに影響する場合、これらのリーダーインスタンスの 1 つがプライマリインスタンスとして引き継ぎます。このメカニズムはフェイルオーバーと呼ばれます。多くの Aurora 機能がフェイルオーバーメカニズムに適用されます。例えば、Aurora はデータベースの問題を検出し、必要に応じてフェイルオーバーのメカニズムを自動的にアクティブにします。Aurora には、フェイルオーバーが完了するまでの時間を短縮する機能もあります。これにより、フェイルオーバー中にデータベースを書き込みに使用できない時間を最小限に抑えることができます。

Aurora は、できるだけ早く復旧するように設計されており、多くの場合、復旧までの最も速い方法は、再起動または同じ DB インスタンスへのフェイルオーバーです。再起動はフェイルオーバーよりも速く、オーバーヘッドも少なくなります。

フェイルオーバーによって新しいプライマリインスタンスが昇格しても同じ接続文字列を使用するには、クラスターエンドポイントに接続します。クラスターエンドポイントは常に、クラスター内の現在のプライマリインスタンスを表します。クラスターエンドポイントの詳細については、「Amazon Aurora 接続管理」を参照してください。

> **Tip**
>
> 各 AWS リージョン 内では、アベイラビリティーゾーンは、停止時に分離できるよう、相互に異なる場所を示します。DB クラスターのプライマリインスタンスとリーダーインスタンスを複数のアベイラビリティーゾーンに配信して、DB クラスターの可用性を改善することをお勧めします。そうすることで、アベイラビリティーゾーン全体に影響する問題によりクラスターの停止が引き起こされることはありません。
> マルチ AZ DB クラスターをセットアップするには、クラスターの作成時に簡単な選択を行います。AWS Management Console、AWS CLI、または Amazon RDS API を使用できま

す。また、既存の Aurora DB クラスターをマルチ AZ DB クラスターに転換するには、新しいリーダー DB インスタンスを追加し、別のアベイラビリティーゾーンを指定します。

Aurora グローバルデータベースを使用した AWS リージョン間での高可用性

複数の AWS リージョン で高可用性を実現するために、Aurora グローバルデータベースを設定できます。各 Aurora グローバルデータベースは複数の AWS リージョン にまたがっているため、低レイテンシーでのグローバル読み取りが実現し、AWS リージョン 全体の停止からの災害対策を可能とします。Aurora は、プライマリ AWS リージョン からのすべてのデータと更新を、各セカンダリリージョンに対し自動的にレプリケートします。詳しくは、「Amazon Aurora Global Database の使用」を参照してください。

Aurora DB クラスターの耐障害性

Aurora DB クラスターは、耐障害性を持つように設計されています。クラスターボリュームは 1 つの AWS リージョン 内の複数のアベイラビリティーゾーン (AZ) にまたがり、各アベイラビリティーゾーンにはクラスターボリュームデータのコピーが含まれます。この機能は、DB クラスターがデータ喪失なしでアベイラビリティーゾーンの障害に耐えることができ、発生するのはサービスの短時間の中断のみであることを意味します。

DB クラスターのプライマリインスタンスが失敗した場合、Aurora は次の 2 つの方法のいずれかで、新しいプライマリインスタンスに自動的にフェイルオーバーします。

- 既存の Aurora レプリカを新しいプライマリインスタンスに昇格する
- 新しいプライマリインスタンスを作成する

DB クラスターに 1 つ以上の Aurora レプリカがある場合は、障害発生中に 1 つの Aurora レプリカがプライマリインスタンスに昇格されます。障害イベントによって短い中断が発生し、その間例外によって読み取りと書き込みオペレーションが失敗します。ただし、サービスの一般的な復元時間は 60 秒未満であり、多くの場合 30 秒未満です。DB クラスターの可用性を高めるために、複数のアベイラビリティーゾーン内で少なくとも 1 つ以上の Aurora レプリカを作成することをお勧めします。

> **Tip**
>
> Aurora MySQL 2.10 以降では、クラスター内に複数のリーダー DB インスタンスを持つことで、フェイルオーバー中の可用性を向上させることができます。Aurora MySQL 2.10 以降で

は、Aurora は、ライター DB インスタンスとフェイルオーバー先のリーダーインスタンスのみを再起動します。クラスター内の他のリーダー DB インスタンスは、フェイルオーバー中も使用可能であり、リーダーエンドポイントへの接続を通じてクエリの処理を続行します。また、Aurora DB クラスターで RDS Proxy を使用することで、フェイルオーバー中の可用性を向上させることができます。詳細については、「Amazon RDS Proxy における高可用性」を参照してください。

各レプリカに優先度を割り当てることで、Aurora レプリカがプライマリインスタンスに昇格される順序をカスタマイズできます。優先度の範囲は、最も高い 0 から最も低い 15 までです。プライマリインスタンスが失敗した場合、Amazon RDS は優先度が高い Aurora Replica を新しいプライマリインスタンスに昇格します。Aurora レプリカの優先度はいつでも変更できます。優先度を変更しても、フェイルオーバーはトリガーされません。

複数の Aurora レプリカで同じ優先度を共有でき、その場合は昇格階層が発生します。複数の Aurora レプリカで同じ優先度を共有する場合、Amazon RDS は最大サイズのレプリカを昇格します。複数の Aurora レプリカで同じ優先度とサイズを共有する場合、Amazon RDS は同じ昇格階層の任意のレプリカを昇格します。

DB クラスターに Aurora レプリカが含まれていない場合、障害イベントの発生時にプライマリインスタンスが同じ AZ に再作成されます。障害イベントによって中断が発生し、その間例外によって読み取りと書き込みオペレーションが失敗します。新しいプライマリインスタンスが再作成されると、サービスが回復します。これは、通常は 10 分未満で行われます。Aurora レプリカのプライマリインスタンスへの昇格は、新しいプライマリインスタンスの作成よりもはるかに短時間で実行されます。

AZ 全体に影響する停止のため、クラスター内のプライマリインスタンスが使用できないとします。この場合、新しいプライマリインスタンスをオンラインにする方法は、クラスターでマルチ AZ 設定を使用するかどうかによって異なります。

- プロビジョン済みまたは Aurora Serverless v2 クラスターに他の AZ のリーダーインスタンスが含まれている場合、Aurora はフェイルオーバーメカニズムを使用して、それらのリーダーインスタンスのいずれかを新しいプライマリインスタンスに昇格させます。
- プロビジョン済みまたは Aurora Serverless v2 クラスターに 1 つの DB インスタンスしか含まれていない場合、またはプライマリインスタンスとすべてのリーダーインスタンスが同じ AZ にある場合は、別の AZ に 1 つまたは複数の新しい DB インスタンスを手動で作成する必要があります。
- クラスターが Aurora Serverless v1 を使用する場合、Aurora は別の AZ に新しい DB インスタンスを自動的に作成します。ただし、このプロセスにはホストの交換が必要であるため、フェイルオーバーよりも時間がかかります。

> **Note**
>
> Amazon Aurora では、外部 MySQL データベースまたは RDS MySQL DB インスタンスとのレプリケーションもサポートします。詳細については、「Aurora と MySQL との間、または Aurora と別の Aurora DB クラスターとの間のレプリケーション (バイナリログレプリケーション)」を参照してください。

Amazon RDS Proxy における高可用性

RDS Proxy を使用すると、複雑な障害処理コードを記述しなくても、データベースの障害を透過的に許容できるアプリケーションを構築できます。プロキシでは、アプリケーション接続を維持したまま、新しいデータベースインスタンスにトラフィックを自動的にルーティングします。また、ドメインネームシステム (DNS) キャッシュをバイパスすることで、Aurora マルチ AZ データベースのフェイルオーバー時間を最大 66% 短縮できます。詳細については、「Amazon RDS Proxy for Aurora の使用」を参照してください。

Amazon Aurora でのレプリケーション

Aurora ではレプリケーションオプションをいくつかご用意しています。各 Aurora DB クラスターには、同じクラスター内の、複数の DB インスタンス間のレプリケーションが組み込まれています。Aurora クラスターをソースまたはターゲットにして、レプリケーションをセットアップすることもできます。Aurora クラスターの中にデータをレプリケートするか、クラスターからデータをレプリケートするときは、Aurora グローバルデータベースのような組み込み機能か、MySQL または PostgreSQL DB エンジンの従来のレプリケーションメカニズムの、いずれかを選択できます。適切なオプションを選択し、それに基づいて、必要に応じて高可用性、利便性、パフォーマンスを適切に組み合わせることができます。以下のセクションでは、各技術を選択する方法およびタイミングについて説明します。

トピック

- Aurora レプリカ
- Aurora MySQL でのレプリケーション
- Aurora PostgreSQL でのレプリケーション

Aurora レプリカ

Aurora プロビジョンド DB クラスターに 2 つめ、3 つめ、と DB インスタンスを作成すると、Aurora が、書き込み DB インスタンスからその他すべての DB インスタンスに、レプリケーションを自動でセットアップします。これらその他の他の DB インスタンスは、読み取り専用で、Aurora レプリカと呼ばれます。また、クラスター内で書き込みと読み取りの DB インスタンスを組み合わせる方法について説明する際は、リーダーインスタンスとも呼ばれます。

Aurora レプリカには主な目的が 2 つあります。これらにクエリを発行することで、アプリケーションの読み取り操作をスケールできます。これは、通常はクラスターのリーダーエンドポイントに接続することで行われます。これにより、Aurora は、読み取り専用の接続負荷を、クラスター内にある Aurora レプリカと同じ数だけ分散できます。Aurora レプリカは、可用性の向上にも役立ちます。クラスター内のライターインスタンスが使用できなくなると、Aurora はリーダーインスタンスのうちの 1 つを自動的に昇格させ、新しいライターとして機能させます。

Aurora DB クラスターは、最大で 15 Aurora のレプリカを含むことができます。Aurora レプリカは、AWS リージョン内で DB クラスターが使用している、複数のアベイラビリティーゾーン間に分散できます。

DB クラスターのデータには、クラスター内の DB インスタンスとは無関係に、独自の高可用性と信頼性機能があります。Aurora ストレージ機能に慣れていない方は、「Amazon Aurora ストレージの概要」を参照してください。DB クラスターボリュームは、物理的に、DB クラスターのデータの複数のコピーで構成されます。DB クラスター内のプライマリインスタンスと Aurora レプリカは、クラスターボリューム内のデータを単一の論理ボリュームとして認識します。

この結果、すべての Aurora レプリカは、最小のレプリカラグでクエリの結果として同じデータを返します。このラグは、通常はプライマリインスタンスが更新を書き込んだ後、100 ミリ秒未満です。レプリカラグは、データベースの変更レートによって異なります。つまり、データベースに対して大量の書き込みオペレーションが発生している間、レプリカラグが増加することがあります。

> **Note**
>
> Aurora レプリカは、次の Aurora PostgreSQL バージョンでライター DB インスタンスとの通信を 60 秒以上失うと再起動します。
>
> - 14.6 以前のバージョン
> - 13.9 以前のバージョン
> - 12.13 以前のバージョン

- すべての Aurora PostgreSQL 11 バージョン

Aurora レプリカは、クラスターボリュームでの読み取りオペレーションに特化しているため、読み取りのスケーリングに最適です。書き込みオペレーションはプライマリインスタンスによって管理されます。クラスターボリュームは DB クラスター内のすべての DB インスタンスで共有されるため、Aurora レプリカごとにデータのコピーをレプリケートする際に必要な追加作業は最小限に抑えられます。

可用性を高めるために、フェイルオーバーターゲットとして Aurora レプリカを使用できます。つまり、プライマリインスタンスが失敗した場合、Aurora レプリカがプライマリインスタンスに昇格されます。短い中断があり、その間はプライマリインスタンスに対して行われた読み取りおよび書き込みリクエストは、例外により失敗します。

フェイルオーバーによって Aurora レプリカを昇格する方が、プライマリインスタンスを再作成するよりも、時間を短縮できます。Aurora DB クラスターに Aurora レプリカが含まれていない場合、DB クラスターは、DB インスタンスが障害から復元している間、使用できなくなります。

フェイルオーバーが発生すると、DB エンジンのバージョンによっては、一部の Aurora レプリカが再起動されます。例えば、Aurora MySQL 2.10 以降では、Aurora は、フェイルオーバー時にのみ、ライター DB インスタンスとフェイルオーバーターゲットを再起動します。Aurora DB エンジンバージョンの再起動動作については、「Amazon Aurora DB クラスターまたは Amazon Aurora DB インスタンスの再起動」を参照してください。再起動またはフェイルオーバー時にページキャッシュがどうなるかについては、「存続できるページキャッシュ」を参照してください。

高可用性のシナリオでは、1 つ以上の Aurora レプリカを作成することをお勧めします。これらのレプリカは、プライマリインスタンスと同じ DB インスタンスクラスとし、Aurora DB クラスターの異なるアベイラビリティーゾーンに配置します。フェイルオーバーターゲットとしての Aurora レプリカの詳細については、「Aurora DB クラスターの耐障害性」を参照してください。

暗号化されていない Aurora DB クラスター用に暗号化された Aurora レプリカを作成することはできません。暗号化された Aurora DB クラスター用に暗号化されていない Aurora レプリカを作成することはできません。

> **Tip**
>
> Aurora クラスター内の Aurora レプリカをレプリケーションの唯一の形式として使用すると、データの高可用性を維持できます。また、組み込みの Aurora レプリケーションを他の

種類のレプリケーションと組み合わせることも可能です。これにより、データの可用性と地理的ディストリビューションのレベルをさらに高めることができます。

Aurora レプリカを作成する方法については、「DB クラスターに Aurora レプリカを追加する」を参照してください。

Aurora MySQL でのレプリケーション

Aurora レプリカに加えて、Aurora MySQL でのレプリケーションには次のオプションがあります。

- さまざまな AWS リージョンの Aurora MySQL DB クラスター。
  - Aurora グローバルデータベースを使用することで、複数のリージョンをまたいでデータをレプリケートできます。詳細については、「Aurora グローバルデータベースを使用した AWS リージョン間での高可用性」を参照してください。
  - MySQL バイナリログ (binlog) レプリケーションを使用することで、別の AWS リージョンに、Aurora MySQL DB クラスターの Aurora リードレプリカを作成できます。各クラスターは、この方法で最大 5 つのリードレプリカを、それぞれ異なるリージョンに作成できます。
- 同一の リージョン内の 2 つの Aurora MySQL DB クラスター (MySQL バイナリログ (binlog) のレプリケーションを使用)。
- データのソースとしての RDS for MySQL DB インスタンスと Aurora MySQL DB クラスター (RDS for MySQL DB インスタンスの Aurora リードレプリカを作成)。通常、この方法は、進行中のレプリケーションではなく、Aurora MySQL への移行に時に使用されます。

Aurora MySQL でのレプリケーションの詳細については、「Amazon Aurora MySQL でのレプリケーション」を参照してください。

Aurora PostgreSQL でのレプリケーション

Aurora レプリカに加えて、Aurora PostgreSQL を使ったレプリケーションには以下のオプションがあります。

- 1 つのリージョンと、異なるリージョンに置かれた最大 5 つの読み取り専用セカンダリ DB クラスター内にある Aurora プライマリ DB クラスター (Aurora グローバルデータベースを使用)。Aurora PostgreSQL では、クロスリージョン Aurora レプリカはサポートしていません。ただし、Aurora グローバルデータベースを使用して、Aurora PostgreSQL DB クラスターの読み取り機能を複

数の AWS リージョンに拡張し、可用性の目標を達成することができます。詳細については、「Amazon Aurora Global Database の使用」を参照してください。

- 同じリージョンにある 2 つの Aurora PostgreSQL DB クラスター (PostgreSQL の論理レプリケーション機能を使用)。
- データのソースとしての RDS for PostgreSQL DB インスタンスと Aurora PostgreSQL DB クラスター (RDS for PostgreSQL DB インスタンスの Aurora リードレプリカを作成)。通常、この方法は、進行中のレプリケーションではなく、Aurora PostgreSQL への移行時に使用されます。

Aurora PostgreSQL でのレプリケーションの詳細については、「Amazon Aurora PostgreSQL でのレプリケーション」を参照してください。

Aurora 向け DB インスタンスの請求

Amazon Aurora クラスターの Amazon RDS プロビジョンド インスタンスは、以下のコンポーネントに基づいて請求されます。

- DB インスタンス時間 (1 時間あたり) - DB インスタンスの DB インスタンスクラス (db.t2.small や db.m4.large など) に基づきます。料金は 1 時間単位で表示されますが、請求の計算方法には秒単位が適用され、時間は 10 進数の形式で表示されます。RDS の使用料は 1 秒ごとに課金され、10 分未満の場合は 10 分の料金が発生します。詳細については、「Aurora DB インスタンスクラス」を参照してください。
- ストレージ (1 か月あたりの GiB) - DB インスタンスにプロビジョニングしたストレージ容量。準備したストレージ容量を当月以内に拡張した場合、請求は比例配分されます。詳細については、「Amazon Aurora ストレージと信頼性」を参照してください。
- 入出力 (I/O) リクエスト (1 か月あたり 100 万リクエスト) - 請求期間内に行ったストレージ I/O リクエストの合計数。 Aurora Standard DB クラスター設定に対するものに限ります。

  Amazon Aurora I/O 請求の詳細については、「Amazon Aurora DB クラスターのストレージ設定」を参照してください。
- バックアップストレージ (1 か月あたりの GiB) - バックアップストレージは、自動データベースバックアップおよび作成したアクティブなデータベースのスナップショットに関連付けられているストレージです。バックアップ保持期間を延長するか、追加のデータベーススナップショットを撮ると、データベースが消費するバックアップストレージが増加します。1 秒単位の請求はバックアップストレージには適用されません (1 か月あたり GB 単位で請求されます)。

  詳細については、「Amazon Aurora DB クラスターのバックアップと復元」を参照してください。

- データ転送 (GB あたり) - DB インスタンスと、インターネットおよび AWS リージョンの間で送受信されるデータ転送。

Amazon RDS には、ニーズに基づいてコストを最適化するための以下の購入オプションがあります。

- オンデマンドインスタンス - 使用した DB インスタンス時間に対して時間単位でお支払いいただきます。料金は 1 時間単位で表示されますが、請求の計算方法には秒単位が適用され、時間は 10 進数の形式で表示されます。現在、RDS の使用料は 1 秒ごとに課金され、10 分未満の場合は 10 分の料金が発生します。
- リザーブドインスタンス - DB インスタンスを 1 年間または 3 年間予約することで、オンデマンド DB インスタンスの料金と比べて大幅な割引が得られます。リザーブドインスタンスの使用状況では 1 時間以内に複数のインスタンスを起動、削除、スタート、終了することができ、すべてのインスタンスにおいてリザーブドインスタンスのメリットが得られます。
- Aurora Serverless v2 – Aurora Serverless v2 はオンデマンド容量を提供しますが、これは請求単位が DB インスタンス時間ではなく Aurora 容量単位 (ACU) 時間であり、Aurora Serverless v2 容量は、データベースの負荷に応じて、指定した範囲内で増減します。すべての容量が Aurora Serverless v2 であるクラスターを設定できます。または Aurora Serverless v2 とオンデマンドまたは予約済みのプロビジョニングされたインスタンスの組み合わせを設定できます。Aurora Serverless v2 ACU が動作する仕組みの詳細については、「Aurora Serverless v2 の働き」を参照してください。

Aurora の料金情報については、Aurora の料金表ページを参照してください。

トピック

- Aurora 向けオンデマンド DB インスタンス
- Aurora 向けリザーブド DB インスタンス

Aurora 向けオンデマンド DB インスタンス

Amazon RDS オンデマンド DB インスタンスは、DB インスタンスのクラス (db.t3.small や db.m5.large など) に基づいて請求されます。Amazon RDS の料金情報については、Amazon RDS の製品ページを参照してください。

DB インスタンスの課金は DB インスタンスが利用可能になった時点からスタートされます。料金は 1 時間単位で表示されますが、請求の計算方法には秒単位が適用され、時間は 10 進数の形式で表示されます。Amazon RDS の使用料は 1 秒ごとに課金され、10 分未満の場合は 10 分の料金が発生します。請求可能な設定の変更 (例: コンピューティング容量またはストレージ容量のスケーリング) の場合は、10 分の料金が請求されます。課金は DB インスタンスが終了するまで継続します。終了とは、DB インスタンスが削除された場合、または DB インスタンスに障害が発生した場合です。

DB インスタンスに対する課金が不要になった場合は、これ以上 DB インスタンス時間に請求が行われないようにインスタンスを停止するか、削除する必要があります。課金される DB インスタンスの状態に関する詳細については、「Aurora クラスター内の DB インスタンスのステータスの表示」を参照してください。

停止した DB インスタンス

DB インスタンスが停止していても、プロビジョンド IOPS を含むプロビジョニング済みストレージに対して課金されます。また、指定された保持ウィンドウ内の手動スナップショットや自動バックアップのストレージを含むバックアップストレージに対しても課金されます。DB インスタンス時間に対しては請求されません。

マルチ AZ DB インスタンス

DB インスタンスがマルチ AZ 配置になるように指定する場合、Amazon RDS 料金ページに記載されたマルチ AZ 料金表に従って課金されます。

Aurora 向けリザーブド DB インスタンス

リザーブド DB インスタンスを使用することで、DB インスタンスを 1 年間または 3 年間予約できます。オンデマンド DB インスタンスの料金と比べて、リザーブド DB インスタンスには大幅な割引が適用されます。リザーブド DB インスタンスは物理インスタンスと言うよりも、アカウントで特定のオンデマンド DB インスタンスを使用した場合に適用される請求の割引と言えます。リザーブド DB インスタンスの割引は、インスタンスタイプと AWS リージョン に関連付けられています。

リザーブド DB インスタンスの一般的な使用プロセスとしては、まず使用可能なリザーブド DB インスタンスのタイプに関する情報を取得します。次に、該当するタイプのリザーブド DB インスタンスを購入します。最後に、既存のリザーブド DB インスタンスに関する情報を取得します。

リザーブド DB インスタンスの概要

Amazon RDS のリザーブド DB インスタンスを購入すると、このリザーブド DB インスタンスの該当期間中、特定の DB インスタンスタイプに対して割引料金が適用されます。Amazon RDS のリザーブド DB インスタンスを使用するには、オンデマンドインスタンスの場合と同様に、新しい DB インスタンスを作成します。

新しく作成する DB インスタンスの仕様は、次のリザーブド DB インスタンスの仕様と同じである必要があります。

- AWS リージョン
- DB エンジン
- DB インスタンスのタイプ

新しい DB インスタンスの仕様がアカウント内の既存のリザーブド DB インスタンスと一致する場合は、リザーブド DB インスタンスに適用される割引料金で請求されます。一致しない場合、DB インスタンスはオンデマンド料金で請求されます。

リザーブド DB インスタンスとして使用している DB インスタンスを変更できます。変更がリザーブド DB インスタンスの仕様の範囲内である場合、割引の一部またはすべてが、変更された DB インスタンスに適用されます。インスタンスクラスの変更など、変更が仕様の範囲外である場合、割引は適用されません。詳細については、「サイズ柔軟なリザーブド DB インスタンス」を参照してください。

トピック

- 提供タイプ

- Aurora DB クラスター設定の柔軟性
- サイズ柔軟なリザーブド DB インスタンス
- Aurora リザーブド DB インスタンスの請求例
- リザーブド DB インスタンスの削除

リザーブド DB インスタンスの料金などの詳細については、Amazon RDS リザーブドインスタンスを参照してください。

提供タイプ

リザーブド DB インスタンスには、予想される使用量に基づいて Amazon RDS のコストを最適化するための 3 種類のオプション — 前払いなし、一部前払い、全前払い — があります。

前払いなし

このオプションは前払い料金なしでリザーブド DB インスタンスへのアクセスを提供します。前払いなしのリザーブド DB インスタンスでは、使用量にかかわらず、期間内の時間はすべて、割引された時間料金で請求されます。前払い料金は必要ありません。このオプションは、1 年間の予約でのみ利用できます。

一部前払い

このオプションでは、リザーブド DB インスタンスの一部を前払いする必要があります。期間内の残りの時間は、使用量にかかわらず、割引された時間料金で請求されます。このオプションは、以前の "重度使用" オプションに代わるオプションです。

全前払い

期間のスタート時に全額を支払います。使用時間数に関係なく、残りの期間にそれ以外のコストは生じません。

一括請求を使用している場合、組織内のすべてのアカウントが 1 つのアカウントとして扱われます。これは、組織内のすべてのアカウントが、他のアカウントで購入したリザーブド DB インスタンスの時間単位のコスト利点を受けることができるということを意味しています。一括請求 (コンソリデーティッドビリング) の詳細については、AWS 請求情報とコスト管理ユーザーガイドの「Amazon RDS リザーブド DB インスタンス」を参照してください。

Aurora DB クラスター設定の柔軟性

Aurora リザーブド DB インスタンスは、両方の DB クラスター設定で使用できます。

- Aurora I/O-Optimized – お支払いいただくのは DB クラスターの使用量とストレージに対してのみで、読み取りと書き込み I/O オペレーションに追加料金はかかりません。
- Aurora Standard – DB クラスターの使用量とストレージに加えて、I/O オペレーションの 100 万件のリクエストごとに標準料金をお支払いいただきます。

Aurora はこれらの構成間の価格差を自動的に計算します。 Aurora I/O-Optimized では、1 時間あたりの正規化ユニットの消費量が Aurora Standard より 30% 増えます。

Aurora クラスターストレージ設定の詳細については、「Amazon Aurora DB クラスターのストレージ設定」を参照してください。Aurora クラスターストレージの料金情報については、「Amazon Aurora の料金」を参照してください。

サイズ柔軟なリザーブド DB インスタンス

リザーブド DB インスタンスを購入する際、指定する項目の 1 つはインスタンスクラス (db.r5.large など) です。DB インスタンスクラスの詳細については、「Aurora DB インスタンスクラス」を参照してください。

既存の DB インスタンスがあり、これをスケールして容量を増やす必要がある場合、リザーブド DB インスタンスはスケールした DB インスタンスに自動的に適用されます。つまり、リザーブド DB インスタンスは DB インスタンスクラスのすべてのサイズに自動的に適用されます。サイズに柔軟性のあるリザーブド DB インスタンスは、同じ AWS リージョン およびデータベースエンジンの DB インスタンスで利用できます。サイズ柔軟なリザーブド DB インスタンスは、そのインスタンスクラスタイプでしかスケールできません。例えば、db.r5.large のリザーブド DB インスタンスは db.r5.xlarge には適用できますが、db.r6g.large には適用できません。db.r5 と db.r6g は異なるインスタンスクラスタイプであるためです。

また、リザーブド DB インスタンスの利点はマルチ AZ およびシングル AZ の両設定に適用されます。柔軟性とは、同じ DB インスタンスクラスタイプ内の設定間を自由に移動できることを意味します。例えば、1 つのラージ DB インスタンス (1 時間あたりの正規化された単位 4) で実行されているシングル AZ 配置から、2 つのスモール DB インスタンス (1 時間あたりの正規化された単位 2*2=4) で実行されているマルチ AZ 配置に移行できます。

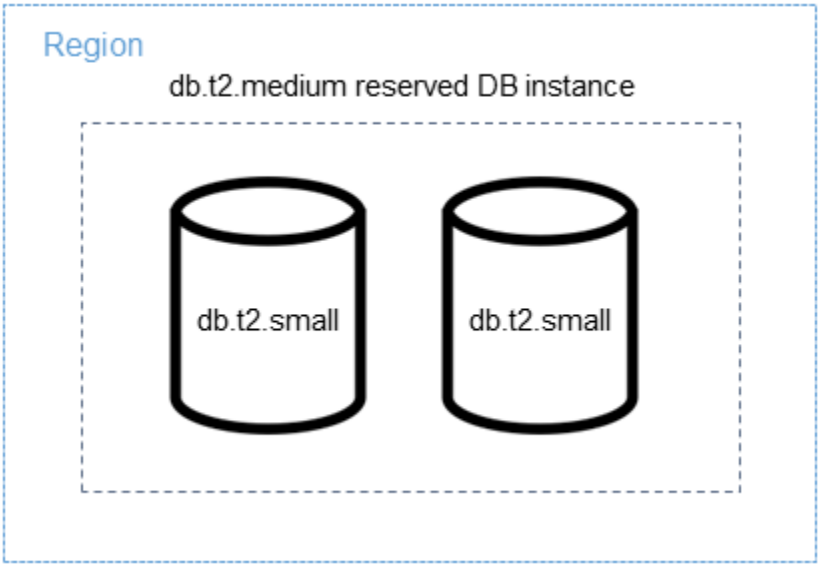
サイズ柔軟なリザーブド DB インスタンスは、以下の Aurora データベースエンジンで使用できます。

- Aurora MySQL
- Aurora PostgreSQL

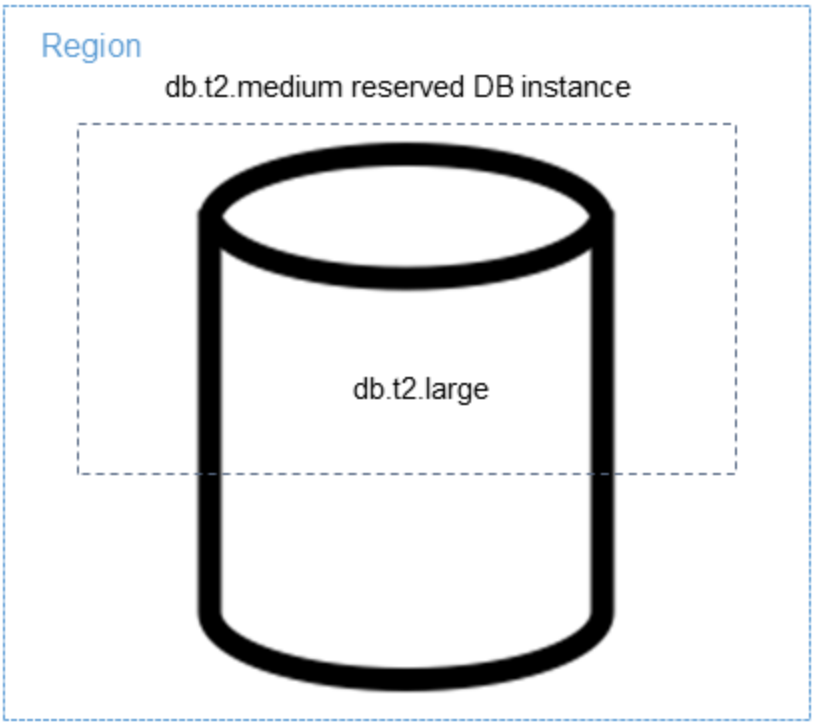
リザーブド DB インスタンスのサイズ別の使用は、1 時間あたりの正規化された単位を使用して比較できます。例えば、2 つの db.r3.large DB インスタンスでの 1 単位の使用は、1 つの db.r3.small での 1 時間あたりの正規化された単位 8 の使用に相当します。次の表は、DB インスタンスのサイズ別の 1 時間あたりの正規化された単位の数を示しています。

インスタンスサイズ	1 つの DB インスタンスの 1 時間あたりの正規化された単位 Aurora Standard	1 つの DB インスタンスの 1 時間あたりの正規化された単位 Aurora I/O-Optimized	3 つの DB インスタンス (ライターと 2 つのリーダー) の 1 時間あたりの正規化された単位、Aurora Standard	3 つの DB インスタンス (ライターと 2 つのリーダー) の 1 時間あたりの正規化された単位、Aurora I/O-Optimized
small	1	1.3	3	3.9
medium	2	2.6	6	7.8
large	4	5.2	12	15.6
xlarge	8	10.4	24	31.2
2xlarge	16	20.8	48	62.4
4xlarge	32	41.6	96	124.8
8xlarge	64	83.2	192	249.6
12xlarge	96	124.8	288	374.4
16xlarge	128	166.4	384	499.2
24xlarge	192	249.6	576	748.8
32xlarge	256	332.8	768	998.4

例えば、`db.t2.medium` リザーブド DB インスタンスを購入し、同じ AWS リージョン のアカウントで 2 つの `db.t2.small` DB インスタンスを実行しているとします。この場合、料金上の利点は両方のインスタンスに全面的に適用されます。



また、同じ AWS リージョン のアカウントで 1 つの db.t2.large インスタンスを実行している場合、この DB インスタンスの使用の 50 パーセント に支払い特典が適用されます。



> **Note**
>
> T DB インスタンスクラスは、開発およびテストサーバー、またはその他の本稼働以外のサーバーにのみ使用することをお勧めします。T インスタンスクラスの詳細については、「DB インスタンスクラスタイプ」を参照してください。

Aurora リザーブド DB インスタンスの請求例

以下の例は、Aurora I/O-Optimized と Aurora Standard DB クラスター設定の両方を使用した Aurora DB クラスターのリザーブド DB インスタンスの料金を示しています。

Aurora Standard を使用した場合の例

リザーブド DB インスタンスの価格で、ストレージ、バックアップ、および I/O に関連する通常のコストに対する割引は提供されません。次の例は、リザーブド DB インスタンスの月あたりのコスト合計を示しています。

- Aurora MySQL リザーブドシングル AZ db.r5.large DB インスタンスクラスのコストは、米国東部 (バージニア北部) の場合、1 時間あたり 0.19 USD、または 1 か月あたり 138.70 USD
- Aurora ストレージのコストは、1 か月 1 GiB あたり 0.10 USD (この例では 1 か月あたり 45.60 USD を想定)
- Aurora I/O のコストは、100 万 リクエストあたり 0.20 USD (この例では 1 か月あたり 20 USD を想定)
- Aurora バックアップストレージのコストは、1 か月 1 GiB あたり 0.021 USD (この例では 1 か月あたり 30 USD を想定)

リザーブド DB インスタンスにこれらすべてのオプション (138.70 USD + 45.60 USD + 20 USD + 30 USD) を加えると、1 か月あたりの総コストは 234.30 USD です。

リザーブド DB インスタンスの代わりにオンデマンド DB インスタンスを使用する場合、Aurora MySQL シングル AZ db.r5.large DB インスタンスクラスのコストは米国東部 (バージニア北部) では、1 時間あたり 0.29 USD、または 1 か月あたり 217.50 USD です。つまり、オンデマンド DB インスタンスの場合、これらすべてのオプション (217.50 USD + 45.60 USD + 20 USD + 30 USD) が加わり、1 か月あたりの総コストは 313.10 USD となります。リザーブド DB インスタンスを使用すると、月々約 79 ドルの節約になります。

2 つのリーダーインスタンスを持つ Aurora Standard DB クラスターを使用した場合の例

Aurora DB クラスターにリザーブドインスタンスを使用するには、クラスター内の DB インスタンスごとに 1 つのリザーブドインスタンスを購入するだけです。

最初の例を拡張して、1 つのライター DB インスタンスと 2 つの Aurora レプリカを含む Aurora MySQL DB クラスターがあり、クラスターには合計 3 つの DB インスタンスがあるとします。2 つの Aurora レプリカでは、追加のストレージ料金やバックアップ料金が発生しません。db.r5.large Aurora MySQL リザーブド DB インスタンスを 3 つ購入した場合、コストは 234.30 USD (ライター DB インスタンス) + 2 * (138.70 USD + Aurora レプリカあたり 20 USD の I/O) となり、合計で 1 か月あたり 551.70 USD になります。

1 つのライター DB インスタンスと 2 つの Aurora レプリカを含む Aurora MySQL DB クラスターに対応するオンデマンドコストは、313.10 USD + 2 * (インスタンスあたり 217.50 USD + 20 USD I/O) で、合計で 1 か月あたり 788.10 USD になります。リザーブド DB インスタンスを使用すると、月々 236.40 USD の節約になります。

Aurora I/O-Optimized を使用した場合の例

既存の Aurora Standard リザーブド DB インスタンスは Aurora I/O-Optimized で再利用できます。Aurora I/O-Optimized でリザーブドインスタンス割引のメリットを最大限に活用するには、現在のリザーブドインスタンスと同様、30% の追加リザーブドインスタンスを購入できます。

次の表は、Aurora I/O-Optimized を使用した場合に追加のリザーブドインスタンスを見積もる方法の例を示しています。必要なリザーブドインスタンスがごくわずかである場合、リザーブドインスタンスのサイズの柔軟性を活用して整数にすることができます。これらの例では、「現行」とは現在所有している Aurora Standard リザーブドインスタンスを指します。追加リザーブドインスタンスとは、Aurora I/O-Optimized 使用時に現在のリザーブドインスタンス割引を維持するために購入する必要がある Aurora Standard リザーブドインスタンスの数のことです。

DB インスタンスクラス	現行の Aurora Standard リザーブドインスタンス	Aurora I/O-Optimized に必要なリザーブドインスタンス	必要な追加のリザーブドインスタンス	サイズの柔軟性を使った、必要な追加のリザーブドインスタンス
db.r6g.large	10	10 * 1.3 = 13	3 * db.r6g.large	3 * db.r6g.large

DB インスタンスクラス	現行の Aurora Standard リザーブドインスタンス	Aurora I/O-Optimized に必要なリザーブドインスタンス	必要な追加のリザーブドインスタンス	サイズの柔軟性を使った、必要な追加のリザーブドインスタンス
db.r6g.4xlarge	20	20 * 1.3 = 26	6 * db.r6g.4xlarge	6 * db.r6g.4xlarge
db.r6g.12xlarge	5	5 * 1.3 = 6.5	1.5 * db.r6g.12xlarge	db.r6g.12xlarge、r6g.4xlarge、r6g.2xlarge をそれぞれ 1 つ (0.5 * db.r6g.12xlarge = 1 * db.r6g.4xlarge + 1 * db.r6g.2xlarge)
db.r6i.24xlarge	15	15 * 1.3 = 19.5	4.5 * db.r6i.24xlarge	4 * db.r6i.24xlarge + 1 * db.r6i.12xlarge (0.5 * db.r6i.24xlarge = 1 * db.r6i.12xlarge)

2 つのリーダーインスタンスを持つ Aurora I/O-Optimized DB クラスターを使用した場合の例

1 つのライター DB インスタンスと 2 つの Aurora レプリカを含む Aurora MySQL DB クラスターがあり、クラスターには合計 3 つの DB インスタンスがあるとします。Aurora I/O-Optimized DB クラスター設定が使用されます。このクラスターにリザーブド DB インスタンスを使用するには、同じ DB インスタンスクラスの 4 つのリザーブド DB インスタンスを購入する必要があります。Aurora I/O-Optimized を使用している 3 つの DB インスタンスは 1 時間あたり 3.9 の正規化された単位を消費

しますが、Aurora Standard を使用している 3 つの DB インスタンスでは 1 時間あたり 3 の正規化された単位を消費します。ただし、各 DB インスタンスの月間 I/O コストは節約できます。

> **Note**
>
> これらの例で説明しているのはサンプルの価格であり、実際の価格とは一致しない場合があります。Amazon Aurora の料金情報については、「Aurora の料金」を参照してください。

リザーブド DB インスタンスの削除

リザーブド DB インスタンスには 1 年契約と 3 年契約があります。リザーブド DB インスタンスをキャンセルすることはできません。ただし、リザーブド DB インスタンスの割引対象である DB インスタンスは削除できます。リザーブド DB インスタンスの割引対象である DB インスタンスの削除プロセスは、他の DB インスタンスの削除プロセスと同じです。

リソースを使用するかどうかにかかわらず、前払いコストが請求されます。

リザーブド DB インスタンスの割引対象である DB インスタンスを削除した場合、互換性がある仕様の別の DB インスタンスを起動できます。この場合、予約期間 (1 年または 3 年) 中、割引料金を利用できます。

リザーブド DB インスタンスを使用する

AWS Management Console、AWS CLI、および RDS API を使用して、リザーブド DB インスタンスを使用できます。

コンソール

リザーブド DB インスタンスを AWS Management Console で使用するには、次の手順に従います。

リザーブド DB インスタンス提供タイプの料金表と情報を取得するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[リザーブドインスタンス] を選択します。
3. [Purchase Reserved DB Instance] を選択します。
4. [製品の説明] で、DB エンジンとライセンスタイプを選択します。
5. [DB インスタンスクラス] で、DB インスタンスのクラスを選択します。

6. [デプロイオプション] で、シングル AZ または マルチ AZ DB インスタンスのデプロイが必要かどうかを選択します。

> **Note**
>
> リザーブド Amazon Aurora インスタンスのデプロイオプションは [シングル AZ DB インスタンス] に設定されます。ただし、Aurora DB クラスターを作成する場合、デフォルトのデプロイオプションは、[別の AZ に Aurora レプリカまたはリーダーノードを作成] (マルチ AZ) です。
> Aurora レプリカを含む、使用予定の各インスタンスについて、リザーブド DB インスタンスを購入する必要があります。そのため、Aurora でのマルチ AZ 配置では、リザーブド DB インスタンスを追加購入する必要があります。

7. [期間] で、DB インスタンスを予約する期間を選択します。
8. [提供タイプ] で、提供タイプを選択します。

   提供タイプを選択すると、料金情報が表示されます。

> **Important**
>
> リザーブド DB インスタンスの購入と料金の発生を防ぐには、[キャンセル] を選択します。

リザーブド DB インスタンス提供タイプに関する情報を取得したら、次の手順に従い、この情報を使用して提供タイプを購入できます。

リザーブド DB インスタンスを購入するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[リザーブドインスタンス] を選択します。
3. [Purchase Reserved DB Instance] (リザーブド DB インスタンスの購入) を選択します。
4. [製品の説明] で、DB エンジンとライセンスタイプを選択します。
5. [DB インスタンスクラス] で、DB インスタンスのクラスを選択します。
6. [マルチ AZ 配置] で、シングル AZ またはマルチ AZ DB インスタンス配置が必要かどうかを選択します。

> **Note**
>
> リザーブド Amazon Aurora インスタンスのデプロイオプションは [シングル AZ DB インスタンス] に設定されます。リザーブド DB インスタンスから Amazon Aurora DB クラスターを作成すると、DB クラスターは自動的にマルチ AZ として作成されます。Aurora レプリカを含む、使用予定の各 DB インスタンスについて、リザーブド DB インスタンスを購入する必要があります。

7. [Term] で、DB インスタンスを予約する期間を選択します。
8. [提供タイプ] で、提供タイプを選択します。

   提供タイプを選択すると、料金情報が表示されます。
9. (オプション) 購入したリザーブド DB インスタンスに独自の識別子を割り当てると、インスタンスを追跡しやすくなります。[Reserved Id] に、リザーブド DB インスタンスの識別子を入力します。
10. 送信 を選択します。

    リザーブド DB インスタンスを購入すると、リザーブドインスタンス リストに表示されます。

リザーブド DB インスタンスを購入したら、次の手順に従ってリザーブド DB インスタンスに関する情報を取得できます。

AWS アカウントのリザーブド DB インスタンスの情報を入手するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. [ナビゲーション] ペインで [リザーブドインスタンス] を選択します。

   アカウントのリザーブド DB インスタンスが表示されます。特定のリザーブド DB インスタンスに関する詳細な情報を確認するには、リストにあるそのインスタンスを選択します。これによって、コンソールの下部にある詳細ペインにそのインスタンスの詳細情報を表示できます。

AWS CLI

リザーブド DB インスタンスを AWS CLI で使用するには、以下の例に従います。

Example 使用可能なリザーブド DB インスタンスの提供タイプに関する情報を入手する

使用可能なリザーブド DB インスタンス提供タイプに関する情報を取得するには、AWS CLI コマンド [describe-reserved-db-instances-offerings] を呼び出します。

```
aws rds describe-reserved-db-instances-offerings
```

この呼び出しにより、以下のような出力が返されます。

```
OFFERING  OfferingId                            Class         Multi-AZ  Duration  Fixed
 Price  Usage Price  Description  Offering Type
OFFERING  438012d3-4052-4cc7-b2e3-8d3372e0e706  db.r3.large   y         1y
 1820.00 USD  0.368 USD    mysql        Partial  Upfront
OFFERING  649fd0c8-cf6d-47a0-bfa6-060f8e75e95f  db.r3.small   n         1y
 227.50 USD  0.046 USD    mysql        Partial  Upfront
OFFERING  123456cd-ab1c-47a0-bfa6-12345667232f  db.r3.small   n         1y
 162.00 USD   0.00 USD    mysql        All      Upfront
    Recurring Charges:   Amount  Currency  Frequency
    Recurring Charges:   0.123   USD       Hourly
OFFERING  123456cd-ab1c-37a0-bfa6-12345667232d  db.r3.large   y         1y
 700.00 USD   0.00 USD    mysql        All      Upfront
    Recurring Charges:   Amount  Currency  Frequency
    Recurring Charges:   1.25    USD       Hourly
OFFERING  123456cd-ab1c-17d0-bfa6-12345667234e  db.r3.xlarge  n         1y
 4242.00 USD   2.42 USD    mysql        No       Upfront
```

リザーブド DB インスタンス提供タイプに関する情報を取得したら、この情報を使用して提供タイプを購入できます。

リザーブド DB インスタンスを購入するには、以下のパラメータを指定して AWS CLI コマンド [purchase-reserved-db-instances-offering] を呼び出します。

- `--reserved-db-instances-offering-id` - 購入する提供タイプの ID。提供タイプの ID を取得するには、前の例を参照してください。
- `--reserved-db-instance-id` - 購入したリザーブド DB インスタンスに独自の識別子を割り当てると、インスタンスを追跡しやすくなります。

Example リザーブド DB インスタンスを購入する

次の例では、ID が *649fd0c8-cf6d-47a0-bfa6-060f8e75e95f* のリザーブド DB インスタンスを購入し、識別子として *MyReservation* を割り当てます。

Linux、macOS、Unix の場合:

```
aws rds purchase-reserved-db-instances-offering \
    --reserved-db-instances-offering-id 649fd0c8-cf6d-47a0-bfa6-060f8e75e95f \
    --reserved-db-instance-id MyReservation
```

Windows の場合:

```
aws rds purchase-reserved-db-instances-offering ^
    --reserved-db-instances-offering-id 649fd0c8-cf6d-47a0-bfa6-060f8e75e95f ^
    --reserved-db-instance-id MyReservation
```

このコマンドにより、以下のような出力が返されます。

```
RESERVATION  ReservationId      Class        Multi-AZ  Start Time
 Duration  Fixed Price  Usage Price  Count  State            Description  Offering Type
RESERVATION  MyReservation      db.r3.small  y         2011-12-19T00:30:23.247Z  1y
    455.00 USD   0.092 USD    1      payment-pending  mysql        Partial  Upfront
```

リザーブド DB インスタンスを購入したら、リザーブド DB インスタンスに関する情報を取得できます。

AWS アカウントのリザーブド DB インスタンスに関する情報を取得するには、以下の例に従って、AWS CLI コマンド describe-reserved-db-instances を呼び出します。

Example リザーブド DB インスタンスを取得する

```
aws rds describe-reserved-db-instances
```

このコマンドにより、以下のような出力が返されます。

```
RESERVATION  ReservationId     Class        Multi-AZ  Start Time
 Duration  Fixed Price  Usage Price  Count  State    Description  Offering Type
RESERVATION  MyReservation     db.r3.small  y         2011-12-09T23:37:44.720Z  1y
   455.00 USD   0.092 USD    1      retired  mysql        Partial  Upfront
```

RDS API

RDS API を使用して、リザーブド DB インスタンスを操作できます。

- 使用可能なリザーブド DB インスタンス提供タイプに関する情報を取得するには、Amazon RDS API オペレーション DescribeReservedDBInstancesOfferings を呼び出します。
- リザーブド DB インスタンス提供タイプに関する情報を取得したら、この情報を使用して提供タイプを購入できます。次のパラメータを指定して、PurchaseReservedDBInstancesOffering RDS API オペレーションを実行します。
  - `--reserved-db-instances-offering-id` - 購入する提供タイプの ID。
  - `--reserved-db-instance-id` - 購入したリザーブド DB インスタンスに独自の識別子を割り当てると、インスタンスを追跡しやすくなります。
- リザーブド DB インスタンスを購入したら、リザーブド DB インスタンスに関する情報を取得できます。DescribeReservedDBInstances RDS API オペレーション を呼び出します。

リザーブド DB インスタンスの請求を表示

リザーブド DB インスタンスの請求は、AWS Management Consoleの「請求ダッシュボード」 で表示できます。

リザーブド DB インスタンスの請求を表示する

1. AWS Management Consoleにサインインします。
2. 右上のアカウントメニューから請求ダッシュボードを選択します。
3. ダッシュボード右上の請求の詳細アイコンを選択します。
4. 「AWSサービス料金」で、リレーショナルデータベースサービスを展開します。
5. リザーブド DB インスタンスが置かれている AWS リージョン、例えば米国西部 (オレゴン) を展開します。

   リザーブド DB インスタンスと当月の時間単位の料金は、Amazon Relational Database Service##########リザーブドインスタンスに表示されます。

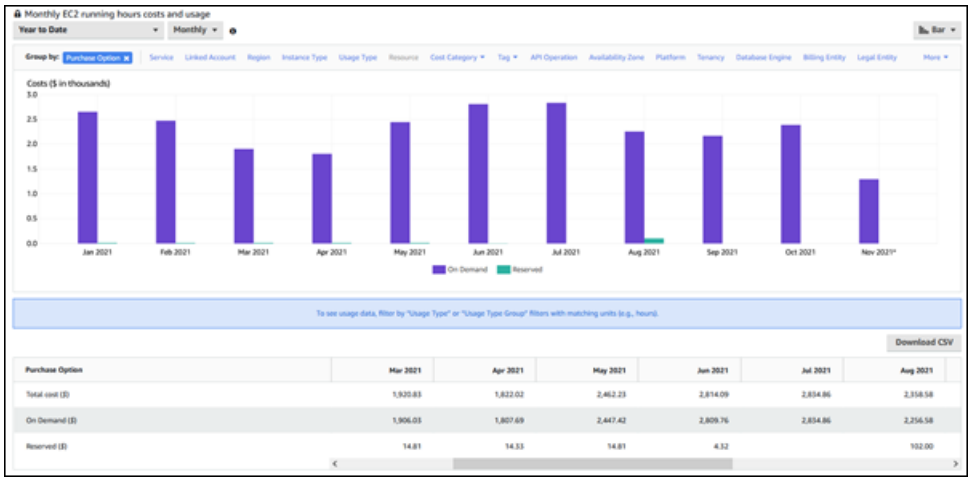
   

   この例では、リザーブド DB インスタンスは全額前払いで購入したため、時間単位の料金は発生しません。
6. リザーブドインスタンスの見出し横にある「Cost Explorer」(棒グラフ) アイコンを選択します。

Cost Explorer には、毎月の EC2 稼働時間のコストと使用状況グラフが表示されます。

7. グラフ右横の使用タイプグループフィルターをクリアします。
8. 使用コストを調べる期間と時間単位を選択します。

次の例は、オンデマンドおよびリザーブド DB インスタンスの年間使用コストを月単位で表示しています。



2021 年 1 月から 6 月までのリザーブド DB インスタンスの料金は、一部前払いのインスタンスの月額料金で、2021 年 8 月の料金は全額前払いのインスタンスの 1 回限りの料金です。

一部前払いのインスタンスのリザーブドインスタンス割引は 2021 年 6 月に有効期限が切れましたが、DB インスタンスは削除されませんでした。有効期限を過ぎてからは、オンデマンド料金で請求されました。

Amazon Aurora の環境をセットアップする

Amazon Aurora を初めて使用する場合は、事前に以下のタスクをすべて実行してください。

トピック

- AWS アカウントへのサインアップ
- 管理アクセスを持つユーザーを作成する
- プログラム的なアクセス権を付与する
- 要件の確認
- セキュリティグループを作成して VPC 内の DB クラスターへのアクセスを提供する

既存の AWS アカウント があり、Aurora の要件を理解していて、IAM と VPC セキュリティグループをデフォルト設定で使用したい場合には、Amazon Aurora の開始方法 にスキップできます。

AWS アカウントへのサインアップ

AWS アカウントがない場合は、以下のステップを実行して作成します。

AWS アカウントにサインアップするには

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

   AWS アカウントにサインアップすると、AWS アカウントのルートユーザーが作成されます。ルートユーザーには、アカウントのすべてのAWS のサービスとリソースへのアクセス権があります。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

サインアップ処理が完了すると、AWS からユーザーに確認メールが送信されます。https://aws.amazon.com/ の [マイアカウント] を選んで、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理できます。

管理アクセスを持つユーザーを作成する

AWS アカウント にサインアップしたら、AWS アカウントのルートユーザー をセキュリティで保護し、AWS IAM Identity Center を有効にして、管理ユーザーを作成します。これにより、日常的なタスクにルートユーザーを使用しないようにします。

AWS アカウントのルートユーザーをセキュリティで保護する

1. [ルートユーザー] を選択し、AWS アカウントのメールアドレスを入力して、アカウント所有者として AWS Management Console にサインインします。次のページでパスワードを入力します。

   ルートユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドの「ルートユーザーとしてサインインする」を参照してください。
2. ルートユーザーの多要素認証 (MFA) を有効にします。

   手順については、IAM ユーザーガイドの「AWS アカウント のルートユーザーの仮想 MFA デバイスを有効にする (コンソール)」を参照してください。

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Center の有効化」を参照してください。
2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

   IAM アイデンティティセンターディレクトリ をアイデンティティソースとして使用するチュートリアルについては、「AWS IAM Identity Center ユーザーガイド」の「デフォルト IAM アイデンティティセンターディレクトリを使用したユーザーアクセスの設定」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

  IAM Identity Center ユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドの「AWS アクセスポータルにサインインする」を参照してください。

追加のユーザーにアクセス権を割り当てる

1. IAM アイデンティティセンターで、最小特権のアクセス許可を適用するというベストプラクティスに従ったアクセス許可セットを作成します。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」を参照してください。
2. グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「グループの参加」を参照してください。

プログラム的なアクセス権を付与する

AWS Management Console の外部で AWS を操作するには、プログラマチックアクセス権が必要です。プログラマチックアクセス権を付与する方法は、AWS にアクセスしているユーザーのタイプによって異なります。

ユーザーにプログラマチックアクセス権を付与するには、以下のいずれかのオプションを選択します。

プログラマチックアクセス権を必要とするユーザー	目的	方法
ワークフォースアイデンティティ (IAM Identity Center で管理されているユーザー)	一時的な認証情報を使用して、AWS CLI、AWS SDK、または AWS API へのプログラマチックリクエストに署名します。	使用するインターフェイス用の手引きに従ってください。 • AWS CLI については、AWS Command Line Interface ユーザーガイドの「AWS IAM Identity Center を使用するための AWS CLI の設定」を参照してください。 • AWS SDK、ツール、および AWS API については、AWS SDK とツールリファレンスガイドの「IAM

プログラマチックアクセス権を必要とするユーザー	目的	方法
		Identity Center 認証」を参照してください。
IAM	一時的な認証情報を使用して、AWS CLI、AWS SDK、または AWS API へのプログラムによるリクエストに署名します。	「IAM ユーザーガイド」の「AWS リソースでの一時的な認証情報の使用」の指示に従ってください。
IAM	(非推奨) 長期的な認証情報を使用して、AWS CLI、AWS SDK、AWS API へのプログラムによるリクエストに署名します。	使用するインターフェイス用の手順に従ってください。 • AWS CLI については、AWS Command Line Interface ユーザーガイドの「IAM ユーザー認証情報を使用した認証」を参照してください。 • AWS SDK とツールについては、AWS SDK とツールリファレンスガイドの「長期認証情報を使用して認証する」を参照してください。 • AWS API については、IAM ユーザーガイドの「IAM ユーザーのアクセスキーの管理」を参照してください。

要件の確認

Aurora の基本的な構成要素は DB クラスターです。DB クラスターに 1 つ以上の DB インスタンスを属させることができます。DB クラスターによって、クラスターエンドポイントと呼ばれるネット

ワークアドレスが提供されます。アプリケーションは、DB クラスター内に作成されたデータベースへアクセスする必要がある場合、DB クラスターによって公開されたクラスターエンドポイントに接続します。DB クラスターの作成時に指定する情報によって、設定要素 (メモリ、データベースエンジンとバージョン、ネットワーク設定、セキュリティ、メンテナンス時間など) が制御されます。

DB クラスターとセキュリティグループを作成する前に、DB クラスターとネットワークに関する要件を理解しておく必要があります。重要な留意事項を以下に示します。

- リソース要件 – アプリケーションまたはサービスに関するメモリとプロセッサの要件。DB クラスターの作成時にどのような DB インスタンスクラスを使用するかを決定する場合に、これらの設定が使用されます。DB インスタンスクラスの仕様については、「Aurora DB インスタンスクラス」を参照してください。
- VPC、サブネット、およびセキュリティグループ – DB クラスターは仮想プライベートクラウド (VPC) に配置されます。セキュリティグループルールは、DB クラスターに接続するように設定される必要があります。各 VPC オプションに関するルールを次に説明します。
  - デフォルトの VPC — AWS アカウントに AWS リージョン内のデフォルトの VPC がある場合、その VPC は DB クラスターをサポートするように設定されます。DB クラスターの作成時にデフォルトの VPC を指定する場合は、次の操作を行います。
    - アプリケーションやサービスから Aurora DB クラスターへの接続を許可する VPC セキュリティグループを必ず作成します。VPC セキュリティグループを作成するには、VPC コンソールの [Security Group] (セキュリティグループ) オプションまたは AWS CLI を使用します。詳細については、「ステップ 3: VPC セキュリティグループを作成する」を参照してください。
    - デフォルトの DB サブネットグループを指定する必要があります。これが AWS リージョン内に作成する最初の DB クラスターである場合、そのクラスターの作成時に、Amazon RDS は、デフォルトの DB サブネットグループを作成します。
  - ユーザー定義の VPC — DB クラスターの作成時にユーザー定義の VPC を指定する場合は、次の操作を行います。
    - アプリケーションやサービスから Aurora DB クラスターへの接続を許可する VPC セキュリティグループを必ず作成します。VPC セキュリティグループを作成するには、VPC コンソールの [Security Group] (セキュリティグループ) オプションまたは AWS CLI を使用します。詳細については、「ステップ 3: VPC セキュリティグループを作成する」を参照してください。
    - DB クラスターをホストするには、VPC は特定の要件 (2 つ以上のサブネットを保持しており、各サブネットは個別のアベイラビリティーゾーン内にあることなど) を満たす必要があります。詳細については、「Amazon VPC VPC とAmazon Aurora」を参照してください。

  - DB クラスターで使用できる VPC 内のサブネットを定義する DB サブネットグループを指定する必要があります。詳細については、「VPC 内の DB クラスターの使用」の「DB サブネットグループ」のセクションを参照してください。
- 高可用性: フェイルオーバーサポートが必要かどうか。Aurora のマルチ AZ 配置ではプライマリインスタンスと Aurora レプリカが作成されます。プライマリインスタンスと Aurora レプリカをフェイルオーバーサポートのために異なるアベイラビリティーゾーンに設定することができます。本番稼働用のワークロードには、高可用性を維持するためにマルチ AZ 配置をお勧めします。開発およびテストの目的では、マルチ AZ 配置以外のデプロイを使用できます。詳細については、「Amazon Aurora の高可用性」を参照してください。
- IAM ポリシー: Amazon RDS オペレーションの実行に必要なアクセス許可を付与するためのポリシーが、自分の AWS アカウントにあるかどうか。IAM 認証情報を使用して AWS に接続している場合、IAM アカウントには、Amazon RDS オペレーションの実行するためのアクセス許可を付与する IAM ポリシーが必要です。詳細については、「Amazon Aurora での Identity and Access Management」を参照してください。
- Open ports: データベースがリッスンするのは、どの TCP/IP ポートであるか。一部の企業のファイアウォールでは、データベースエンジン用のデフォルトポートへの接続がブロックされる場合があります。会社のファイアウォールがデフォルトのポートをブロックする場合は、新しい DB クラスター用に別のポートを選択します。指定したポートをリッスンする DB クラスターを作成すると、DB クラスターを変更することでポートを変更できます。
- AWS リージョン: データベースが必要となる AWS リージョン。アプリケーションやウェブサービスの近くにデータベースを配置すると、ネットワークレイテンシーが低減されます。詳細については、「リージョンとアベイラビリティーゾーン」を参照してください。

セキュリティグループと DB クラスターの作成に必要な情報を把握したら、次のステップに進みます。

セキュリティグループを作成して VPC 内の DB クラスターへのアクセスを提供する

DB クラスターは VPC 内に作成されます。セキュリティグループは、VPC 内の DB クラスターへのアクセスを提供します。セキュリティグループは、関連付けられた DB クラスターのファイアウォールとして動作し、インバウンドトラフィックとアウトバウンドトラフィックの両方をクラスターレベルで制御します。DB クラスターはデフォルトでファイアウォールによって作成され、DB クラスターへのアクセスを禁止するデフォルトのセキュリティグループとなります。このため、DB クラスターへの接続を可能にするルールをセキュリティグループに追加する必要があります。前のステッ

プで決定したネットワークと設定に関する情報を使用して、DB クラスターへのアクセスを許可するルールを作成します。

例えば、アプリケーションが VPC 内にある DB クラスター上のデータベースにアクセスする場合、アプリケーションがデータベースにアクセスする際に使用するポート範囲と IP アドレスが指定された、カスタム TCP ルールを追加する必要があります。アプリケーションが Amazon EC2 インスタンスにある場合は、Amazon EC2 インスタンスに設定した VPC セキュリティグループを使用できます。

DB クラスターの作成時に、Amazon EC2 インスタンスと DB クラスター間の接続を設定できます。詳しくは、「EC2 インスタンスとの自動ネットワーク接続を設定する」を参照してください。

> **Tip**
>
> DB クラスターの作成時に、Amazon EC2 インスタンスと DB インスタンスクラスター間で自動的にネットワーク接続を設定できるようになります。詳しくは、「EC2 インスタンスとの自動ネットワーク接続を設定する」を参照してください。

Aurora で使用するための VPC を作成する方法の詳細については、チュートリアル: DB クラスターで使用する VPC を作成する (IPv4 専用) を参照してください。DB インスタンスにアクセスするための一般的なシナリオについては、VPC の DB クラスターにアクセスするシナリオ を参照してください。

VPC セキュリティグループを作成するには

1. AWS Management Console にサインインして、Amazon VPC コンソール (https://console.aws.amazon.com/vpc) を開きます。

   > **Note**
   >
   > RDS コンソールではなく VPC コンソールにアクセスしていることを確認します。

2. AWS Management Console の右上で、VPC セキュリティグループと DB クラスターを作成する先の AWS リージョンを選択します。この AWS リージョンにある Amazon VPC リソースのリストには、少なくとも 1 の VPC と複数のサブネットが表示されます。表示されない場合、この AWS リージョンにはデフォルト VPC がありません。
3. ナビゲーションペインで、[Security Groups] を選択します。
4. セキュリティグループの作成 を選択します。

[Create security group] (セキュリティグループの作成) ページが表示されます。

5. [Basic details] (基本的な詳細) で、[Security group name] (セキュリティグループ名) と [Description] (説明) を入力します。[VPC] で、DB クラスターを作成する先の VPC を選択します。
6. [Inbound rules] (インバウンドルール) で、[Add rule] (ルールを追加) を選択します。
   a. [タイプ] で [カスタム TCP] を選択します。
   b. [Port range] (ポート範囲) で、DB クラスターのために使用するポート値を入力します。
   c. [Source] (ソース) で、セキュリティグループ名を選択するか、DB クラスターにアクセスする IP アドレスの範囲 (CIDR 値) を入力します。[My IP] (マイ IP) を選択すると、ブラウザで検出された IP アドレスから DB クラスターにアクセスできます。
7. IP アドレスや異なるポート範囲を追加する必要がある場合は、[Add rule] (ルールを追加) を選択し、ルールの情報を入力します。
8. (オプション) [Outbound rules] (アウトバウンドルール) で、アウトバウンドトラフィックのルールを追加します。デフォルトではすべてのアウトバウンドトラフィックが許可されます。
9. [セキュリティグループの作成] を選択します。

ここで作成した VPC セキュリティグループは、作成される DB クラスターのセキュリティグループとして使用できます。

> **Note**
>
> デフォルトの VPC を使用する場合、すべての VPC のサブネットにまたがるデフォルトのサブネットグループが作成されます。DB クラスターを作成する場合は、デフォルト VPC を選択し、[DB Subnet Group] (DB サブネットグループ) の [default] (デフォルト) を使用できます。

セットアップ要件を充足したら、Amazon Aurora DB クラスターの作成 の手順に従って、要件とセキュリティグループを使用して DB クラスターを作成できます。特定の DB エンジンを使用する DB クラスターを作成することによる開始方法については、Amazon Aurora の開始方法 を参照してください。

Amazon Aurora の開始方法

このセクションでは、Amazon RDS を使用して Aurora DB クラスターを作成し、接続する方法を示します。

以下の手順は、Aurora の使用を開始するにあたってベーシックを説明するチュートリアルです。以降のセクションでは、各種のエンドポイントや、Aurora クラスターのスケールアップとスケールダウンの方法など、Aurora の高度な概念と手順について説明します。

> **Important**
>
> DB クラスターを作成したり、DB クラスターに接続したりする前に、必ず Amazon Aurora の環境をセットアップする のタスクを完了してください。

トピック

- Aurora MySQL DB クラスターの作成と接続
- Aurora PostgreSQL DB クラスターの作成と接続
- チュートリアル: ウェブサーバーと Amazon Aurora DB クラスターを作成する

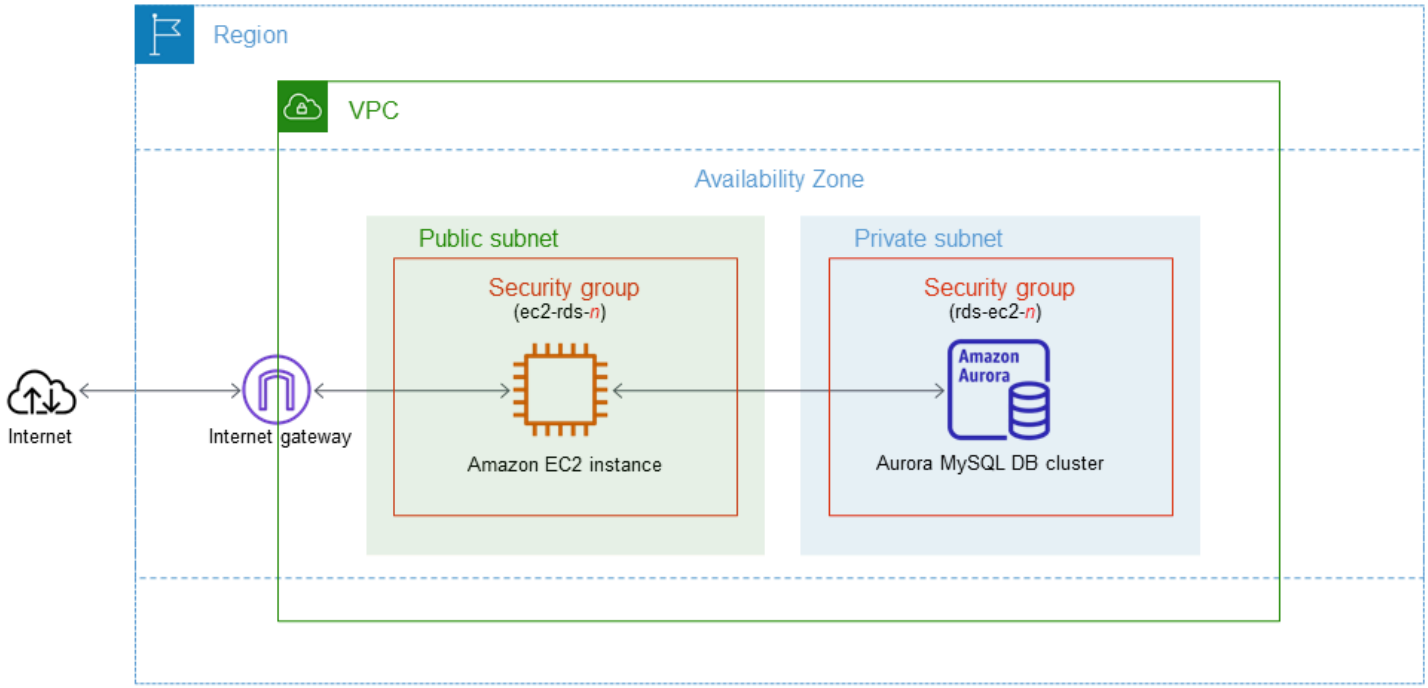
Aurora MySQL DB クラスターの作成と接続

このチュートリアルでは、EC2 インスタンスと Aurora MySQL DB クラスターを作成します。このチュートリアルでは、標準の MySQL クライアントを使用して EC2 インスタンスから DB クラスターにアクセスする方法を説明します。ベストプラクティスとして、このチュートリアルでは、プライベート DB クラスターを仮想プライベートクラウド (VPC) に作成します。ほとんどの場合、EC2 インスタンスなど、同じ VPC 内の他のリソースは DB クラスターにアクセスできますが、VPC 外部のリソースはアクセスできません。

チュートリアルを完了すると、VPC 内の各アベイラビリティーゾーンにパブリックサブネットとプライベートサブネットができます。1 つのアベイラビリティーゾーンで、EC2 インスタンスはパブリックサブネットにあり、DB インスタンスはプライベートサブネットにあります。

> **Important**
>
> AWS アカウントを作成するための料金はかかりません。ただし、このチュートリアルを完了すると、使用する AWS リソースのコストが発生する可能性があります。これらのリソースが不要になった場合は、チュートリアルの完了後に削除できます。

次の図は、チュートリアルが完了した時点の設定を示しています。



このチュートリアルでは、次のいずれかの方法を使用してリソースを作成できます。

1. AWS Management Console を使用する -「ステップ 1: EC2 インスタンスを作成する」と「ステップ 2: Aurora MySQL DB クラスターを作成する」
2. AWS CloudFormation を使用してデータベースインスタンスと EC2 インスタンスを作成する - (オプション) AWS CloudFormation を使用して VPC、EC2 インスタンス、および Aurora MySQL クラスターを作成する

最初の方法では、[簡単作成] を使用して、AWS Management Console でプライベート Aurora MySQL DB クラスターを作成します。DB エンジンタイプ、DB インスタンスサイズ、および DB クラスター識別子のみを指定します。[Easy create (簡易作成)] では、他の設定オプションのデフォルト設定を使用します。

代わりに [標準作成] を使用する場合は、DB クラスターの作成時にさらに多くの設定オプションを指定できます。このようなオプションには、可用性、セキュリティ、バックアップ、メンテナンスの設定があります。パブリック DB クラスターを作成するには、[標準作成] を使用する必要があります。詳細については、the section called “DB クラスターの作成” を参照してください。

トピック

前提条件

開始する前に、以下のセクションのステップを完了してください。

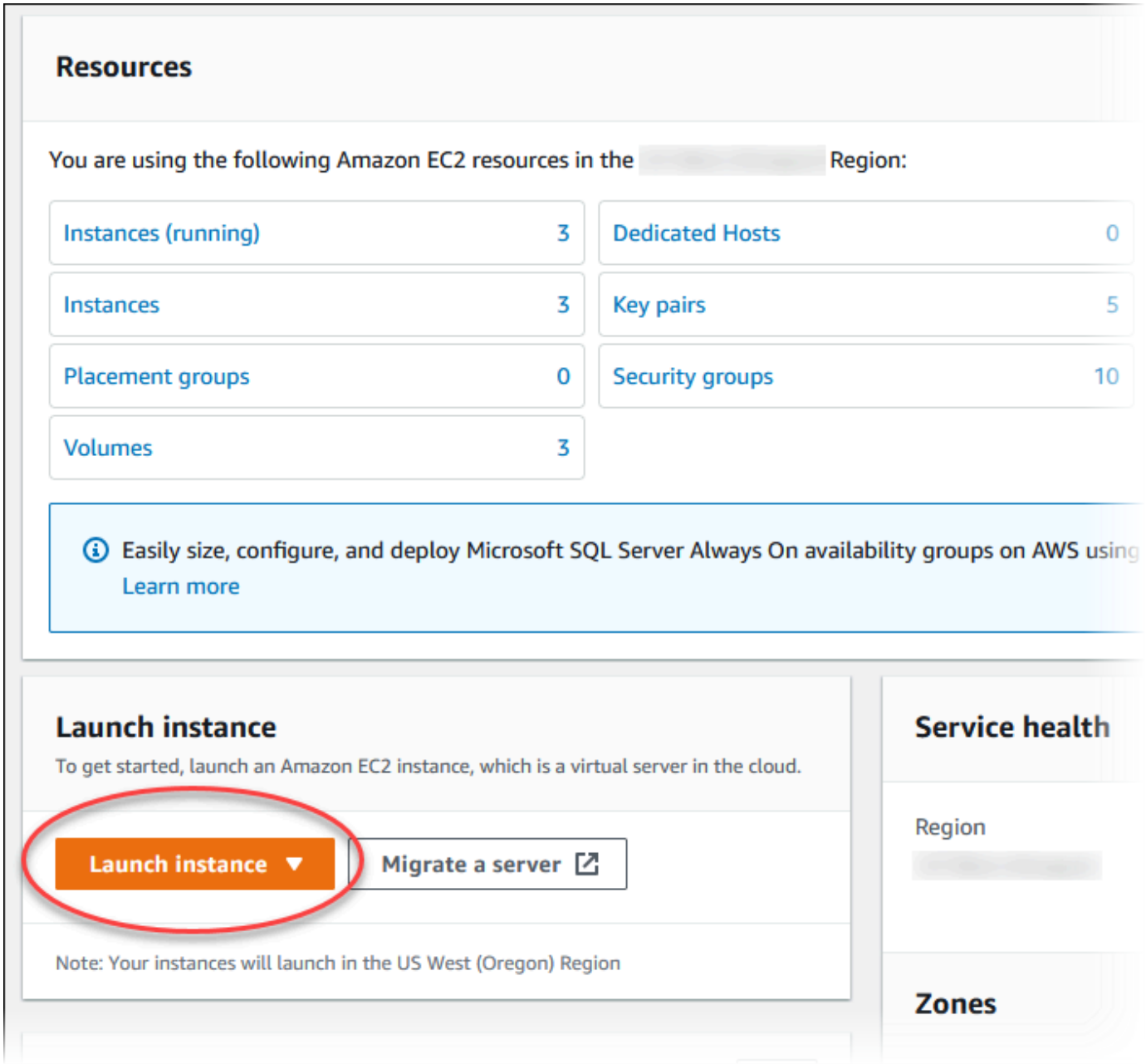
- AWS アカウントへのサインアップ
- 管理アクセスを持つユーザーを作成する

ステップ 1: EC2 インスタンスを作成する

データベースへの接続に使用する Amazon EC2 インスタンスを作成します。

EC2 インスタンスを作成するには

1. AWS Management Console にサインインし、Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. AWS Management Console の右上で、EC2 インスタンスを作成する AWS リージョン を選択します。
3. 次の図に示すように、[EC2 ダッシュボード] を選択し、次に [インスタンスの起動] を選択します。

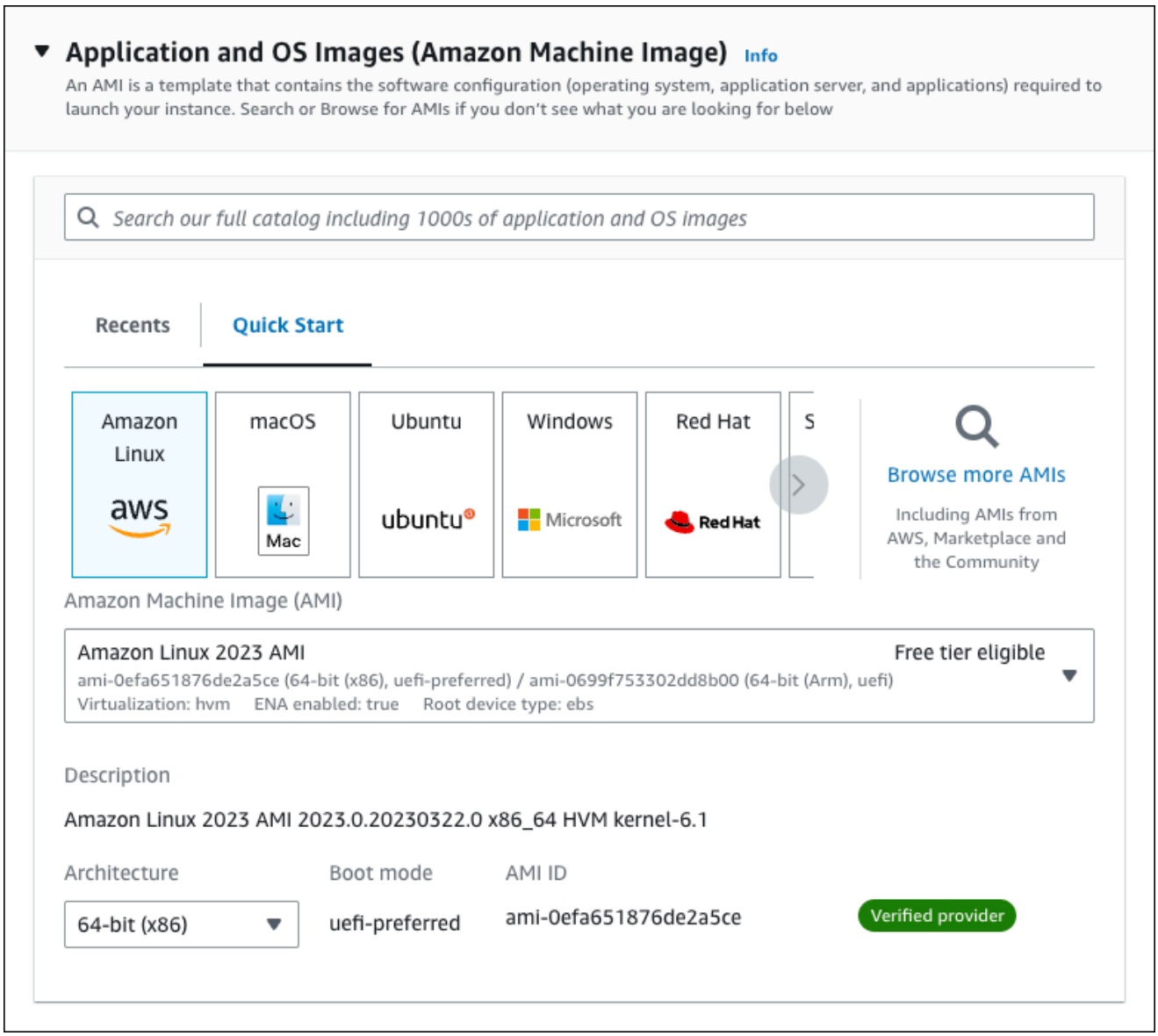


[インスタンスを起動] ページが開きます。

4. [インスタンスを起動] ページで次の設定を選択します。

   a. [Name and tags] (名前とタグ) の、[Name] (名前) で、**`ec2-database-connect`** と入力します。

   b. [アプリケーションおよび OS イメージ (Amazon マシンイメージ)] で、[Amazon Linux] を選択し、[Amazon Linux 2023 AMI] を選択します。他の選択肢は、デフォルトの選択のままにします。



c. [Instance type] (インスタンスタイプ) で [t2.micro] を選択します。

d. [Key pair (login)] (キーペア (ログイン)) で、[Key pair name] (キーペア名) を選択して、既存のキーペアを使用します。Amazon EC2 インスタンスの新しい key pair を作成するには、[Create new key pair] (新しい key pair を作成する) を選択し、[Create key pair] (キーペアを作成する) ウィンドウを使用して作成します。

   キーペアの作成については、Linux インスタンス用 Amazon EC2 ユーザーガイドの「キーペアの作成」を参照してください。

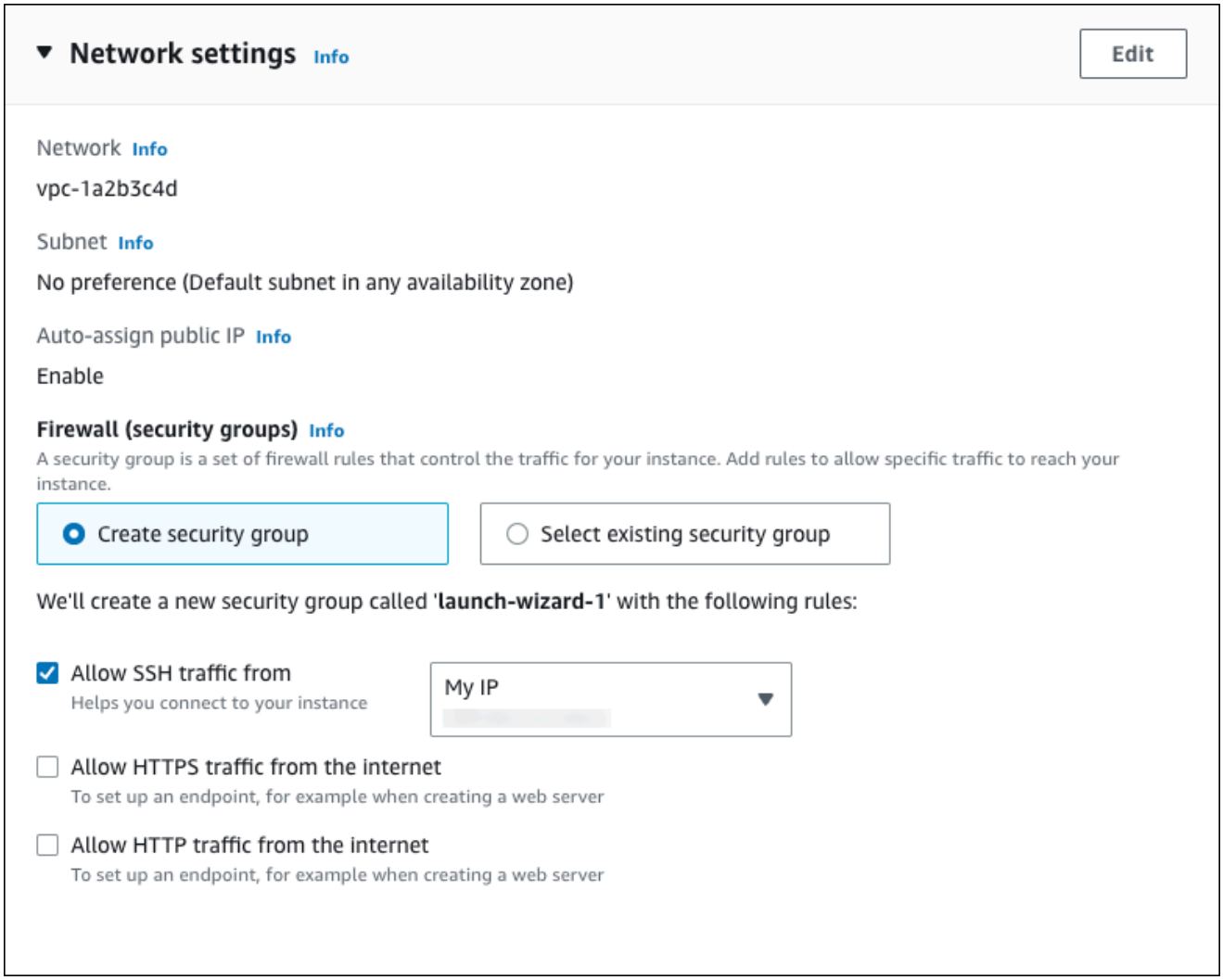
e. ネットワーク設定の [SSH トラフィックを許可] で、EC2 インスタンスへの SSH 接続のソースを選択します。

表示された IP アドレスが SSH 接続に適している場合は、[My IP] (マイ IP)を選択できます。それ以外の場合は、Secure Shell (SSH) を使用して VPC の EC2 インスタンスへの接続に使用する IP アドレスを決定します。パブリック IP アドレスを決定するには、別のブラウザウィンドウまたはタブで、https://checkip.amazonaws.com のサービスを使用できます。IP アドレスの例は 192.0.2.1/32 です。

多くの場合、インターネットサービスプロバイダー (ISP) 経由、またはファイアウォールの内側から静的 IP アドレスなしで接続することがあります。その場合、クライアントコンピュータが使用する IP アドレスの範囲を確認してください。

> **Warning**
>
> SSH アクセスに `0.0.0.0/0` を使用すると、すべての IP アドレスが SSH を使ってパブリック EC2 インスタンスにアクセスできるようになります。この方法は、テスト環境で短時間なら許容できますが、実稼働環境では安全ではありません。実稼働環境では、特定の IP アドレスまたは特定のアドレス範囲にのみ、SSH を使った EC2 インスタンスへのアクセスを承認します。

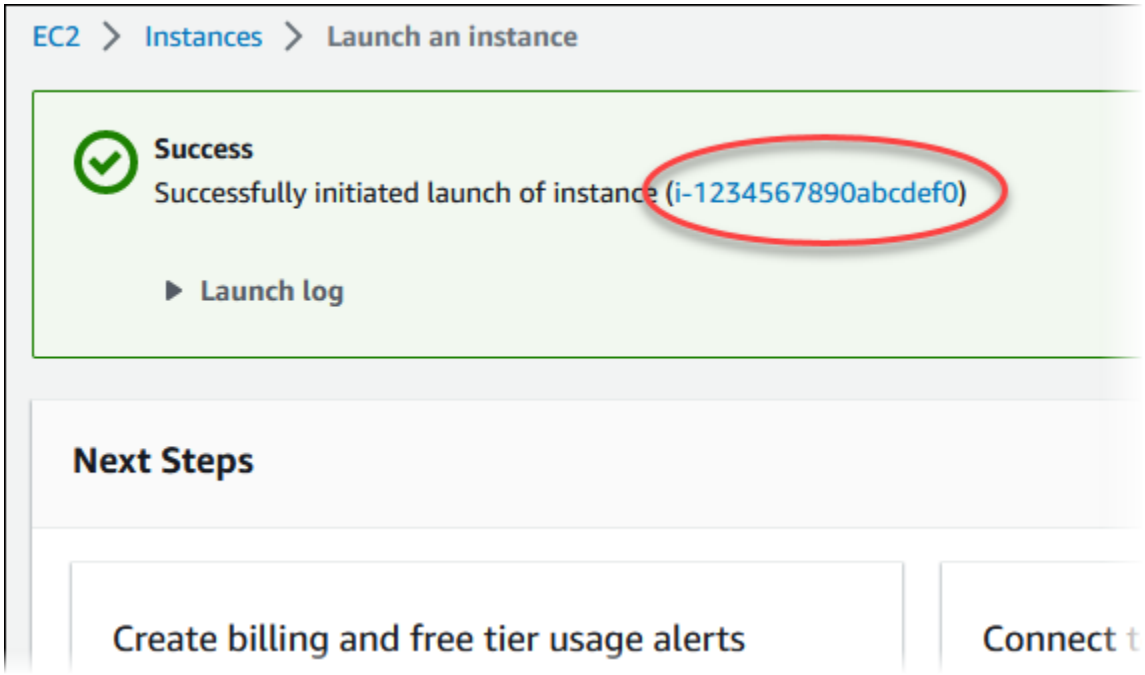
以下のイメージは、[ネットワーク設定] セクションの例を示しています。



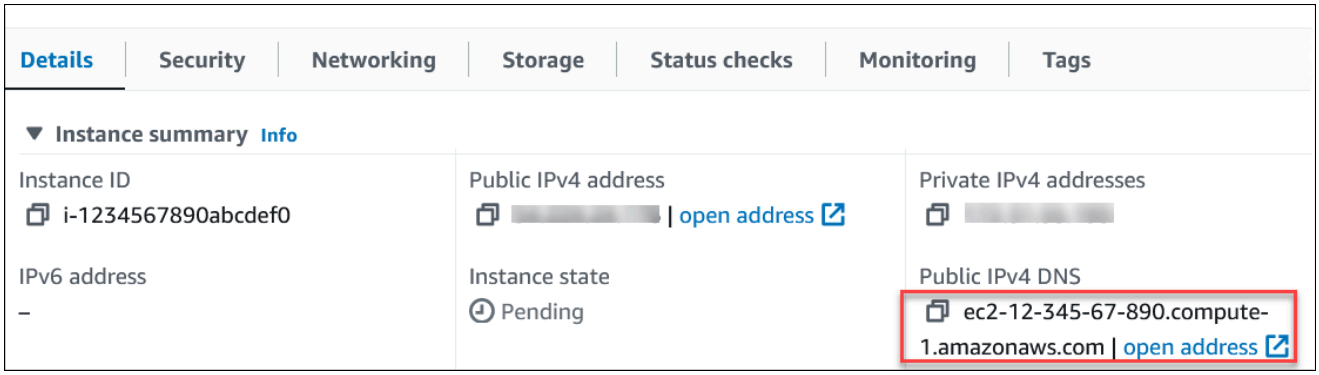
f. 残りのセクションをデフォルト値のままにします。

g. [概要] パネルで、EC2 インスタンス設定の概要を確認し、準備ができたら、[インスタンスの起動] を選択します。

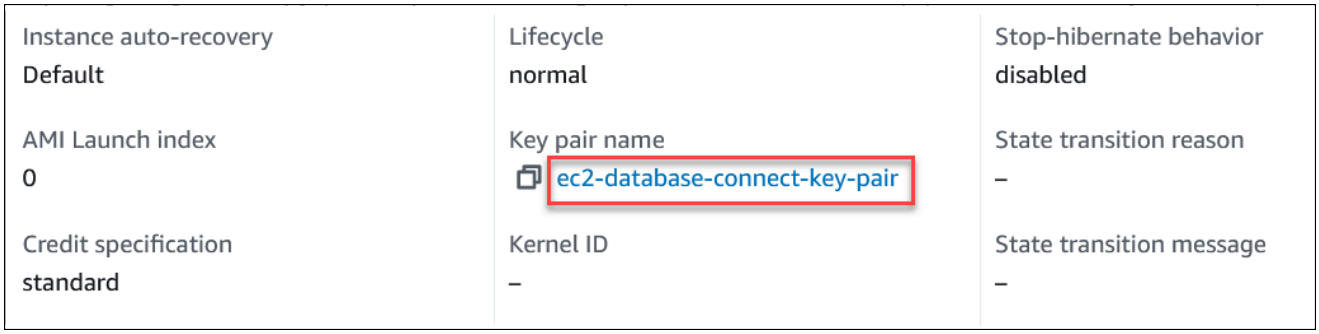
5. [起動ステータス] ページで、新しい EC2 インスタンスの ID (例: `i-1234567890abcdef0`) をメモします。



6. EC2 インスタンス ID を選択して、EC2 インスタンスのリストを開き、EC2 インスタンスを選択します。
7. [詳細] タブで、SSH を使用して接続するときに必要な次の値を書き留めます。
   a. [インスタンスの概要] で、[パブリック IPv4 DNS] の値を書き留めます。



   b. [インスタンスの詳細] で、[キーペア名] の値を書き留めます。



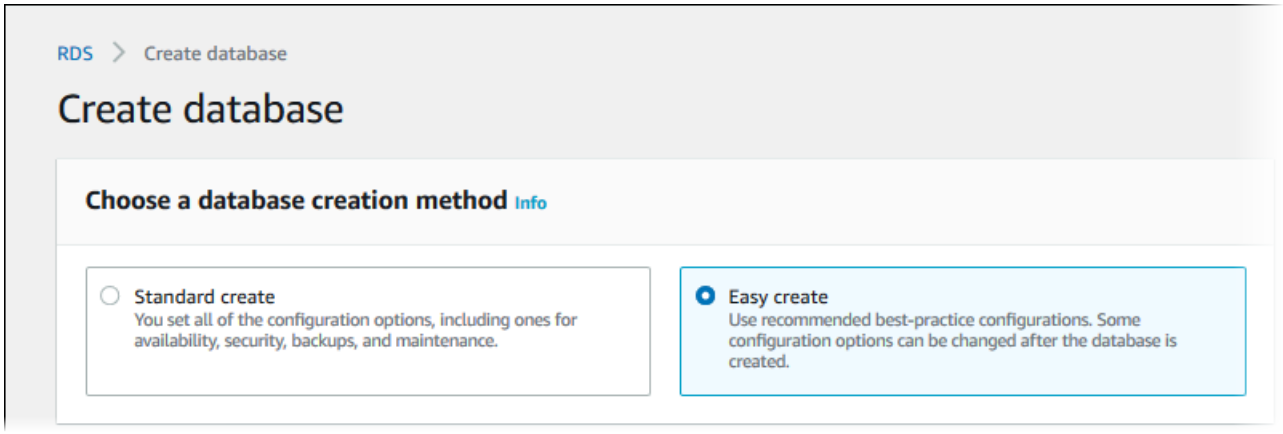
8. EC2 インスタンスの [インスタンス状態] が [実行中] になるまで待ってから、続行します。

ステップ 2: Aurora MySQL DB クラスターを作成する

この例では、[簡易作成] を使用して、db.r6g.large DB インスタンスクラスで Aurora MySQL DB クラスターを作成します。

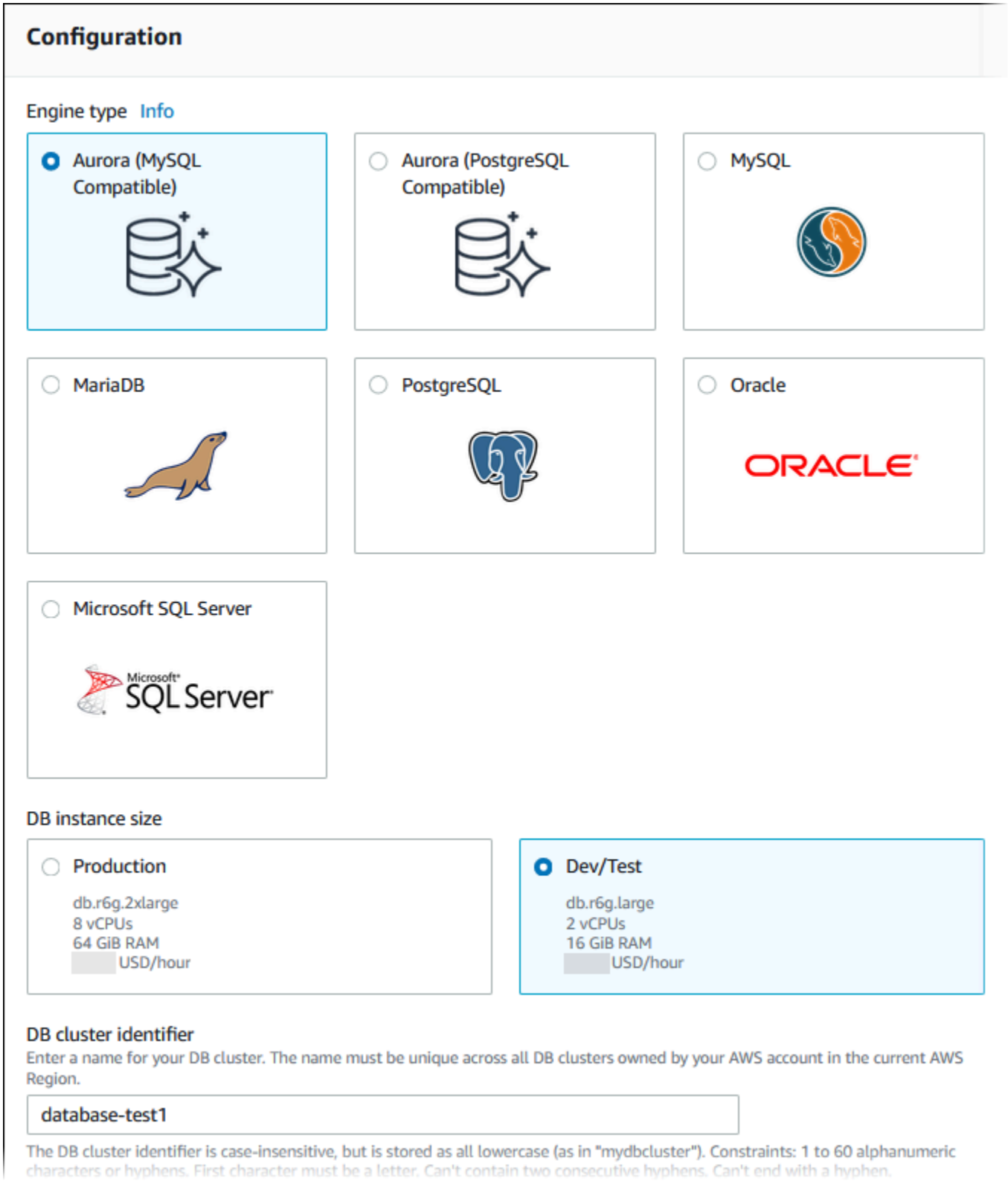
簡易作成で Aurora MySQL DB クラスターを作成するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. Amazon RDS コンソールの右上で、DB クラスターを作成する AWS リージョン を選択します。
3. ナビゲーションペインで、[データベース] を選択します。
4. [Create database (データベースの作成)] を選択し、[Easy create (簡易作成)] が選択されていることを確認します。



5. [設定] で、[エンジンタイプ] として、[Aurora (MySQL 互換)] を選択します。
6. [DB インスタンスサイズ] で、[Dev/Test] を選択します。
7. [DB クラスター識別子] として、**`database-test1`** を入力します。

   [データベースの作成] ページは、次のイメージのようになります。



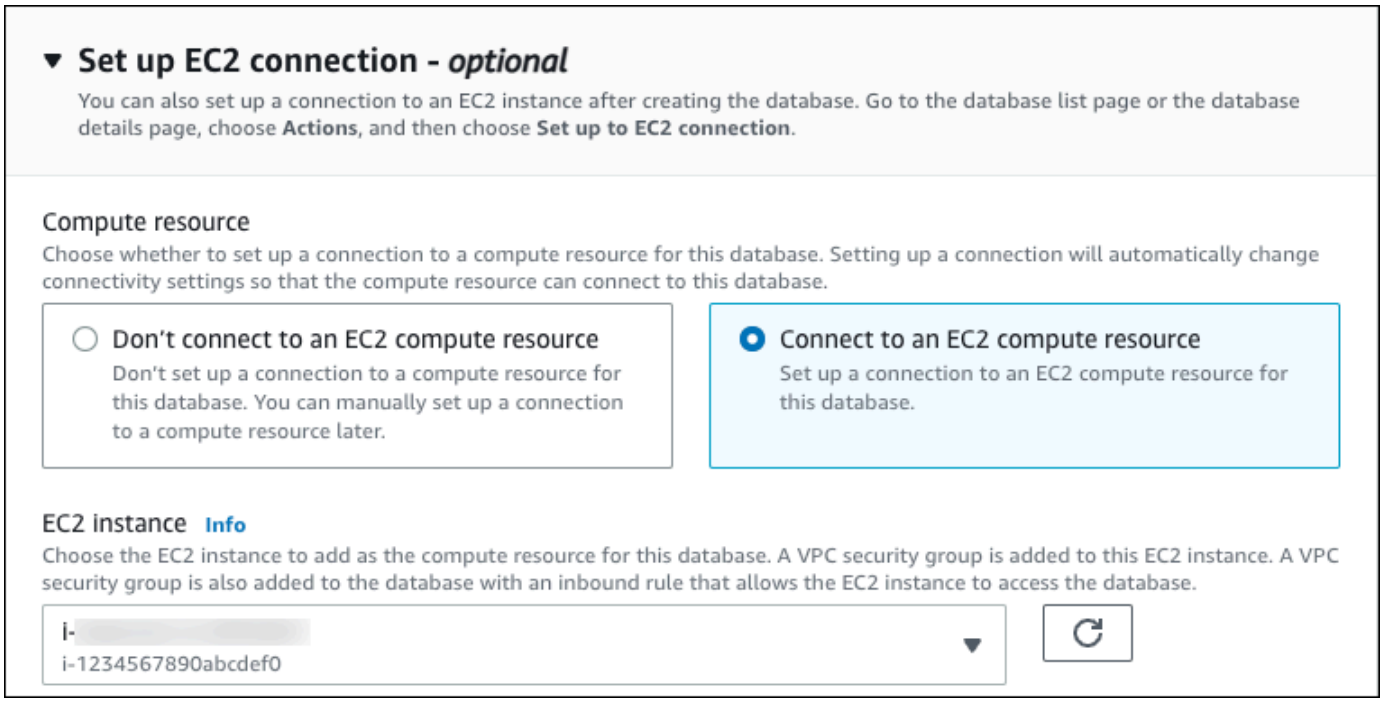
8. [マスターユーザー名] に、マスターユーザーの名前を入力するか、デフォルト名のままにします。

9. DB クラスター用に自動生成されたマスターパスワードを使用するには、[パスワードの自動生成] を選択します。

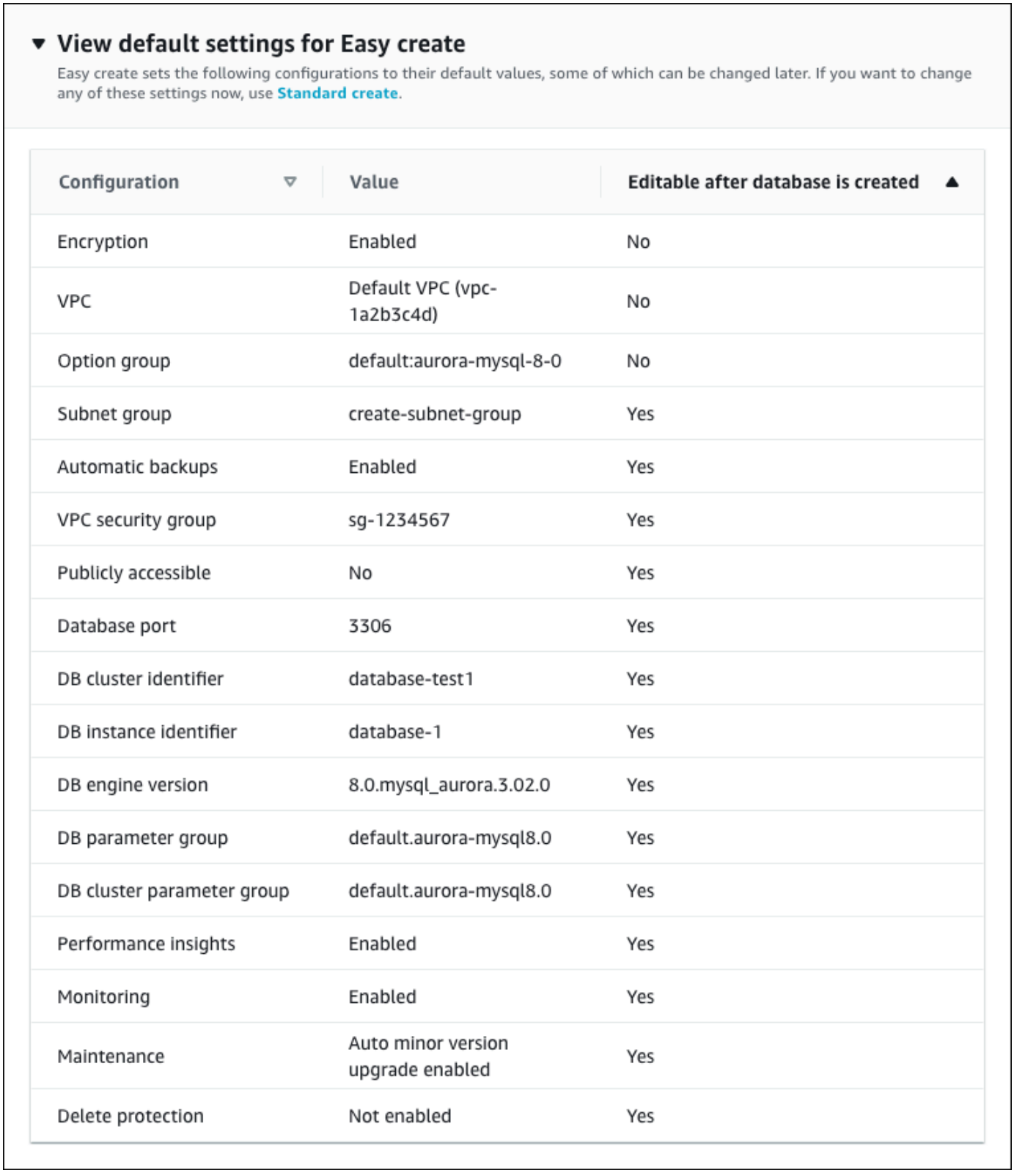
   マスターパスワードを入力するには、[パスワードの自動生成] をオフにして、[マスターパスワード] と [パスワードの確認] に同じパスワードを入力します。

10. 以前に作成した EC2 インスタンスとの接続をセットアップするには、[EC2 接続のセットアップ - オプション] を開きます。

    [EC2 コンピューティングリソースに接続] を選択します。以前に作成した EC2 インスタンスを選択します。



11. [簡易作成のデフォルト設定を表示] を開きます。



View default settings for Easy create

Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use **Standard create**.

Configuration	Value	Editable after database is created
Encryption	Enabled	No
VPC	Default VPC (vpc-1a2b3c4d)	No
Option group	default:aurora-mysql-8-0	No
Subnet group	create-subnet-group	Yes
Automatic backups	Enabled	Yes
VPC security group	sg-1234567	Yes
Publicly accessible	No	Yes
Database port	3306	Yes
DB cluster identifier	database-test1	Yes
DB instance identifier	database-1	Yes
DB engine version	8.0.mysql_aurora.3.02.0	Yes
DB parameter group	default.aurora-mysql8.0	Yes
DB cluster parameter group	default.aurora-mysql8.0	Yes
Performance insights	Enabled	Yes
Monitoring	Enabled	Yes
Maintenance	Auto minor version upgrade enabled	Yes
Delete protection	Not enabled	Yes

[Easy Create (簡易作成)] で使用されるデフォルト設定を調べることができます。[データベース作成後に編集可能] 列には、データベース作成後に変更できるオプションが表示されます。

- その列の設定に [いいえ] があり、別の設定が必要な場合は、[標準作成] を使用して DB クラスターを作成できます。
- その列の設定に [はい] があり、別の設定が必要な場合は、[標準作成] を使用して DB クラスターを作成するか、DB クラスターの作成後に設定を変更できます。

12. [データベースの作成] を選択します。

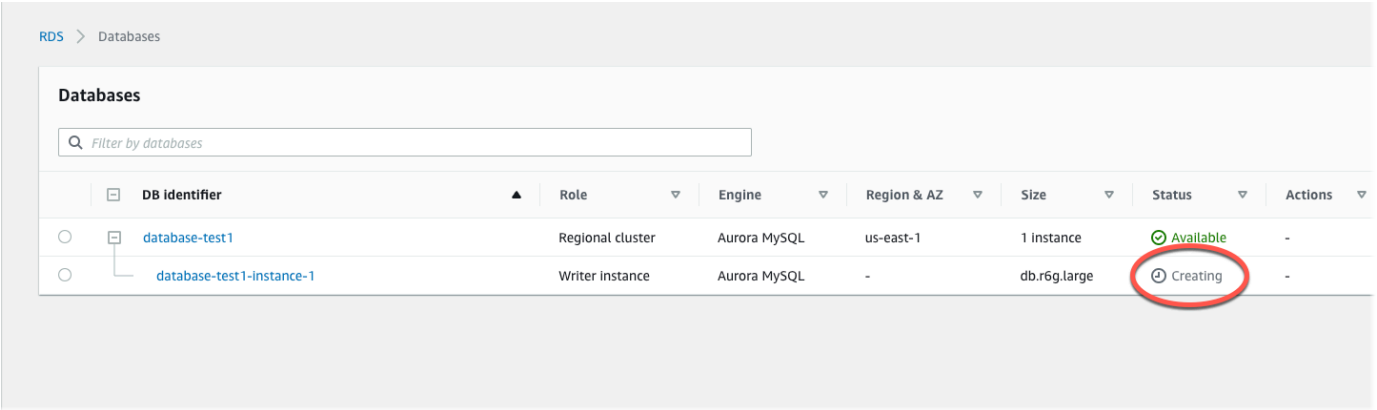
    DB クラスターのマスターユーザー名とパスワードを表示するには、[認証情報の詳細の表示] を選択します。

    表示されるユーザー名とパスワードを使用して、マスターユーザーとして DB クラスターに接続できます。

    > **Important**
    >
    > マスターユーザーのパスワードを再度表示することはできません。記録していない場合は、変更する必要がある場合があります。
    > DB クラスターが有効になった後にマスターユーザーのパスワードを変更する必要がある場合は、DB クラスターを変更することができます。DB クラスターの変更の詳細については、「Amazon Aurora DB クラスターの変更」を参照してください。

13. [データベース] リストで、新しい Aurora MySQL DB クラスターの名前を選択し、詳細を表示します。

    DB クラスターが使用できるようになるまで、ライターインスタンスのステータスは [作成中] のままです。



ライターインスタンスのステータスが [使用可能] に変わったら、DB クラスターに接続できます。DB インスタンスクラスとストレージの合計によっては、新しい DB クラスターを使用できるようになるまで最長 20 分かかることがあります。

(オプション) AWS CloudFormation を使用して VPC、EC2 インスタンス、および Aurora MySQL クラスターを作成する

コンソールを使用して VPC、EC2 インスタンス、および Aurora MySQL DB クラスターを作成する代わりに、インフラストラクチャをコードとして扱うことで、AWS CloudFormation を使用して AWS リソースをプロビジョニングできます。AWS リソースをより小さく管理しやすい単位に整理するには、AWS CloudFormation のネストされたスタック機能を使用できます。詳細については、「AWS CloudFormation コンソールでのスタックの作成」と「ネストされたスタックの操作」を参照してください。

> **Important**
>
> AWS CloudFormation は無料ですが、CloudFormation が作成するリソースは実動のものです。これらのリソースを終了するまで、標準使用料が発生します。合計料金はごくわずかです。料金を最小限に抑える方法については、「AWS 無料利用枠」を参照してください。

AWS CloudFormation コンソールを使用してリソースを作成するには、以下のステップを実行します。

- ステップ 1: CloudFormation テンプレートをダウンロードする
- ステップ 2: CloudFormation を使用してリソースを設定する

CloudFormation テンプレートをダウンロードする

CloudFormation テンプレートは、スタックで作成するリソースの設定情報を含む JSON または YAML テキストファイルです。このテンプレートは、Aurora クラスターとともに VPC と踏み台ホストも作成します。

テンプレートファイルをダウンロードするには、次のリンク Aurora MySQL CloudFormation テンプレートを開きます。

この Github ページで、[Download raw file] ボタンをクリックしてテンプレートの YAML ファイルを保存します。

CloudFormation を使用してリソースを設定する

> **Note**
>
> このプロセスを開始する前に、AWS アカウント に EC2 インスタンスのキーペアがあることを確認してください。詳細については、「Amazon EC2 キーペアおよび Linux インスタンス」を参照してください。

AWS CloudFormation テンプレートを使用する場合は、適切なパラメータを選択して、リソースが正しく作成されていることを確認する必要があります。以下のステップを実行します。

1. AWS Management Console にサインインし、AWS CloudFormation コンソール (https://console.aws.amazon.com/cloudformation) を開きます。
2. [Create Stack] (スタックの作成) を選択します。
3. [テンプレートの指定] セクションで、[コンピュータからテンプレートファイルをアップロード] を選択し、[次へ] をクリックします。
4. [スタックの詳細を指定] ページで、以下のパラメータを設定します。
   a. [スタック名] を AurMySQLTestStack に設定します。
   b. [パラメータ] で、2 つのアベイラビリティーゾーンを選択して [アベイラビリティーゾーン] を設定します。
   c. [Linux 踏み台ホスト設定] で、[キー名] に EC2 インスタンスにログインするキーペアを選択します。
   d. [Linux 踏み台ホスト設定] で、[許可された IP 範囲] を IP アドレスに設定します。Secure Shell (SSH) を使用して VPC 内の EC2 インスタンスに接続するには、https://checkip.amazonaws.com のサービスを使用してパブリック IP アドレスを確認します。IP アドレスの例は 192.0.2.1/32 です。

> **Warning**
>
> SSH アクセスに `0.0.0.0/0` を使用すると、すべての IP アドレスが SSH を使ってパブリック EC2 インスタンスにアクセスできるようになります。この方法は、テスト環境で短時間なら許容できますが、実稼働環境では安全ではありません。実稼働環境で

は、特定の IP アドレスまたは特定のアドレス範囲にのみ、SSH を使った EC2 インスタンスへのアクセスを承認します。

e. [データベース全般設定] で、[データベースインスタンスクラス] を db.r6g.large に設定します。

f. [データベース名] を **database-test1** に設定します。

g. [データベースマスターユーザー名] には、PDB のマスターユーザー名を入力します。

h. このチュートリアルでは、[Secrets Manager で DB マスターユーザーパスワードを管理] を `false` に設定します。

i. [データベースパスワード] には、任意のパスワードを設定します。このパスワードは、チュートリアルの他の手順のために覚えておいてください。

j. [マルチ AZ 配置]を `false` に設定します。

k. その他の設定は、すべてデフォルト値のままにします。[次へ] をクリックして続行します。

5. [スタックオプションの設定] ページでは、すべてのデフォルトオプションをそのまま使用します。[次へ] をクリックして続行します。

6. [スタックの確認] ページで、データベースと Linux 踏み台ホストのオプションを確認した後、[送信] をクリックします。

スタックの作成プロセスが完了したら、データベースへの接続に必要な情報を書き留めるために、BastionStack と AMSNS という名前のスタックを表示します。詳細については、「AWS Management Console での AWS CloudFormation スタックデータとリソースの表示」を参照してください。

ステップ 3: Aurora MySQL DB クラスターに接続する

標準の SQL クライアントアプリケーションを使用して、DB クラスターに接続できます。この例では、mysql コマンドラインクライアントを使用して、Aurora MySQL DB クラスターに接続します。

Aurora MySQL DB クラスターに接続するには

1. DB クラスターのライターインスタンスのエンドポイント (DNS 名) とポート番号を見つけます。

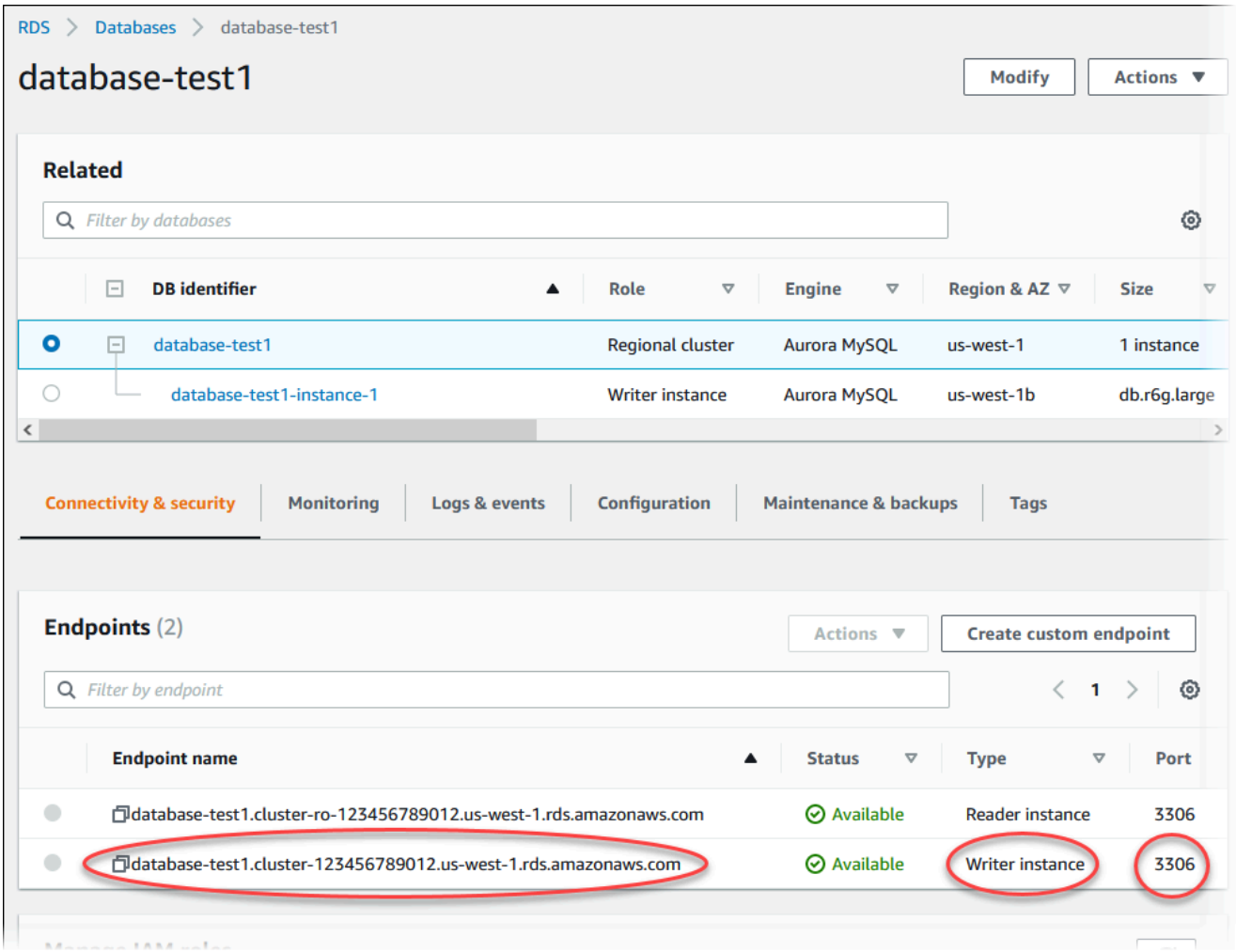
   a. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。

   b. Amazon RDS コンソールの右上で、DB クラスターの AWS リージョン を選択します。

   c. ナビゲーションペインで、データベースを選択します。

d. Aurora MySQL DB クラスターの名前を選択して、詳細を表示します。

e. [接続とセキュリティ] タブで、ライターインスタンスのエンドポイントをコピーします。また、ポート番号を書き留めます。DB クラスターに接続するには、エンドポイントとポート番号の両方が必要です。



2. Linux インスタンスに関する Amazon EC2 ユーザーガイドの「Linux インスタンスに接続する」のステップに従って、先ほど作成した EC2 インスタンスに接続します。

   SSH を使用して EC2 インスタンスに接続することをお勧めします。SSH クライアントユーティリティが Windows、Linux、または Mac にインストールされている場合は、次のコマンド形式でインスタンスに接続できます。

   ```
   ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
   ```

   例えば、`ec2-database-connect-key-pair.pem` が Linux の `/dir1` に保存されていて、EC2 インスタンスのパブリック IPv4 DNS が

ec2-12-345-678-90.compute-1.amazonaws.com であるとします。その場合、SSH コマンドは次のようになります。

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

3. EC2 インスタンスのソフトウェアを更新して、最新のバグ修正とセキュリティ更新を入手します。そのためには、次のコマンドを使用します。

   > **Note**
   >
   > -y オプションを指定すると、確認メッセージを表示せずに更新をインストールします。インストール前に更新を確認するには、このオプションを省略します。

   ```
   sudo dnf update -y
   ```

4. Amazon Linux 2023 で MariaDB から mysql コマンドラインクライアントをインストールするには、次のコマンドを実行します。

   ```
   sudo dnf install mariadb105
   ```

5. Aurora MySQL DB クラスターに接続します。例えば、次のコマンドを入力します。このアクションにより、MySQL クライアントを使用して、Aurora MySQL DB クラスターに接続できます。

   endpoint をライターインスタンスのエンドポイントに置き換え、*admin* を、使用したマスターユーザー名に置き換えます。パスワードの入力を求められたときに使用したマスターパスワードを入力します。

   ```
   mysql -h endpoint -P 3306 -u admin -p
   ```

   ユーザーのパスワードを入力すると、次のような出力が表示されます。

   ```
   Welcome to the MariaDB monitor.  Commands end with ; or \g.
   Your MySQL connection id is 217
   Server version: 8.0.23 Source distribution

   Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.
   ```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>
```

Aurora MySQL DB クラスターへの接続の詳細については、「Amazon Aurora MySQL DB クラスターへの接続」を参照してください。DB クラスターに接続できない場合は、Amazon RDS DB インスタンスに接続できない を参照してください。

セキュリティのためには、暗号化された接続を使用することがベストプラクティスです。クライアントとサーバーが同じ VPC にあり、ネットワークが信頼されている場合に限り、暗号化されていない MySQL 接続を使用します。暗号化された接続の使用については、「SSL を使用した Aurora MySQL への接続」を参照してください。

6. SQL コマンドを実行する。

   例えば、次の SQL コマンドは、現在の日付と時刻を表示します。

```
SELECT CURRENT_TIMESTAMP;
```

ステップ 4: EC2 インスタンスと DB クラスターを削除する

作成したサンプル EC2 インスタンスと DB クラスターに接続して、探索したら、料金がこれ以上発生しないように、それらを削除します。

AWS CloudFormation を使用してリソースを作成した場合は、このステップをスキップして次のステップに進みます。

EC2 インスタンスを削除するには

1. AWS Management Console にサインインし、Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. ナビゲーションペインで、[インスタンス] を選択します。
3. EC2 インスタンスを選択し、[インスタンスの状態]、[インスタンスの終了] の順に選択します。
4. 確認を求めるメッセージが表示されたら、[Terminate (終了)] を選択します。

EC2 インスタンスの削除の詳細については、「Amazon EC2 Linux インスタンス用ユーザーガイド」の「インスタンスの終了」を参照してください。

DB クラスターを削除するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. [データベース] を選択後、DB クラスターに関連付けられた DB インスタンスを選択します。
3. [アクション] で、[削除] を選択します。
4. [最終スナップショットを作成] をオフにします。
5. 確認を完了し、[削除] を選択します。

DB クラスターに関連付けられている DB インスタンスがすべて削除されると、DB クラスターは自動的に削除されます。

(オプション) CloudFormation で作成された EC2 インスタンスと DB クラスターを削除する

AWS CloudFormation を使用してリソースを作成した場合、作成したサンプル EC2 インスタンスと DB クラスターに接続して、探索した後に CloudFormation を削除することで、料金がこれ以上発生しないようにします。

CloudFormation リソースを削除するには

1. AWS CloudFormation コンソールを開きます。
2. CloudFormation コンソールの [スタック] ページで、ルートスタック (VPCStack BastionStack、BastionStack、または AMSNS という名前のスタック) を選択します。
3. [削除] を選択します。
4. 確認を求めるメッセージが表示されたら、[削除] を選択します。

CloudFormation のスタックの削除に関する詳細は、AWS CloudFormation ユーザーガイドの「AWS CloudFormation コンソールでスタックの削除」を参照してください。

(オプション) DB クラスターを Lambda 関数に接続する

Aurora MySQL DB クラスターを Lambda サーバーレスコンピューティングリソースに接続することもできます。Lambda 関数を使用すると、インフラストラクチャをプロビジョニングしたり管理したりせずにコードを実行できます。Lambda 関数を使用すると、1 日に 数十件のイベントから 1 秒間に数百件のイベントまで、あらゆる規模のコード実行リクエストに自動的に応答することもできま

す。詳細については、「Lambda 関数と Aurora DB クラスターを自動的に接続する」を参照してください。

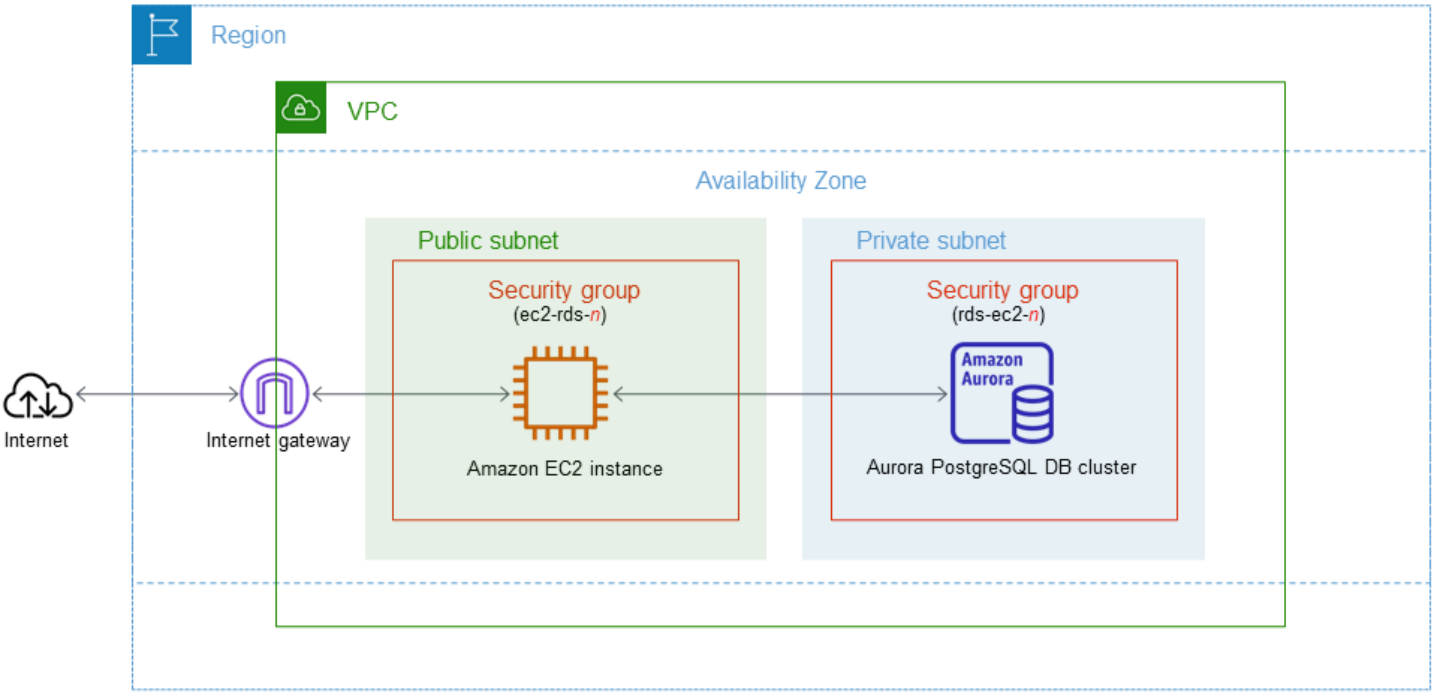
Aurora PostgreSQL DB クラスターの作成と接続

このチュートリアルでは、EC2 インスタンスと Aurora PostgreSQL DB クラスターを作成します。このチュートリアルでは、標準の PostgreSQL クライアントを使用して EC2 インスタンスから DB クラスターにアクセスする方法を説明します。ベストプラクティスとして、このチュートリアルでは、プライベート DB クラスターを仮想プライベートクラウド (VPC) に作成します。ほとんどの場合、EC2 インスタンスなど、同じ VPC 内の他のリソースは DB クラスターにアクセスできますが、VPC 外部のリソースはアクセスできません。

チュートリアルを完了すると、VPC 内の各アベイラビリティーゾーンにパブリックサブネットとプライベートサブネットができます。1 つのアベイラビリティーゾーンで、EC2 インスタンスはパブリックサブネットにあり、DB インスタンスはプライベートサブネットにあります。

> **Important**
>
> AWS アカウントを作成するための料金はかかりません。ただし、このチュートリアルを完了すると、使用する AWS リソースのコストが発生する可能性があります。これらのリソースが不要になった場合は、チュートリアルの完了後に削除できます。

次の図は、チュートリアルが完了した時点の設定を示しています。



このチュートリアルでは、次のいずれかの方法を使用してリソースを作成できます。

1. AWS Management Console を使用する -「ステップ 1: EC2 インスタンスを作成する」と「ステップ 2: Aurora PostgreSQL DB クラスターを作成する」
2. AWS CloudFormation を使用してデータベースインスタンスと EC2 インスタンスを作成する - (オプション) を使用して VPC、EC2 インスタンス、および Aurora PostgreSQL クラスターを作成する AWS CloudFormation

最初の方法では、[簡単作成] を使用して、AWS Management Console でプライベート PostgreSQL DB クラスターを作成します。DB エンジンタイプ、DB インスタンスサイズ、および DB クラスター識別子のみを指定します。[Easy create (簡易作成)] では、他の設定オプションのデフォルト設定を使用します。

代わりに [標準作成] を使用する場合は、DB クラスターの作成時にさらに多くの設定オプションを指定できます。このようなオプションには、可用性、セキュリティ、バックアップ、メンテナンスの設定があります。パブリック DB クラスターを作成するには、[標準作成] を使用する必要があります。詳細については、the section called “DB クラスターの作成” を参照してください。

トピック

- 前提条件
- ステップ 1: EC2 インスタンスを作成する

- ステップ 2: Aurora PostgreSQL DB クラスターを作成する
- (オプション) を使用して VPC、EC2 インスタンス、および Aurora PostgreSQL クラスターを作成する AWS CloudFormation
- ステップ 3: Aurora PostgreSQL DB クラスターに接続する
- ステップ 4: EC2 インスタンスと DB クラスターを削除する
- (オプション) CloudFormation で作成された EC2 インスタンスと DB クラスターを削除する
- (オプション) DB クラスターを Lambda 関数に接続する

前提条件

開始する前に、以下のセクションのステップを完了してください。

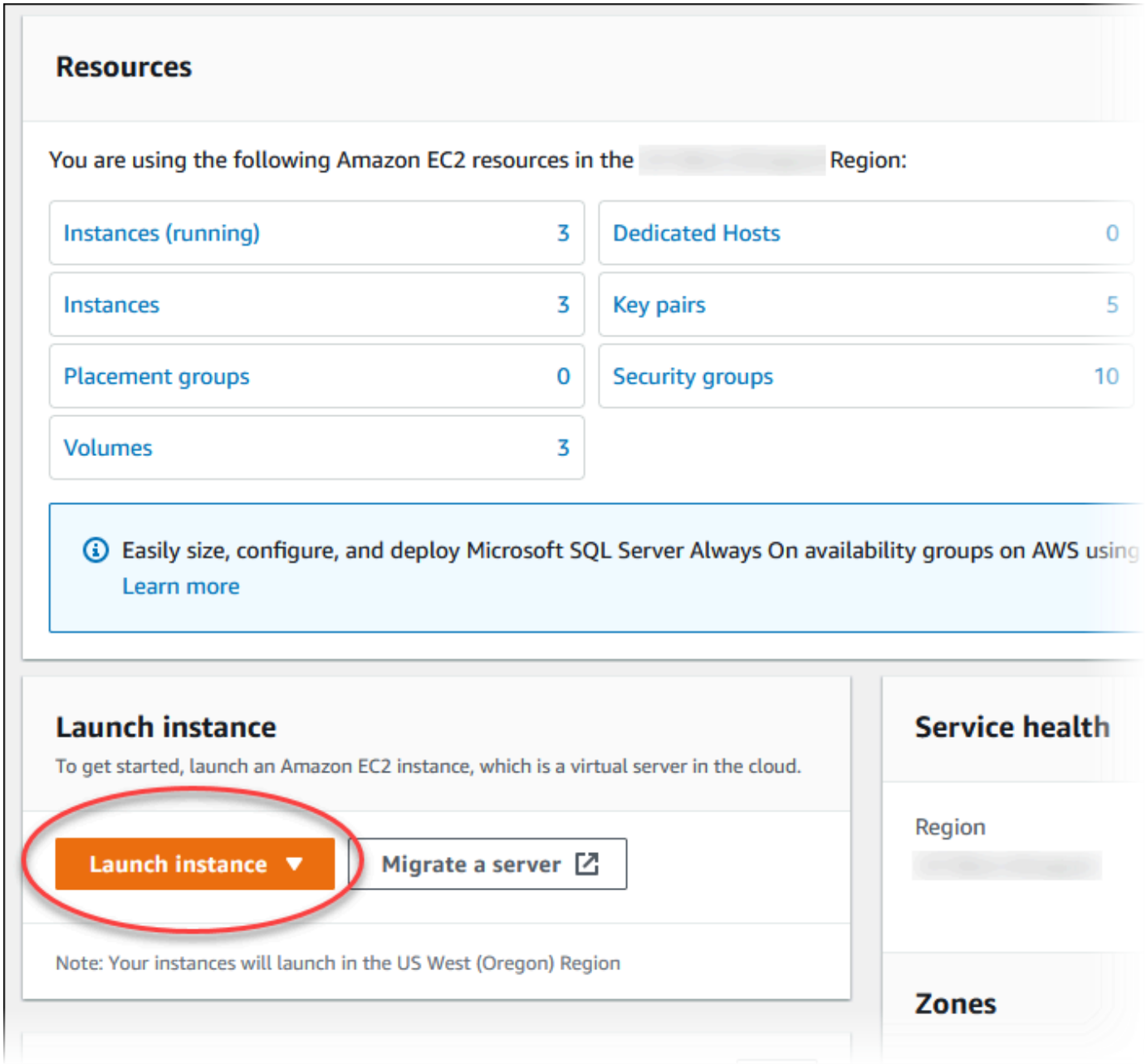
- AWS アカウントへのサインアップ
- 管理アクセスを持つユーザーを作成する

ステップ 1: EC2 インスタンスを作成する

データベースへの接続に使用する Amazon EC2 インスタンスを作成します。

EC2 インスタンスを作成するには

1. AWS Management Console にサインインし、Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. AWS Management Console の右上で、EC2 インスタンスを作成する AWS リージョン を選択します。
3. 次の図に示すように、[EC2 ダッシュボード] を選択し、次に [インスタンスの起動] を選択します。

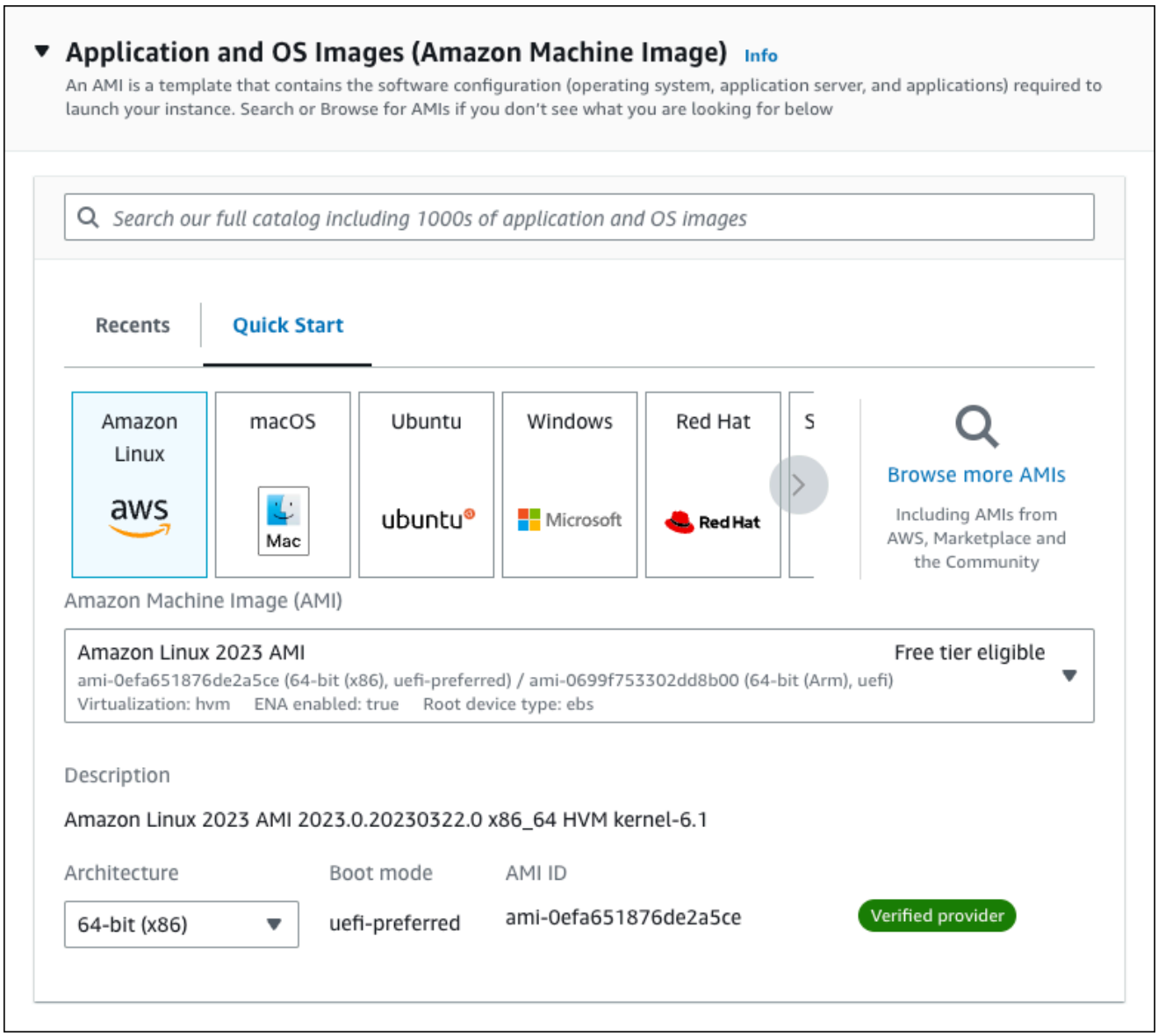


[インスタンスを起動] ページが開きます。

4. [インスタンスを起動] ページで次の設定を選択します。

   a. [Name and tags] (名前とタグ) の、[Name] (名前) で、**`ec2-database-connect`** と入力します。

   b. [アプリケーションおよび OS イメージ (Amazon マシンイメージ)] で、[Amazon Linux] を選択し、[Amazon Linux 2023 AMI] を選択します。他の選択肢は、デフォルトの選択のままにします。



c. [Instance type] (インスタンスタイプ) で [t2.micro] を選択します。

d. [Key pair (login)] (キーペア (ログイン)) で、[Key pair name] (キーペア名) を選択して、既存のキーペアを使用します。Amazon EC2 インスタンスの新しい key pair を作成するには、[Create new key pair] (新しい key pair を作成する) を選択し、[Create key pair] (キーペアを作成する) ウィンドウを使用して作成します。

   キーペアの作成については、Linux インスタンス用 Amazon EC2 ユーザーガイドの「キーペアの作成」を参照してください。

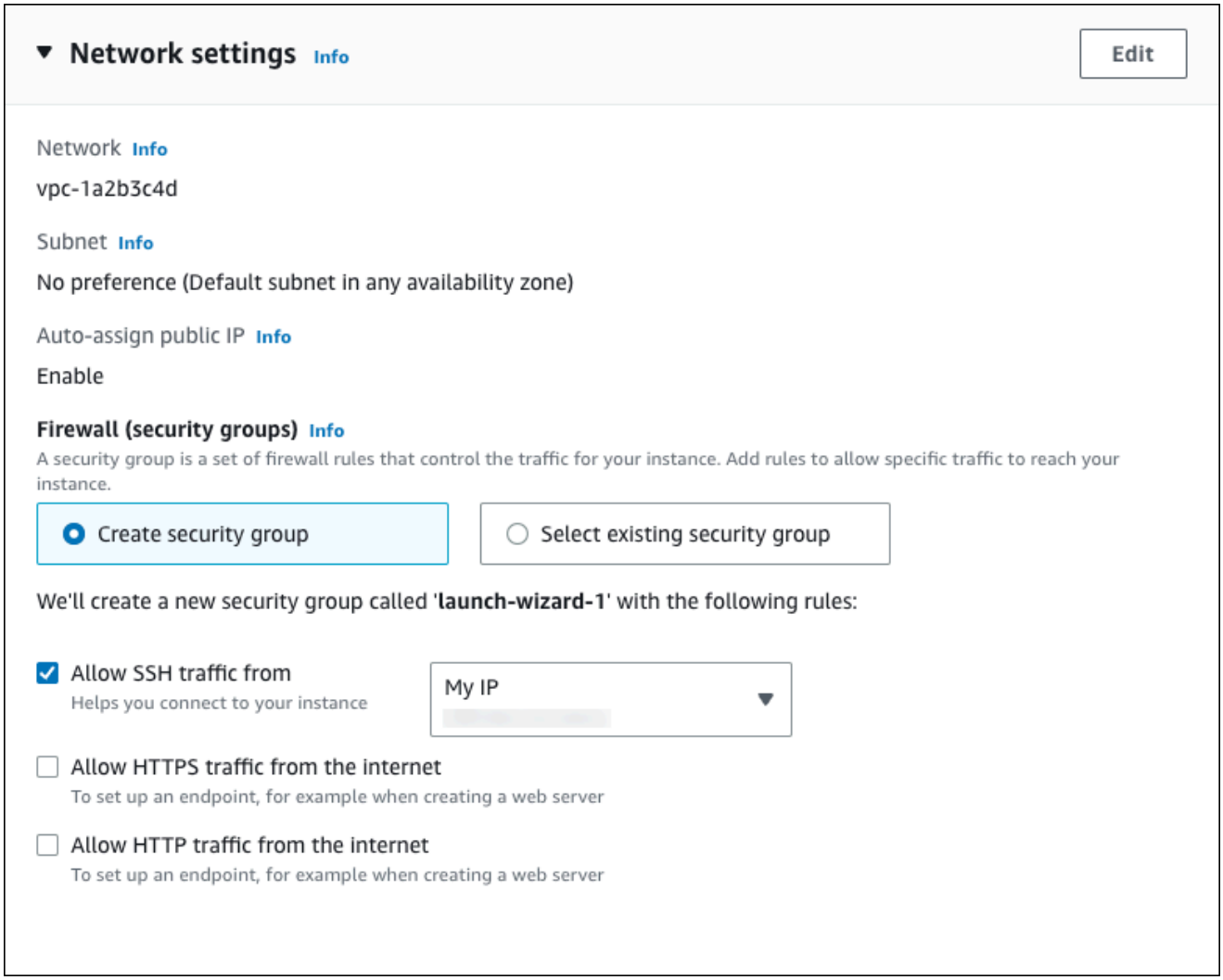
e. ネットワーク設定の [SSH トラフィックを許可] で、EC2 インスタンスへの SSH 接続のソースを選択します。

表示された IP アドレスが SSH 接続に適している場合は、[My IP] (マイ IP)を選択できます。それ以外の場合は、Secure Shell (SSH) を使用して VPC の EC2 インスタンスへの接続に使用する IP アドレスを決定します。パブリック IP アドレスを決定するには、別のブラウザウィンドウまたはタブで、https://checkip.amazonaws.com のサービスを使用できます。IP アドレスの例は 192.0.2.1/32 です。

多くの場合、インターネットサービスプロバイダー (ISP) 経由、またはファイアウォールの内側から静的 IP アドレスなしで接続することがあります。その場合、クライアントコンピュータが使用する IP アドレスの範囲を確認してください。

> **Warning**
>
> SSH アクセスに `0.0.0.0/0` を使用すると、すべての IP アドレスが SSH を使ってパブリック EC2 インスタンスにアクセスできるようになります。この方法は、テスト環境で短時間なら許容できますが、実稼働環境では安全ではありません。実稼働環境では、特定の IP アドレスまたは特定のアドレス範囲にのみ、SSH を使った EC2 インスタンスへのアクセスを承認します。

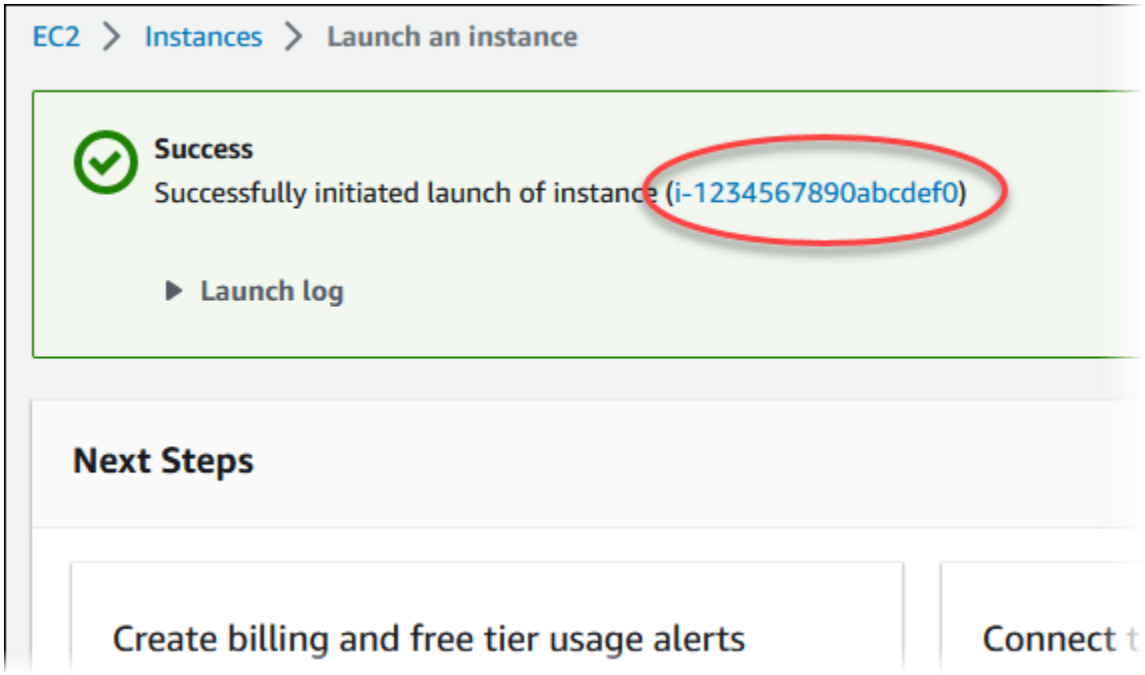
以下のイメージは、[ネットワーク設定] セクションの例を示しています。



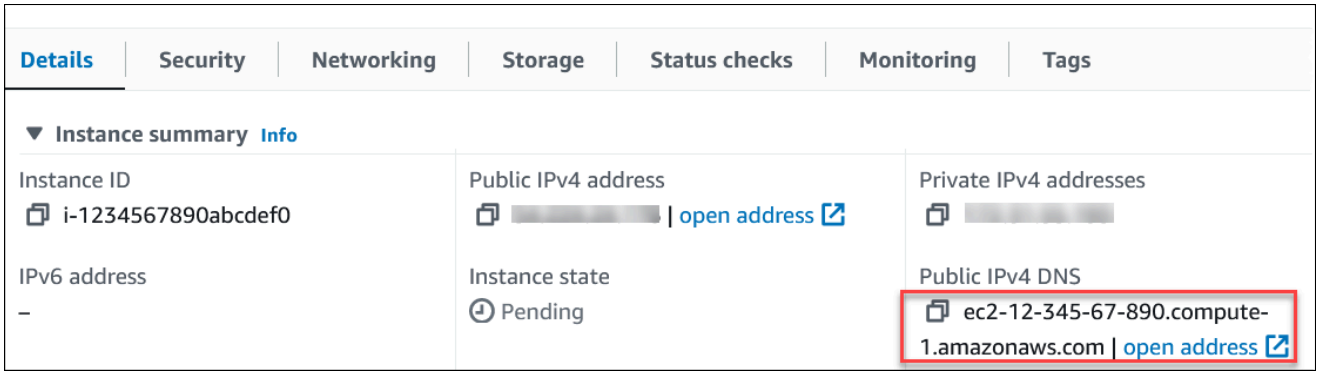
f. 残りのセクションをデフォルト値のままにします。

g. [概要] パネルで、EC2 インスタンス設定の概要を確認し、準備ができたら、[インスタンスの起動] を選択します。

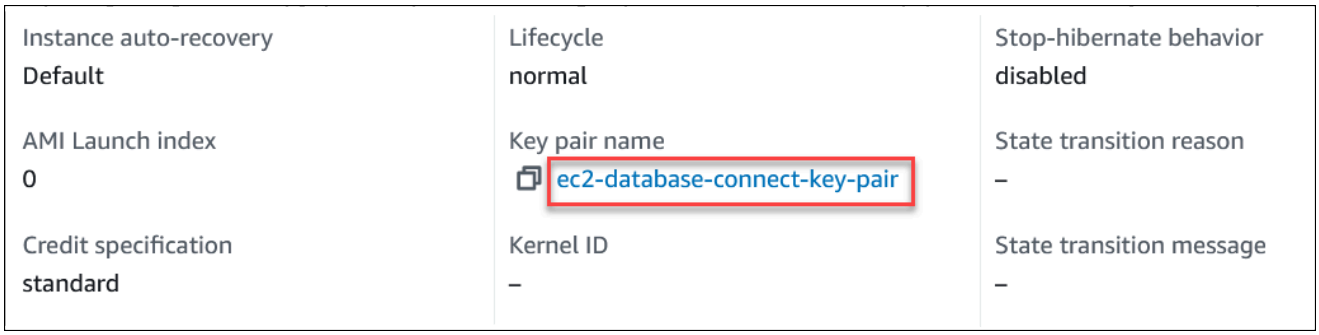
5. [起動ステータス] ページで、新しい EC2 インスタンスの ID (例: `i-1234567890abcdef0`) をメモします。



6. EC2 インスタンス ID を選択して、EC2 インスタンスのリストを開き、EC2 インスタンスを選択します。
7. [詳細] タブで、SSH を使用して接続するときに必要な次の値を書き留めます。
   a. [インスタンスの概要] で、[パブリック IPv4 DNS] の値を書き留めます。



   b. [インスタンスの詳細] で、[キーペア名] の値を書き留めます。



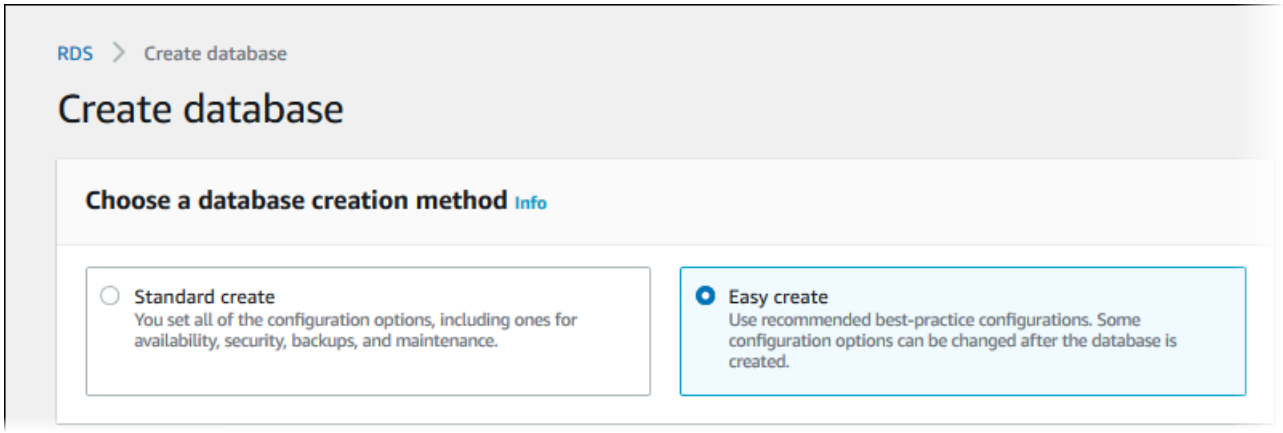
8. EC2 インスタンスの [インスタンス状態] が [実行中] になるまで待ってから、続行します。

ステップ 2: Aurora PostgreSQL DB クラスターを作成する

この例では、[簡易作成] を使用して、db.t4g.large DB インスタンスクラスで Aurora PostgreSQL DB クラスターを作成します。

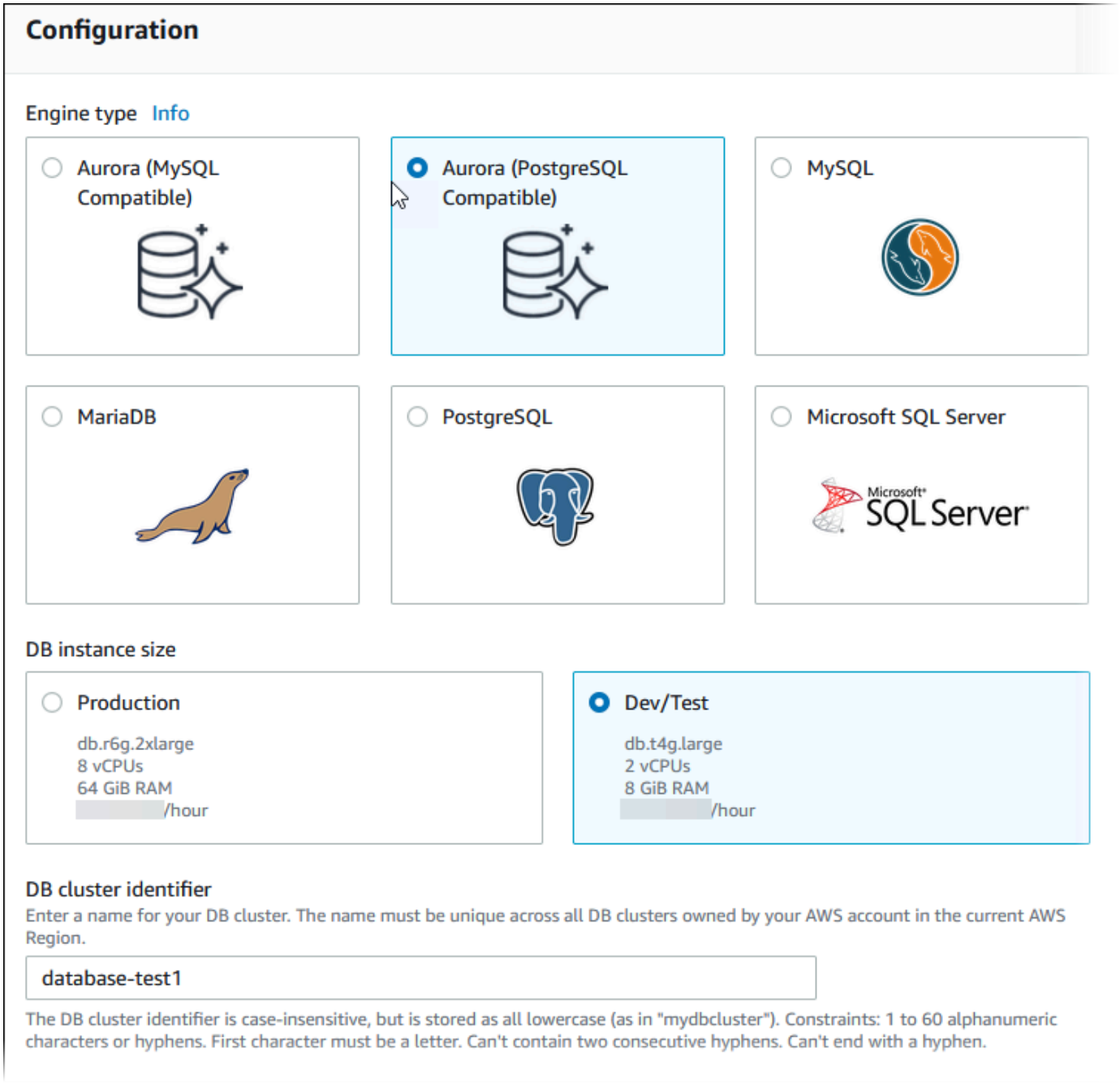
簡易作成で Aurora PostgreSQL DB クラスターを作成するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. Amazon RDS コンソールの右上で、DB クラスターを作成する AWS リージョン を選択します。
3. ナビゲーションペインで、データベースを選択します。
4. [データベースの作成] を選択し、[簡易作成] が選択されていることを確認します。



5. [設定] で、[エンジンタイプ] として、[Aurora (PostgreSQL 互換)] を選択します。
6. [DB インスタンスサイズ] で、[Dev/Test] を選択します。
7. [DB クラスター識別子] として、**`database-test1`** を入力します。

   [データベースの作成] ページは、次のイメージのようになります。

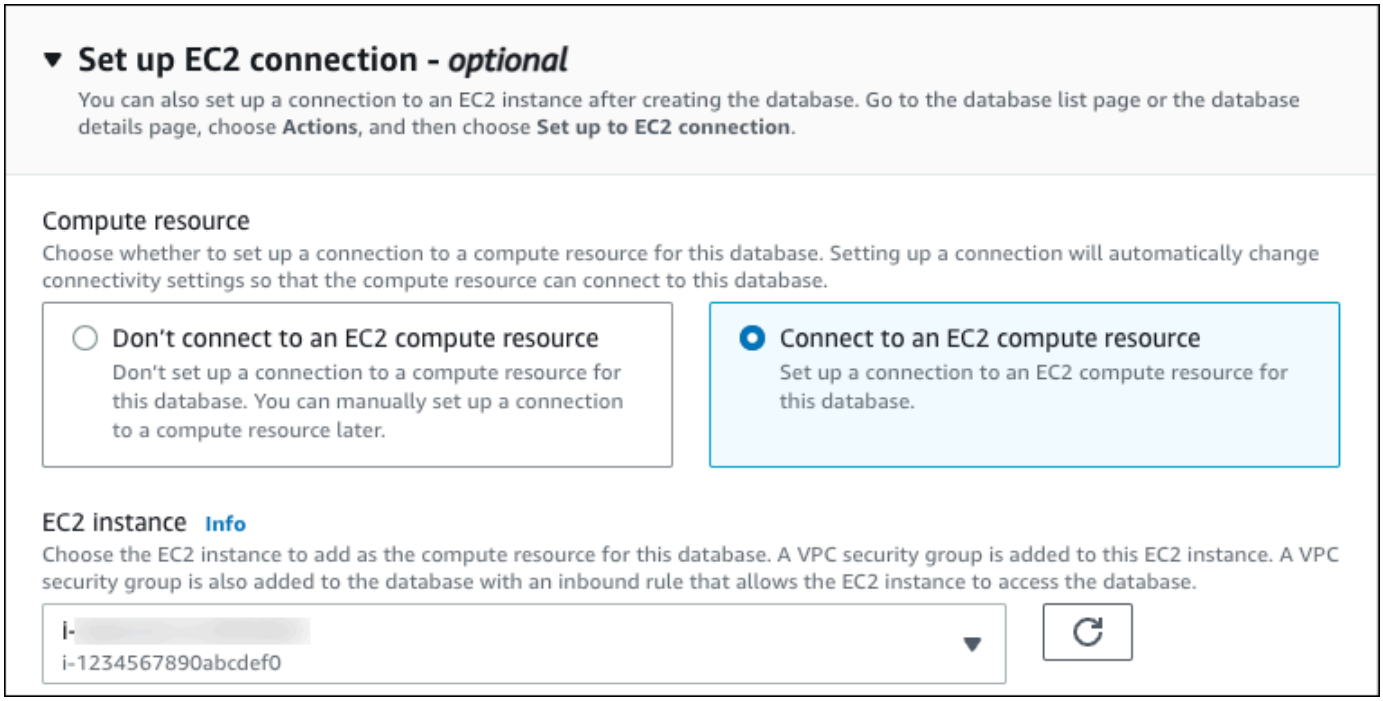


8. [マスターユーザー名] に、ユーザーの名前を入力するか、デフォルト名 (**`postgres`**) のままにします。
9. DB クラスター用に自動生成されたマスターパスワードを使用するには、[パスワードの自動生成] を選択します。

   マスターパスワードを入力するには、[パスワードの自動生成] チェックボックスをオフにして、[マスターパスワード] と [パスワードの確認] に同じパスワードを入力します。

10. 以前に作成した EC2 インスタンスとの接続をセットアップするには、[EC2 接続のセットアップ - オプション] を開きます。

    [EC2 コンピューティングリソースに接続] を選択します。以前に作成した EC2 インスタンスを選択します。



11. [簡易作成のデフォルト設定を表示] を開きます。

View default settings for Easy create

Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use **Standard create**.

Configuration	Value	Editable after database is created
Encryption	Enabled	No
VPC	Default VPC (vpc-1a2b3c4d)	No
Option group	default:aurora-postgresql-13	No
Subnet group	create-subnet-group	Yes
Automatic backups	Enabled	Yes
VPC security group	sg-1234567	Yes
Publicly accessible	No	Yes
Database port	5432	Yes
DB cluster identifier	database-test1	Yes
DB instance identifier	database-1	Yes
DB engine version	13.6	Yes
DB parameter group	default.aurora-postgresql13	Yes
DB cluster parameter group	default.aurora-postgresql13	Yes
Performance insights	Enabled	Yes
Monitoring	Enabled	Yes
Maintenance	Auto minor version upgrade enabled	Yes
Delete protection	Not enabled	Yes

[Easy Create (簡易作成)] で使用されるデフォルト設定を調べることができます。[データベース作成後に編集可能] 列には、データベース作成後に変更できるオプションが表示されます。

- その列の設定に [いいえ] があり、別の設定が必要な場合は、[標準作成] を使用して DB クラスターを作成できます。
- その列の設定に [はい] があり、別の設定が必要な場合は、[標準作成] を使用して DB クラスターを作成するか、DB クラスターの作成後に設定を変更できます。

12. [データベースの作成] を選択します。

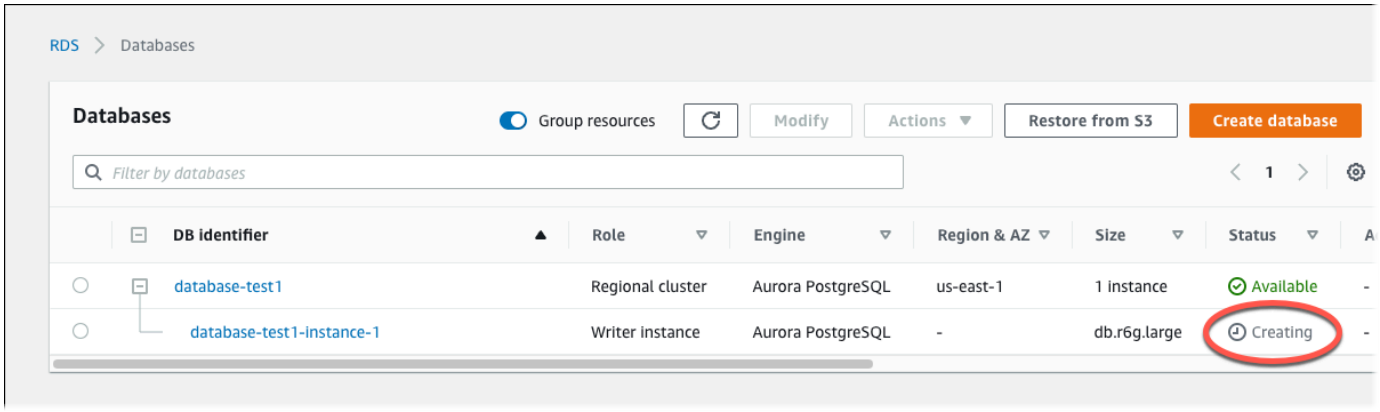
    DB クラスターのマスターユーザー名とパスワードを表示するには、[認証情報の詳細の表示] を選択します。

    表示されるユーザー名とパスワードを使用して、マスターユーザーとして DB クラスターに接続できます。

    > **Important**
    >
    > マスターユーザーのパスワードを再度表示することはできません。記録していない場合は、変更する必要がある場合があります。
    > DB クラスターが有効になった後にマスターユーザーのパスワードを変更する必要がある場合は、DB クラスターを変更することができます。DB クラスターの変更の詳細については、「Amazon Aurora DB クラスターの変更」を参照してください。

13. [データベース] リストで、新しい Aurora PostgreSQL DB クラスターの名前を選択し、詳細を表示します。

    DB クラスターが使用できるようになるまで、ライターインスタンスのステータスは [作成中] のままです。



ライターインスタンスのステータスが [使用可能] に変わったら、DB クラスターに接続できます。DB インスタンスクラスとストレージの合計によっては、新しい DB クラスターを使用できるようになるまで最長 20 分かかることがあります。

(オプション) を使用して VPC、EC2 インスタンス、および Aurora PostgreSQL クラスターを作成する AWS CloudFormation

コンソールを使用して VPC、EC2 インスタンス、および PostgreSQL DB クラスターを作成する代わりに、インフラストラクチャをコードとして扱うことで、AWS CloudFormation を使用して AWS リソースをプロビジョニングできます。AWS リソースをより小さく管理しやすい単位に整理するには、AWS CloudFormation のネストされたスタック機能を使用できます。詳細については、「AWS CloudFormation コンソールでのスタックの作成」と「ネストされたスタックの操作」を参照してください。

> **Important**
>
> AWS CloudFormation は無料ですが、CloudFormation が作成するリソースは実動のものです。これらのリソースを終了するまで、標準使用料が発生します。合計料金はごくわずかです。料金を最小限に抑える方法については、「AWS 無料利用枠」を参照してください。

AWS CloudFormation コンソールを使用してリソースを作成するには、以下のステップを実行します。

- ステップ 1: CloudFormation テンプレートをダウンロードする
- ステップ 2: CloudFormation を使用してリソースを設定する

CloudFormation テンプレートをダウンロードする

CloudFormation テンプレートは、スタックで作成するリソースの設定情報を含む JSON または YAML テキストファイルです。このテンプレートは、Aurora クラスターとともに VPC と踏み台ホストも作成します。

テンプレートファイルをダウンロードするには、次のリンク PostgreSQL CloudFormation テンプレートを開きます。

この Github ページで、[Download raw file] ボタンをクリックしてテンプレートの YAML ファイルを保存します。

CloudFormation を使用してリソースを設定する

> **Note**
>
> このプロセスを開始する前に、AWS アカウント に EC2 インスタンスのキーペアがあることを確認してください。詳細については、「Amazon EC2 キーペアおよび Linux インスタンス」を参照してください。

AWS CloudFormation テンプレートを使用する場合は、適切なパラメータを選択して、リソースが正しく作成されていることを確認する必要があります。以下のステップを実行します。

1. AWS Management Console にサインインし、AWS CloudFormation コンソール (https://console.aws.amazon.com/cloudformation) を開きます。
2. [Create Stack] (スタックの作成) を選択します。
3. [テンプレートの指定] セクションで、[コンピュータからテンプレートファイルをアップロード] を選択し、[次へ] をクリックします。
4. [スタックの詳細を指定] ページで、以下のパラメータを設定します。
   a. [スタック名] を AurPostgreSQLTestStack に設定します。
   b. [パラメータ] で、2 つのアベイラビリティーゾーンを選択して [アベイラビリティーゾーン] を設定します。
   c. [Linux 踏み台ホスト設定] で、[キー名] に EC2 インスタンスにログインするキーペアを選択します。
   d. [Linux 踏み台ホスト設定] で、[許可された IP 範囲] を IP アドレスに設定します。Secure Shell (SSH) を使用して VPC 内の EC2 インスタンスに接続するには、https://checkip.amazonaws.com のサービスを使用してパブリック IP アドレスを確認します。IP アドレスの例は 192.0.2.1/32 です。

> **Warning**
>
> SSH アクセスに `0.0.0.0/0` を使用すると、すべての IP アドレスが SSH を使ってパブリック EC2 インスタンスにアクセスできるようになります。この方法は、テスト環境で短時間なら許容できますが、実稼働環境では安全ではありません。実稼働環境で

は、特定の IP アドレスまたは特定のアドレス範囲にのみ、SSH を使った EC2 インスタンスへのアクセスを承認します。

e. [データベースの全般設定] で、[データベースインスタンスクラス] を db.t4g.large に設定します。

f. [データベース名] を **database-test1** に設定します。

g. [データベースマスターユーザー名] には、PDB のマスターユーザー名を入力します。

h. このチュートリアルでは、[Secrets Manager で DB マスターユーザーパスワードを管理] を `false` に設定します。

i. [データベースパスワード] には、任意のパスワードを設定します。このパスワードは、チュートリアルの他の手順のために覚えておいてください。

j. [マルチ AZ 配置]を `false` に設定します。

k. その他の設定は、すべてデフォルト値のままにします。[次へ] をクリックして続行します。

5. [スタックオプションの設定] ページでは、すべてのデフォルトオプションをそのまま使用します。[次へ] をクリックして続行します。

6. [スタックの確認] ページで、データベースと Linux 踏み台ホストのオプションを確認した後、[送信] をクリックします。

スタック作成プロセスが完了したら、BastionStack と APGNS という名前のスタックを表示して、データベースへの接続に必要な情報をメモします。詳細については、「AWS CloudFormation スタックデータとリソースを AWS Management Console に表示」を参照してください。

ステップ 3: Aurora PostgreSQL DB クラスターに接続する

標準の PostgreSQL クライアントアプリケーションを使用して、DB クラスターに接続できます。この例では、psql コマンドラインクライアントを使用して、Aurora PostgreSQL DB クラスターに接続します。

Aurora PostgreSQL DB クラスターに接続するには

1. DB クラスターのライターインスタンスのエンドポイント (DNS 名) とポート番号を見つけます。

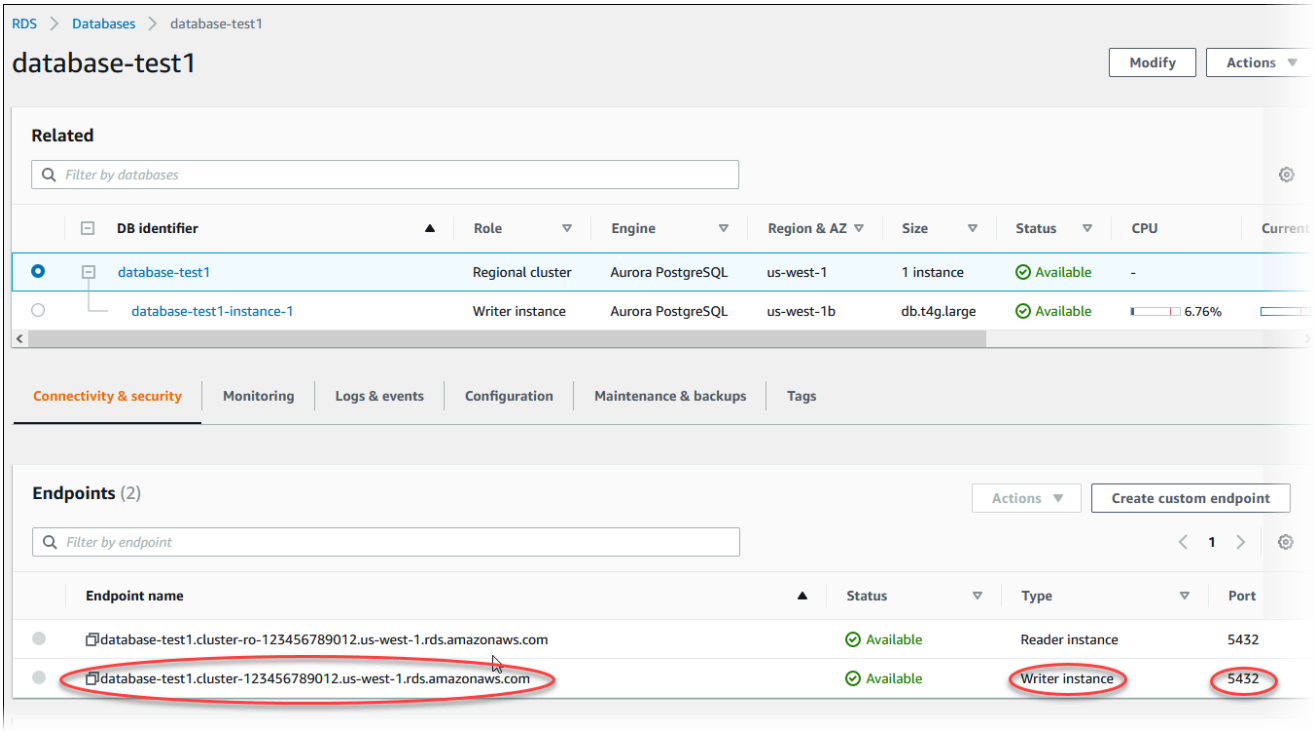
   a. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。

   b. Amazon RDS コンソールの右上で、DB クラスターの AWS リージョン を選択します。

c. ナビゲーションペインで、データベースを選択します。

d. Aurora PostgreSQL DB クラスター名を選択して、詳細を表示します。

e. [接続とセキュリティ] タブで、ライターインスタンスのエンドポイントをコピーします。また、ポート番号を書き留めます。DB クラスターに接続するには、エンドポイントとポート番号の両方が必要です。



2. Linux インスタンスに関する Amazon EC2 ユーザーガイドの「Linux インスタンスに接続する」のステップに従って、先ほど作成した EC2 インスタンスに接続します。

   SSH を使用して EC2 インスタンスに接続することをお勧めします。SSH クライアントユーティリティが Windows、Linux、または Mac にインストールされている場合は、次のコマンド形式でインスタンスに接続できます。

   ```
   ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
   ```

   例えば、`ec2-database-connect-key-pair.pem` が Linux の `/dir1` に保存されていて、EC2 インスタンスのパブリック IPv4 DNS が `ec2-12-345-678-90.compute-1.amazonaws.com` であるとします。SSH コマンドは次のようになります。

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

3. EC2 インスタンスのソフトウェアを更新して、最新のバグ修正とセキュリティ更新を入手します。そのためには、次のコマンドを使用します。

> **Note**
>
> -y オプションを指定すると、確認メッセージを表示せずに更新をインストールします。インストール前に更新を確認するには、このオプションを省略します。

```
sudo dnf update -y
```

4. Amazon Linux 2023 で PostgreSQL から psql コマンドラインクライアントをインストールするには、次のコマンドを実行します。

```
sudo dnf install postgresql15
```

5. Aurora PostgreSQL DB クラスターに接続します。例えば、次のコマンドを入力します。このアクションにより、psql クライアントを使用して、Aurora PostgreSQL DB クラスターに接続できます。

   endpoint をライターインスタンスのエンドポイントに置き換え、*postgres* のために接続するデータベース名 --dbname を、*postgres* に使用したマスターユーザー名に置き換えます。パスワードの入力を求められたときに使用したマスターパスワードを入力します。

```
psql --host=endpoint --port=5432 --dbname=postgres --username=postgres
```

   ユーザーのパスワードを入力すると、次のような出力が表示されます。

```
psql (14.3, server 14.6)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256,
 compression: off)
Type "help" for help.

postgres=>
```

Aurora PostgreSQL DB クラスターへの接続の詳細については、「Amazon Aurora PostgreSQL DB クラスターへの接続」を参照してください。DB クラスターに接続できない場合は、Amazon RDS DB インスタンスに接続できない を参照してください。

セキュリティのためには、暗号化された接続を使用することがベストプラクティスです。クライアントとサーバーが同じ VPC にあり、ネットワークが信頼できる場合に限り、暗号化されていない PostgreSQL 接続を使用します。暗号化された接続の使用については、「SSL/TLS での Aurora PostgreSQL データの保護」を参照してください。

6. SQL コマンドを実行する。

   例えば、次の SQL コマンドは、現在の日付と時刻を表示します。

```
SELECT CURRENT_TIMESTAMP;
```

ステップ 4: EC2 インスタンスと DB クラスターを削除する

作成したサンプル EC2 インスタンスと DB クラスターに接続して、探索したら、料金がこれ以上発生しないように、それらを削除します。

AWS CloudFormation を使用してリソースを作成した場合は、このステップをスキップして次のステップに進みます。

EC2 インスタンスを削除するには

1. AWS Management Console にサインインし、Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. ナビゲーションペインで、[インスタンス] を選択します。
3. EC2 インスタンスを選択し、[インスタンスの状態]、[インスタンスの終了] の順に選択します。
4. 確認を求めるメッセージが表示されたら、[Terminate (終了)] を選択します。

EC2 インスタンスの削除の詳細については、「Amazon EC2 Linux インスタンス用ユーザーガイド」の「インスタンスの終了」を参照してください。

DB クラスターを削除するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。

2. [データベース] を選択後、DB クラスターに関連付けられた DB インスタンスを選択します。
3. [アクション] で、[削除] を選択します。
4. [削除] を選択します。

   DB クラスターに関連付けられている DB インスタンスがすべて削除されると、DB クラスターは自動的に削除されます。

(オプション) CloudFormation で作成された EC2 インスタンスと DB クラスターを削除する

AWS CloudFormation を使用してリソースを作成した場合、作成したサンプル EC2 インスタンスと DB クラスターに接続して、探索した後に CloudFormation を削除することで、料金がこれ以上発生しないようにします。

CloudFormation リソースを削除するには

1. AWS CloudFormation コンソールを開きます。
2. CloudFormation コンソールの [スタック] ページで、ルートスタック (VPCStack BastionStack、BastionStack、または APGNS という名前のスタック) を選択します。
3. [削除] を選択します。
4. 確認を求めるメッセージが表示されたら、[削除] を選択します。

CloudFormation のスタックの削除に関する詳細は、AWS CloudFormation ユーザーガイドの「AWS CloudFormation コンソールでスタックの削除」を参照してください。

(オプション) DB クラスターを Lambda 関数に接続する

Aurora Postgre DB クラスターを Lambda サーバーレスコンピューティングリソースに接続することもできます。Lambda 関数を使用すると、インフラストラクチャをプロビジョニングしたり管理したりせずにコードを実行できます。Lambda 関数を使用すると、1 日に 数十件のイベントから 1 秒間に数百件のイベントまで、あらゆる規模のコード実行リクエストに自動的に応答することもできます。詳細については、「Lambda 関数と Aurora DB クラスターを自動的に接続する」を参照してください。

チュートリアル: ウェブサーバーと Amazon Aurora DB クラスターを作成する

このチュートリアルでは、PHP を使用して Apache ウェブサーバーをインストールする方法と MariaDB、MySQL、または PostgreSQL データベースを作成する方法を示します。ウェブサーバーは Amazon Linux 2023 を使用して Amazon EC2 インスタンスで実行し、Aurora MySQL DB クラスターと Aurora PostgreSQL DB クラスターのいずれかを選択できます。Amazon EC2 インスタンスと DB クラスターはいずれも、Amazon VPC サービスに基づき、仮想プライベートクラウド (VPC) で実行されます。

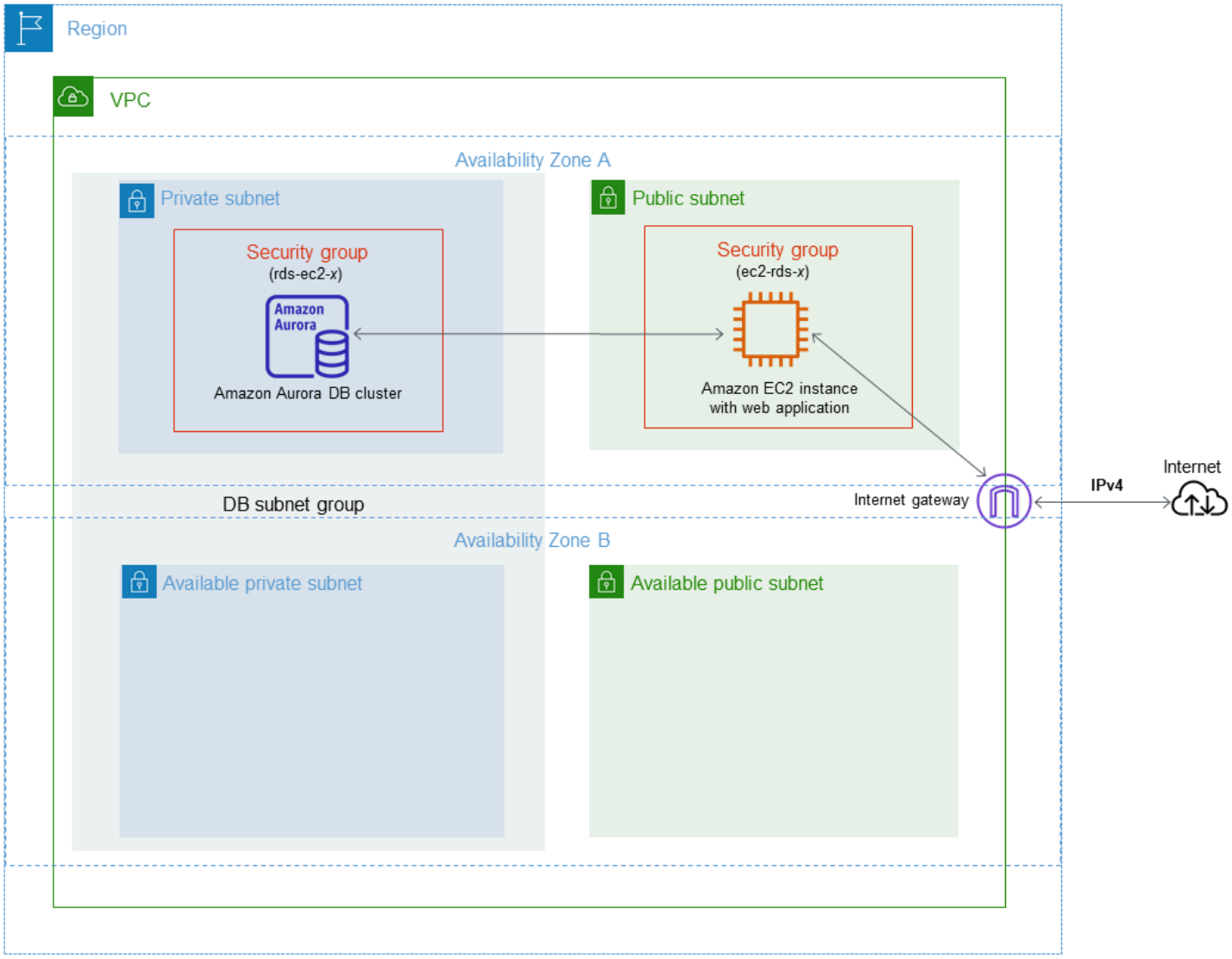
> **Important**
>
> AWS アカウントを作成するための料金はかかりません。ただし、このチュートリアルを完了すると、使用する AWS リソースのコストが発生する可能性があります。これらのリソースが不要になった場合は、チュートリアルの完了後に削除できます。

> **Note**
>
> このチュートリアルは Amazon Linux 2023 で機能します。他の Linux のバージョンでは機能しない場合があります。

次のチュートリアルでは、AWS アカウント のデフォルトの VPC、サブネット、およびセキュリティグループを使用する EC2 インスタンスを作成します。このチュートリアルでは、DB クラスターを作成し、作成した EC2 インスタンスとの接続を自動的にセットアップする方法を示します。次に、EC2 インスタンスにウェブサーバーをインストールする方法を示します。DB クラスターライターのエンドポイントを使用して、ウェブサーバーを VPC の DB クラスターに接続します。

1. EC2 インスタンスの起動
2. Amazon Aurora DB クラスターの作成
3. EC2 インスタンスにウェブサーバーをインストールします

次の図は、チュートリアルが完了した時点の設定を示しています。



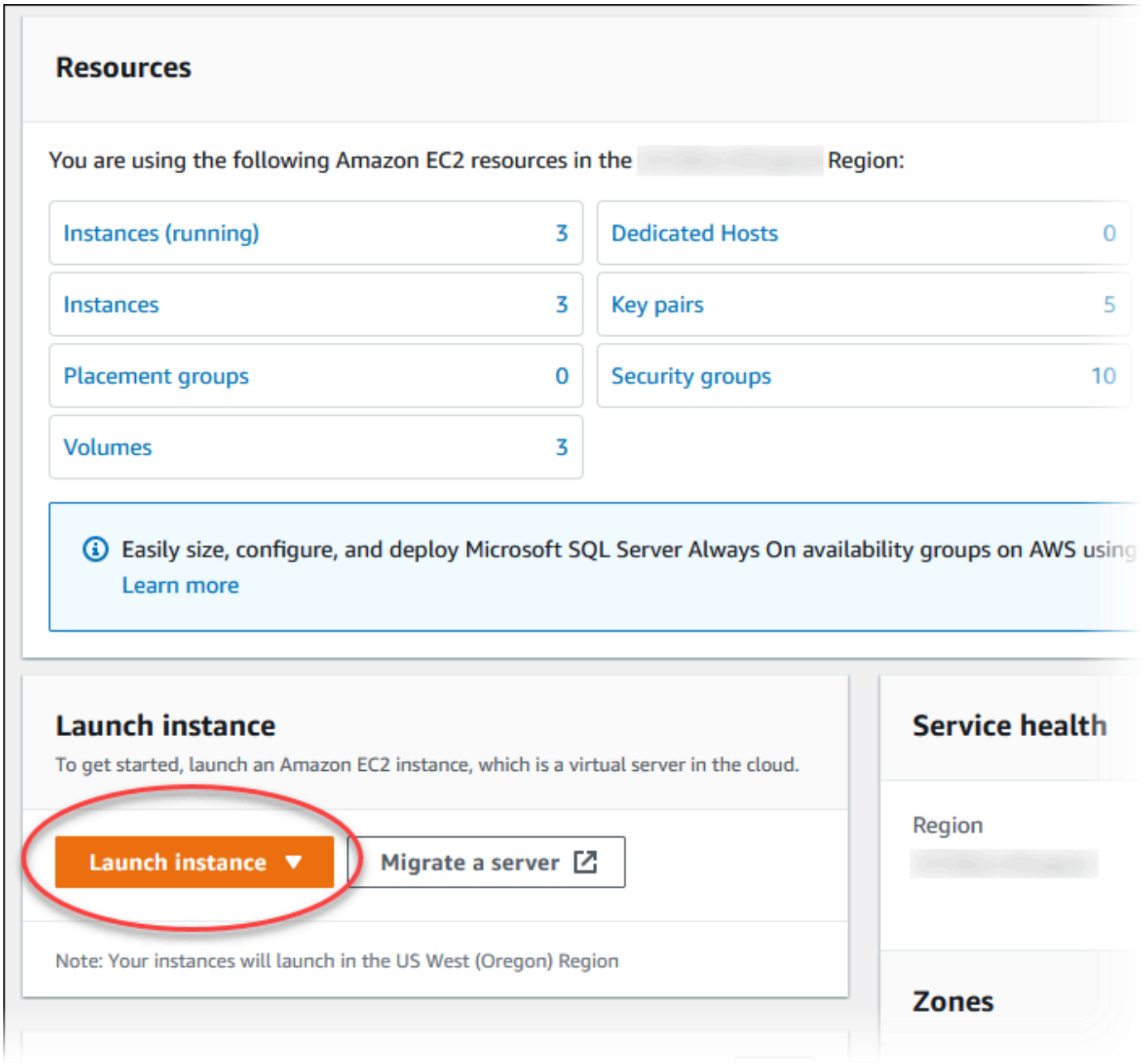
> **Note**
>
> チュートリアルを完了すると、VPC 内の各アベイラビリティーゾーンにパブリックサブネットとプライベートサブネットができます。このチュートリアルでは、AWS アカウント にデフォルトの VPC を使用し、EC2 インスタンスと DB クラスター間の接続を自動的に設定します。このシナリオで代わりに新しい VPC を設定する場合は、チュートリアル: DB クラスターで使用する VPC を作成する (IPv4 専用) のタスクを完了してください。

EC2 インスタンスの起動

VPC のパブリックサブネットで Amazon EC2 インスタンスを作成します。

EC2 インスタンスを起動するには

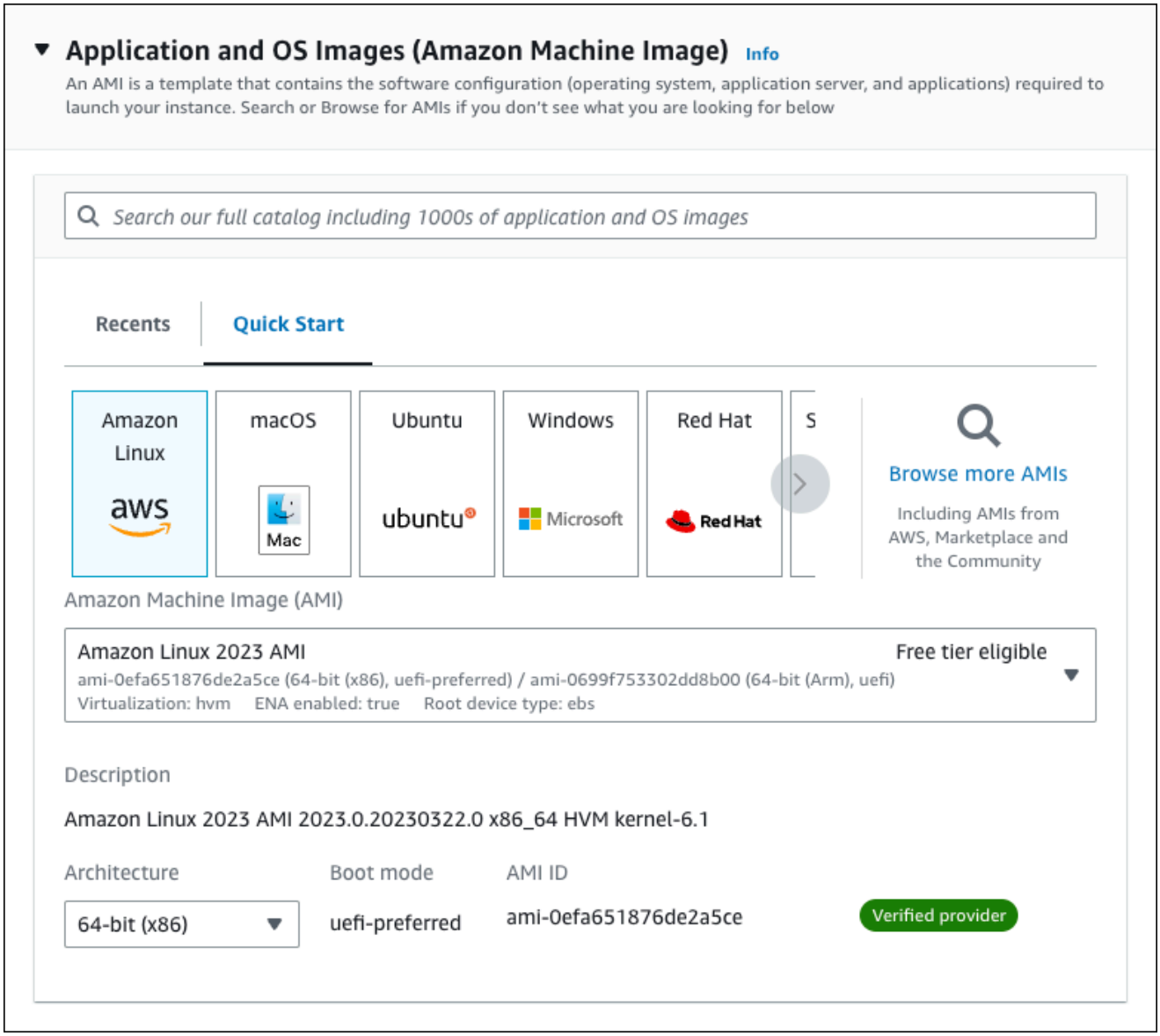
1. AWS Management Console にサインインし、Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. AWS Management Console の右上で、EC2 インスタンスを作成する AWS リージョン を選択します。
3. 次に示すように、[EC2 ダッシュボード] を選択し、次に [インスタンスの作成] を選択します。



4. [インスタンスを起動] ページで次の設定を選択します。

a. [Name and tags] (名前とタグ) の、[Name] (名前) で、**`tutorial-ec2-instance-web-server`** と入力します。

b. [アプリケーションおよび OS イメージ (Amazon マシンイメージ)] で、[Amazon Linux] を選択し、[Amazon Linux 2023 AMI] を選択します。他の選択肢をデフォルトのままにします。



c. [Instance type] (インスタンスタイプ) で [t2.micro] を選択します。

d. [Key pair (login)] (キーペア (ログイン)) で、[Key pair name] (キーペア名) を選択して、既存のキーペアを使用します。Amazon EC2 インスタンスの新しい key pair を作成するには、[Create new key pair] (新しい key pair を作成する) を選択し、[Create key pair] (キーペアを作成する) ウィンドウを使用して作成します。

キーペアの作成については、Linux インスタンス用 Amazon EC2 ユーザーガイドの「キーペアの作成」を参照してください。

e. [Network settings] (ネットワーク設定) で、次の値を設定し、他の値はデフォルトのままにします。

- [Allow SSH traffic from] (SSH トラフィックを許可) で、EC2 インスタンスへの SSH 接続のソースを選択します。

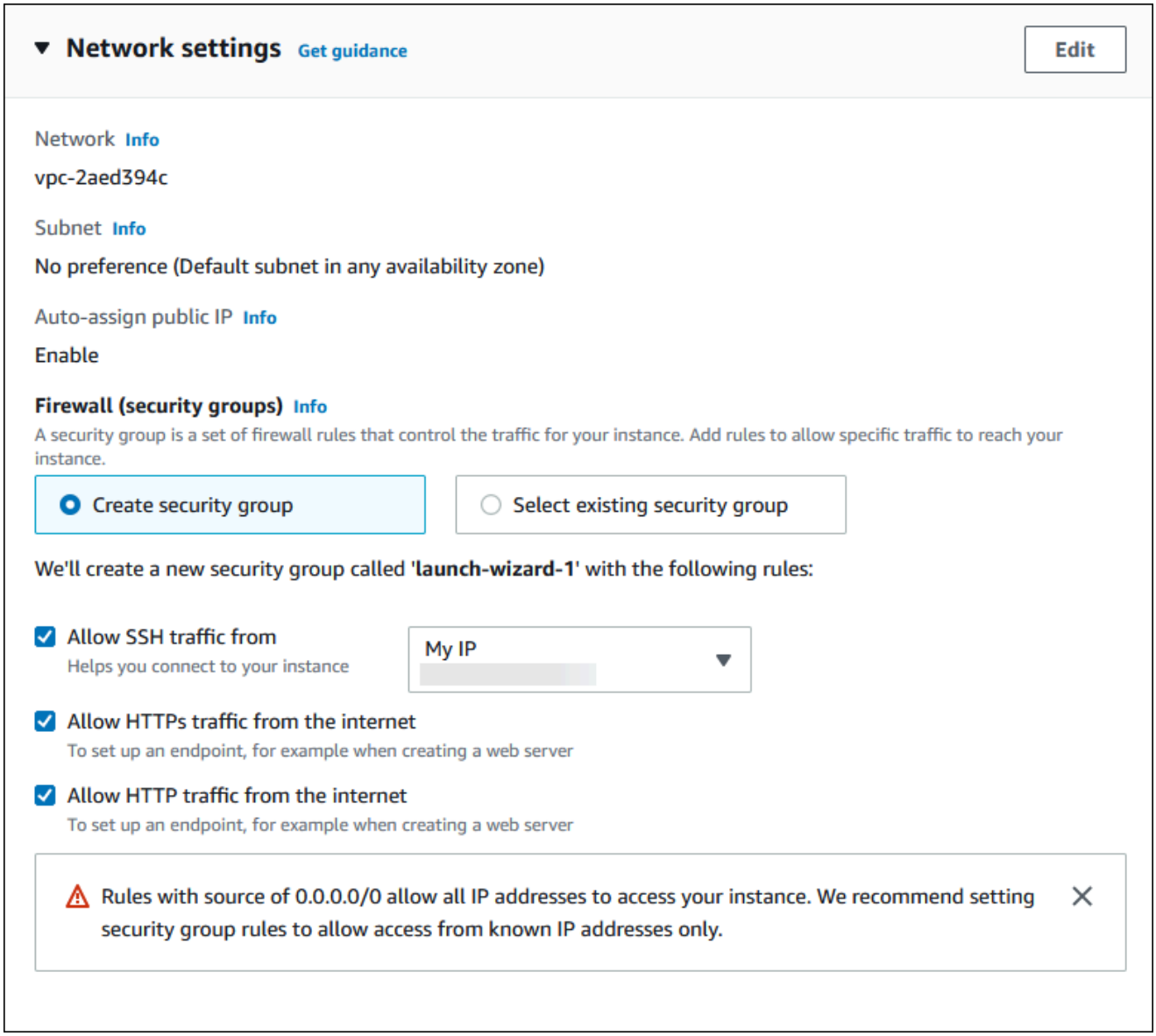
  表示された IP アドレスが SSH 接続に適している場合は、[My IP] (マイ IP)を選択できます。

  それ以外の場合は、Secure Shell (SSH) を使用して VPC の EC2 インスタンスへの接続に使用する IP アドレスを決定します。パブリック IP アドレスを決定するには、別のブラウザウィンドウまたはタブで、https://checkip.amazonaws.com のサービスを使用できます。IP アドレスの例は `203.0.113.25/32` です。

  多くの場合、インターネットサービスプロバイダー (ISP) 経由、またはファイアウォールの内側から静的 IP アドレスなしで接続することがあります。その場合、クライアントコンピュータが使用する IP アドレスの範囲を確認してください。

  > **Warning**
  >
  > SSH アクセスに `0.0.0.0/0` を使用すると、すべての IP アドレスが SSH を使ってパブリックインスタンスにアクセスできるようになります。この方法は、テスト環境で短時間なら許容できますが、実稼働環境では安全ではありません。実稼働環境では、特定の IP アドレスまたは特定のアドレス範囲にのみ、SSH を使ったインスタンスへのアクセスを限定します。

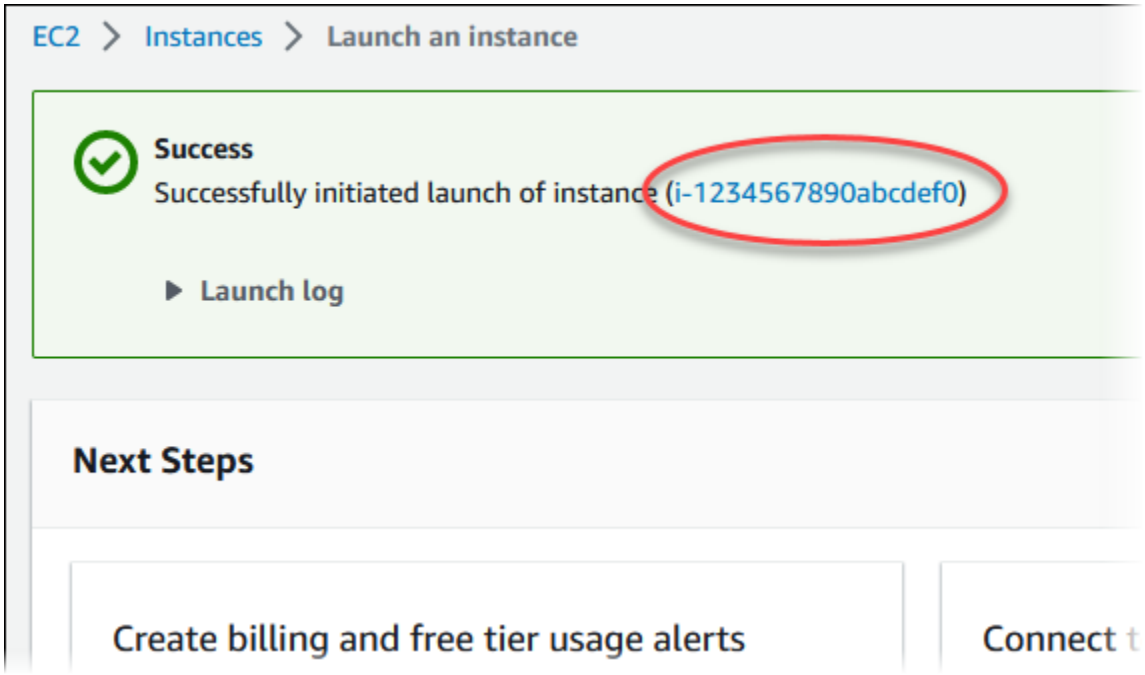
- [Allow HTTPs traffic from the internet] (インターネットからの HTTPs トラフィックを許可する) をオンにします。
- [Allow HTTP traffic from the internet] (インターネットからの HTTP トラフィックを許可する) をオンにします。



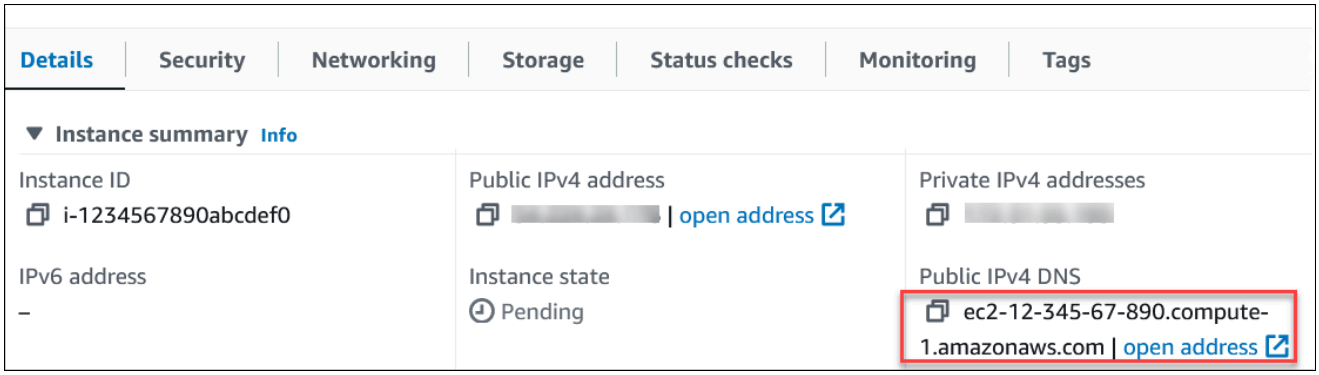
f. 残りのセクションをデフォルト値のままにします。

g. Summary (概要) パネルでインスタンス設定の要約を確認します。準備が完了したら、[Launch instance] (インスタンスを起動) を選択します。

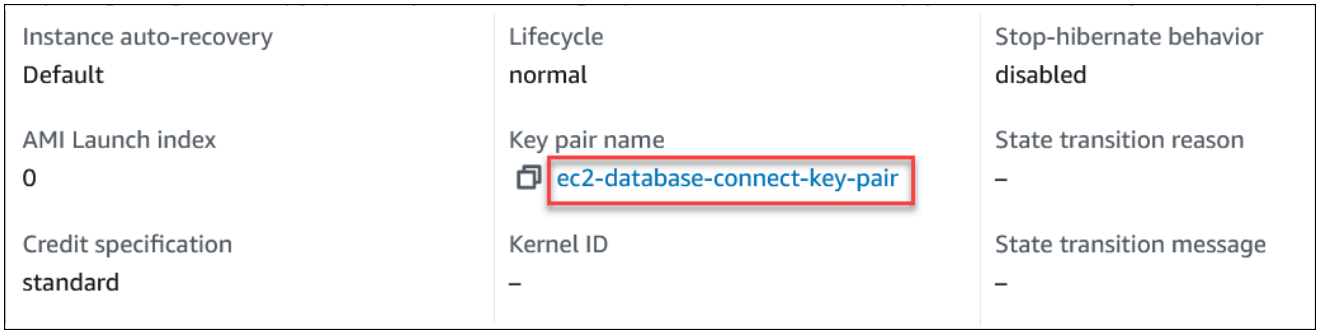
5. [起動ステータス] ページで、新しい EC2 インスタンスの ID (例: `i-1234567890abcdef0`) をメモします。



6. EC2 インスタンス ID を選択して、EC2 インスタンスのリストを開き、EC2 インスタンスを選択します。
7. [詳細] タブで、SSH を使用して接続するときに必要な次の値を書き留めます。
   a. [インスタンスの概要] で、[パブリック IPv4 DNS] の値を書き留めます。



   b. [インスタンスの詳細] で、[キーペア名] の値を書き留めます。



8. インスタンスの [Instance Status] (インスタンスのステータス) が [Running] (実行中) になるまで続行せずに待ちます。
9. Amazon Aurora DB クラスターの作成 を完了します。

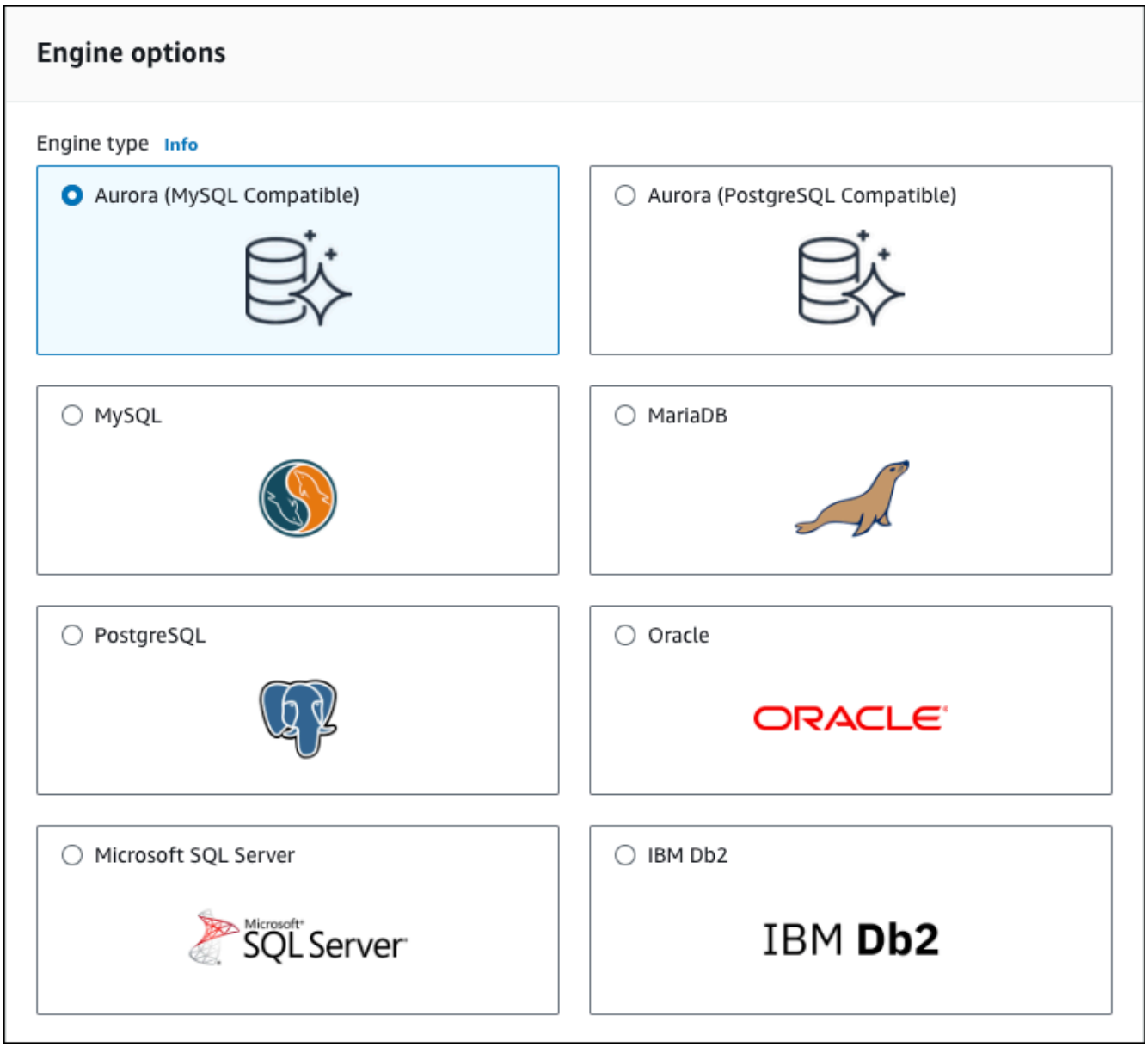
Amazon Aurora DB クラスターの作成

ウェブアプリケーションによって使用されるデータが維持される Amazon Aurora MySQL または Aurora PostgreSQL DB クラスターを作成します。

Aurora MySQL

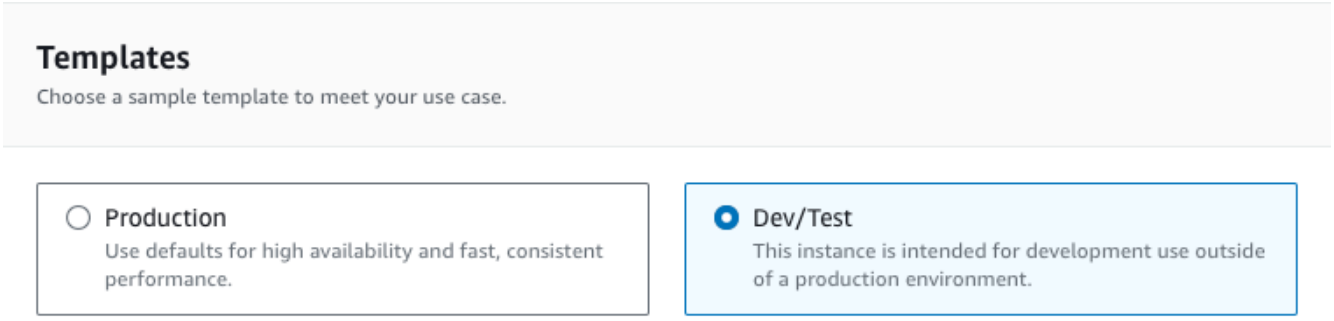
Aurora MySQLDB クラスターを作成するのために

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. AWS Management Console の右上隅で、AWS リージョン が EC2 インスタンスを作成したものと同じであることを確認してください。
3. ナビゲーションペインで [データベース] を選択します。
4. [データベースの作成] を選択します。
5. [データベースの作成] ページで、[標準作成] を選択します。
6. [エンジンのオプション] として、[Aurora (MySQL 互換)] を選択します。



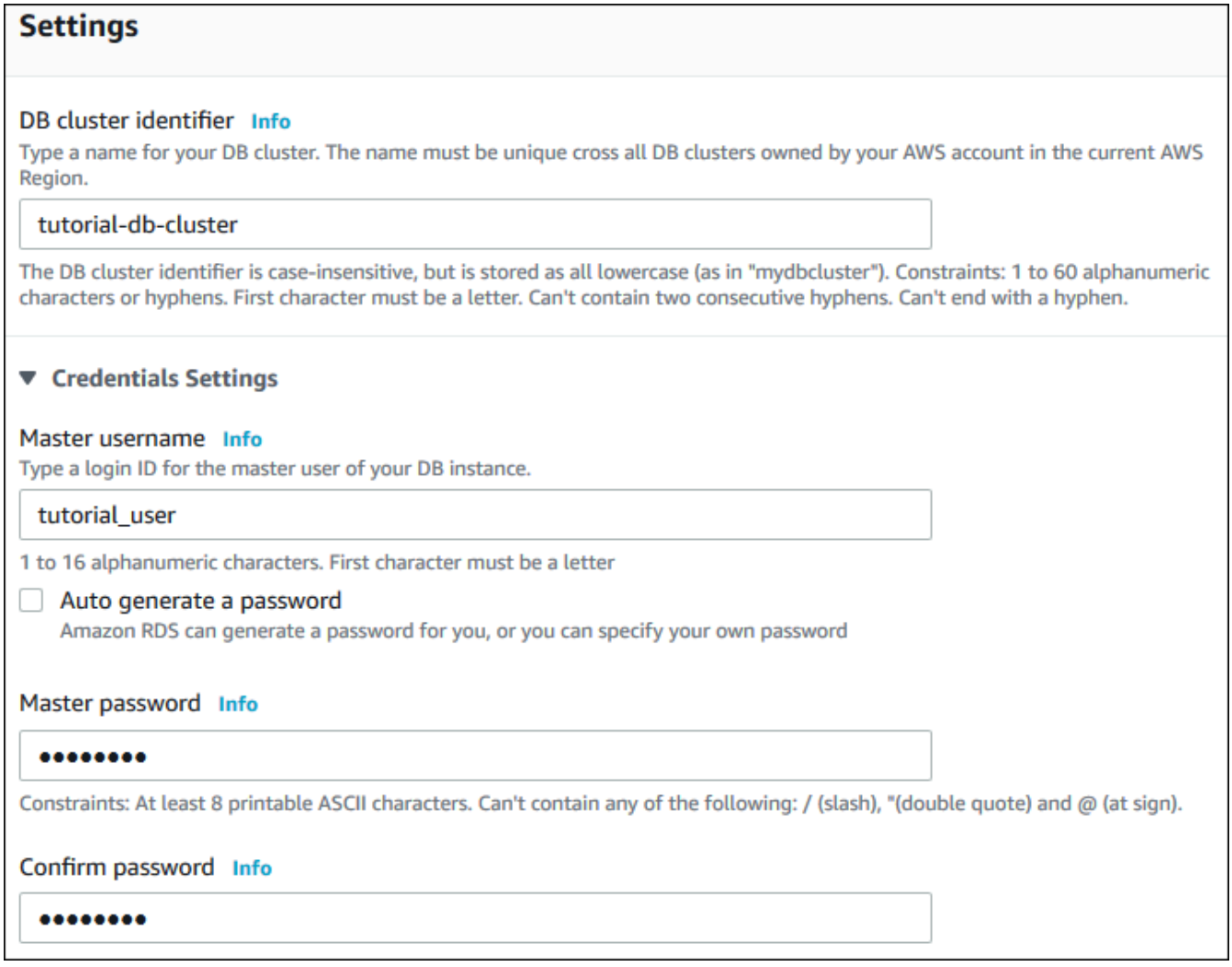
[バージョン] と他のエンジンオプションは、既定値のままにします。

7. [テンプレート] セクションで、[Dev/Test] を選択します。



8. [設定] セクションで、これらの値を設定します。

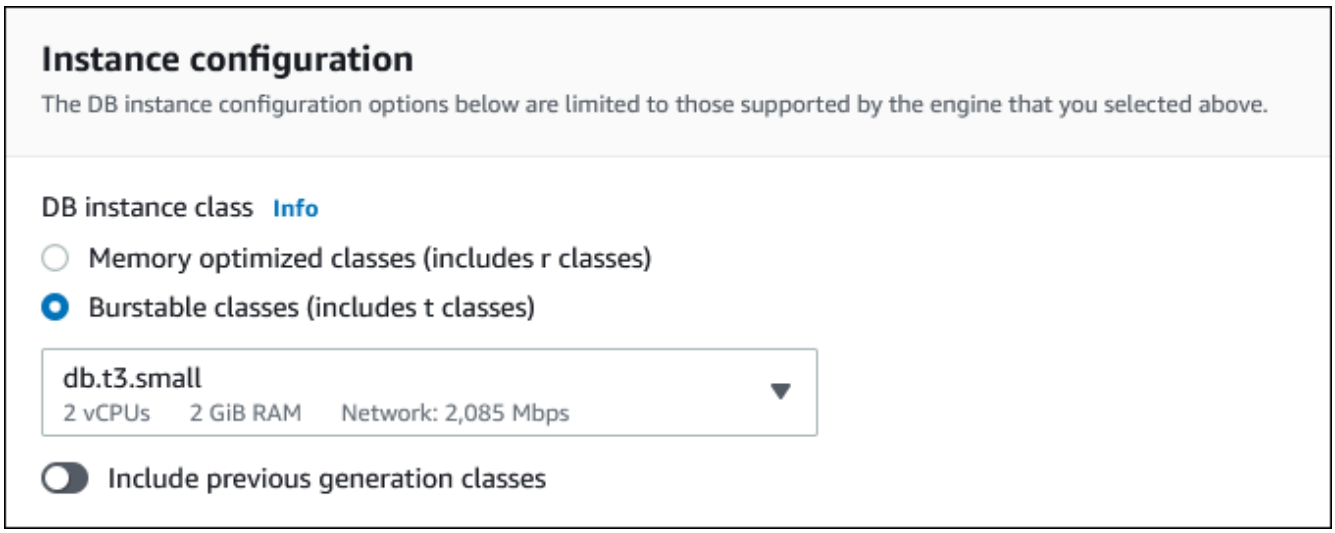
   - [DB cluster identifier](DB クラスター識別子): **`tutorial-db-cluster`** を入力します。
   - [Master username] (マスターユーザー名) — **`tutorial_user`** を入力します。
   - [Auto generate a password] (パスワードの自動生成) — オプションはオフのままにします。
   - [Master password] (マスターパスワード): パスワードを入力します。
   - [パスワードの確認] - パスワードを再入力します。



9. [Instance configuration] (インスタンス設定) セクションで、次の値を設定します。

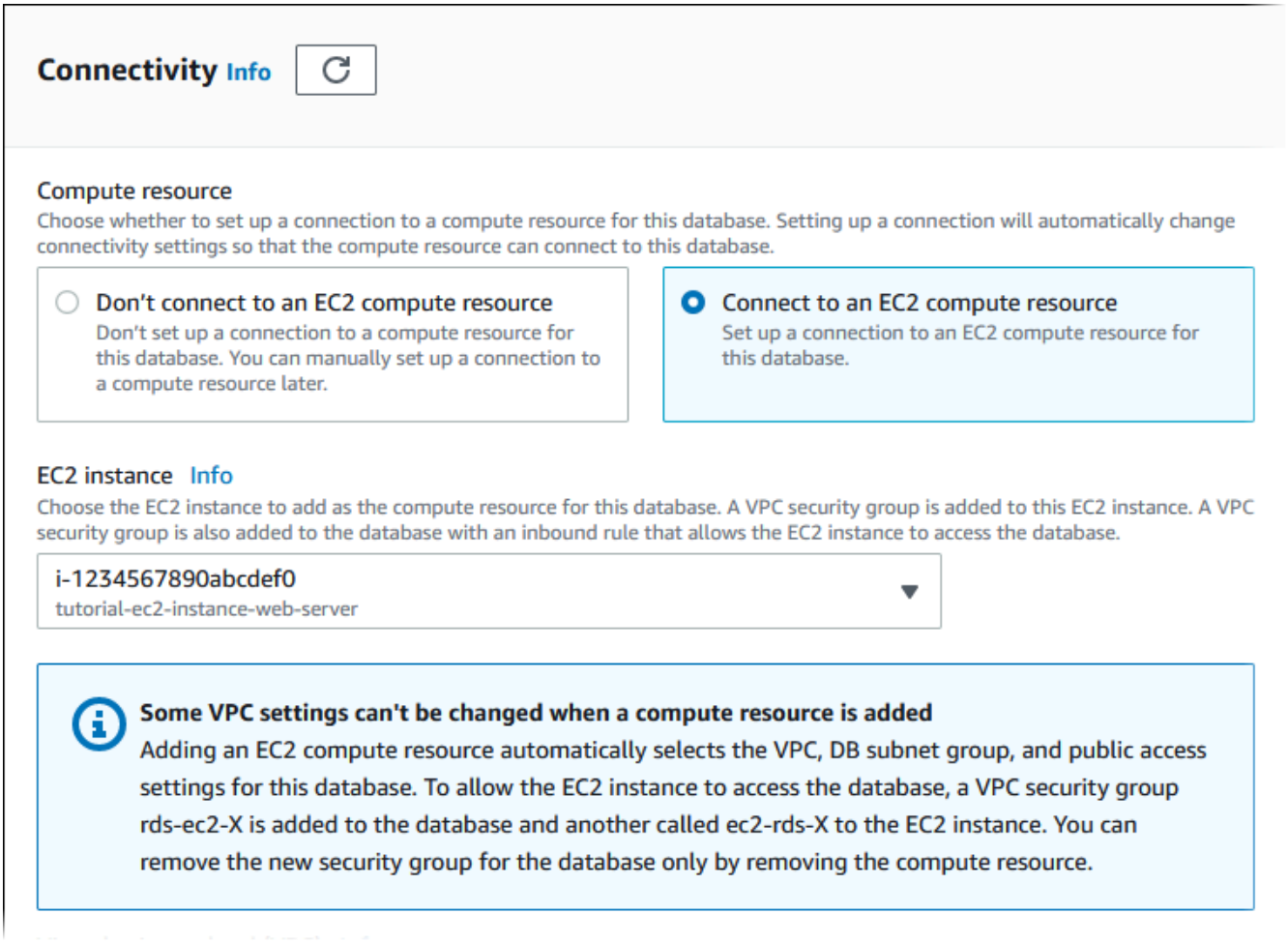
   - バースト可能クラス (t クラスを含む)
   - db.t3.small または db.t3.medium

> **Note**
>
> T DB インスタンスクラスを、開発サーバーおよびテストサーバー、または他の本稼働以外のサーバーにのみ使用することをお勧めします。T インスタンスクラスの詳細については、「DB インスタンスクラスタイプ」を参照してください。



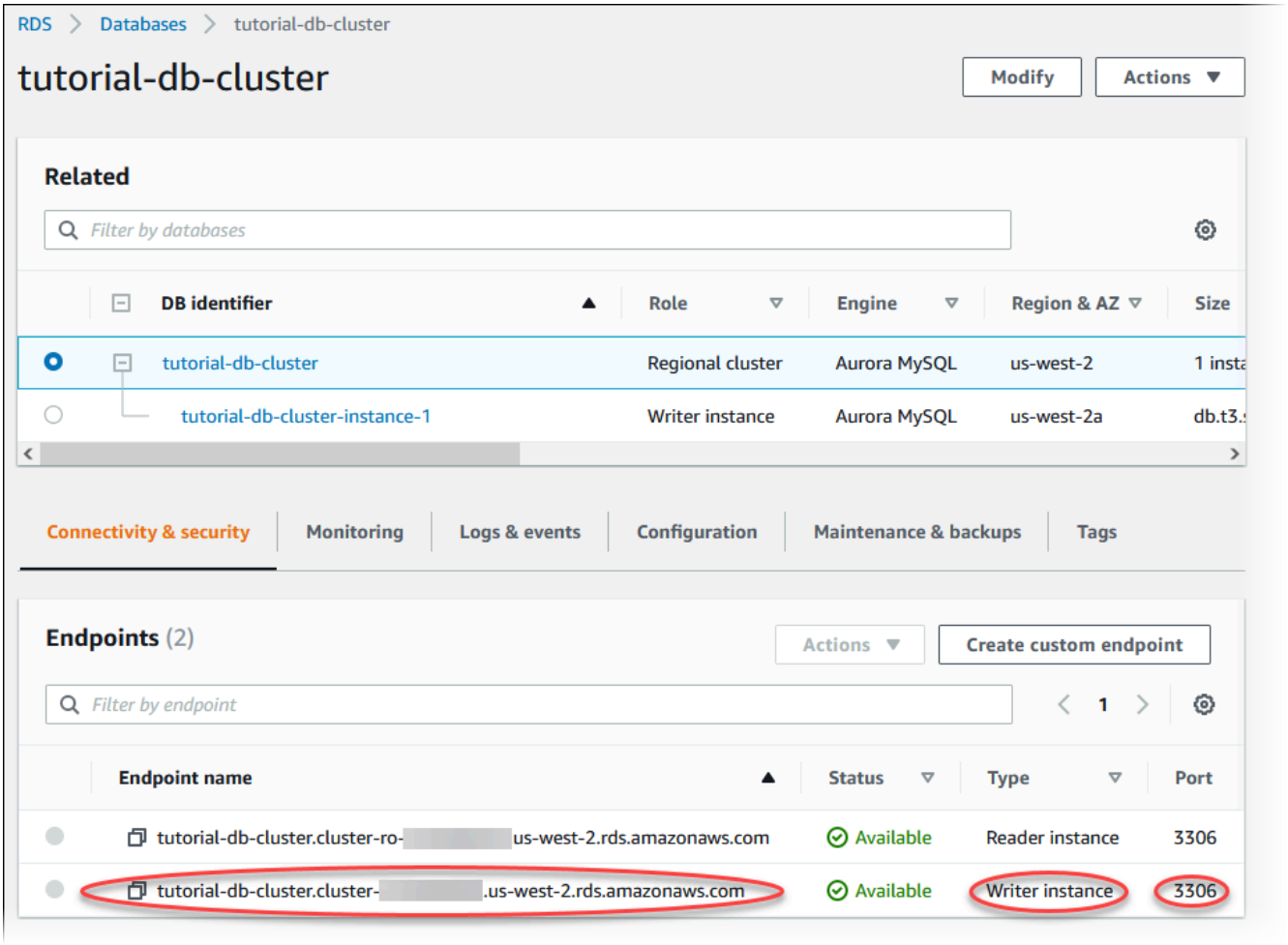
10. [Availability & durability] (可用性と耐久性) セクションで、デフォルト値を使用します。
11. [Connectivity] (接続) セクションで、次の値を設定し、他の値はデフォルトのままにします。

    - [Compute resource] (コンピューティングリソース) で、選択してください[Connect to an EC2 compute resource] (EC2 コンピューティングリソースに接続する) を選択します。
    - [EC2 instance] (EC2 インスタンス) で、以前に作成した [tutorial-ec2-instance-web-server] などの EC2 インスタンスを選択します。



12. [Additional configuration (追加設定)] セクションを開き、[Initial database name (初期データベース名)] に **`sample`** と入力します。その他のオプションについては、デフォルト設定のままにしておきます。
13. Aurora MySQL DB クラスターを作成するには、[Create database (データベースの作成)] を選択します。

    新しい DB クラスターは、[作成中] ステータスとして [データベース] リストに表示されます。
14. 新しい DB クラスターの [ステータス] が [Available (使用可能)] と表示されるまで待ちます。詳細を表示する DB クラスターの名前を選択します。
15. [接続とセキュリティ] セクションで、書き込み DB インスタンスの [エンドポイント] および [ポート] を表示します。



書き込み DB インスタンスのエンドポイントとポートを書き留めます。この情報を使用して、ウェブサーバーを DB クラスターに接続します。

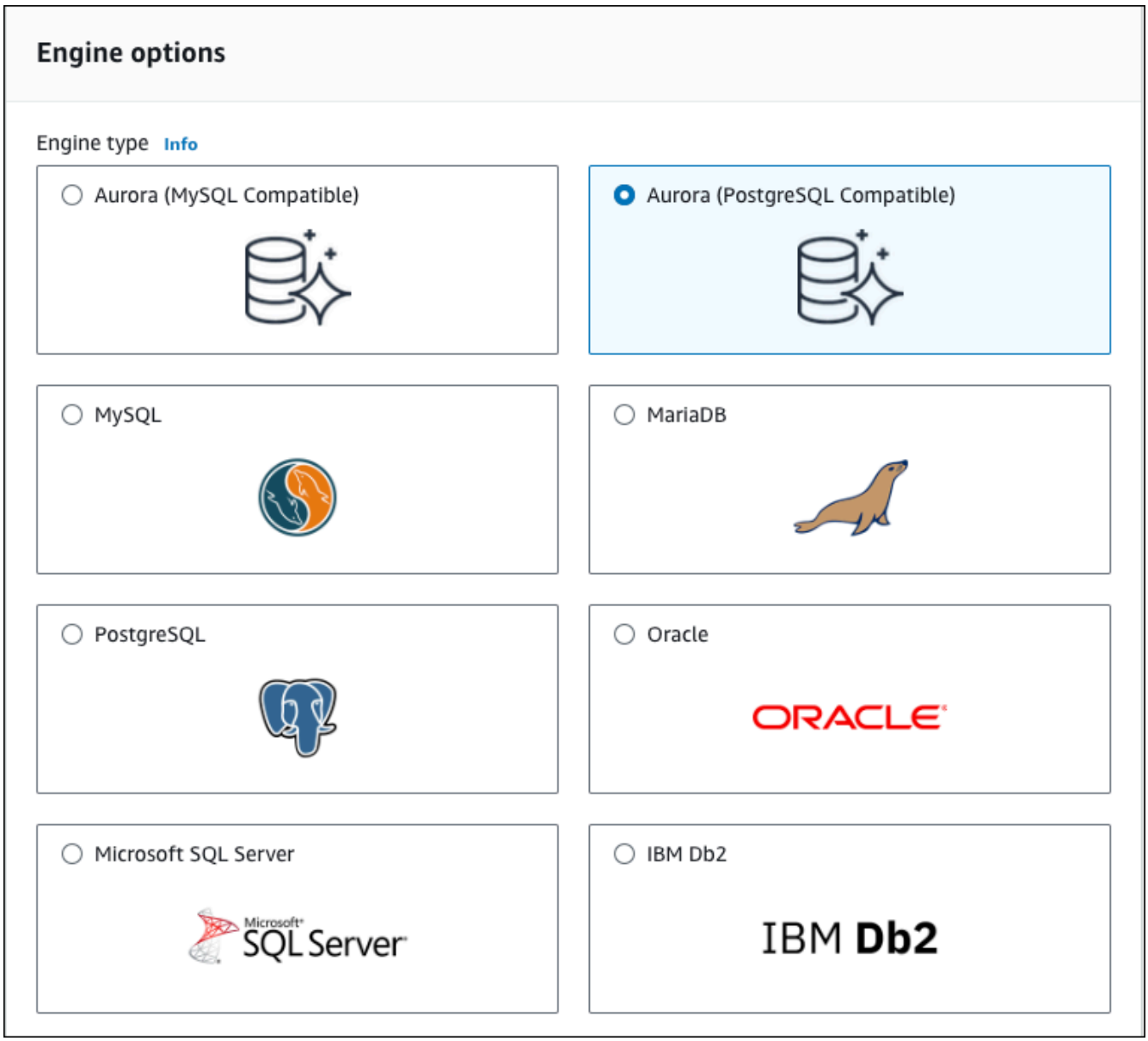
16. EC2 インスタンスにウェブサーバーをインストールします を完了します。

Aurora PostgreSQL

Aurora PostgreSQL DB クラスターを作成するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. AWS Management Console の右上隅で、AWS リージョン が EC2 インスタンスを作成したものと同じであることを確認してください。
3. ナビゲーションペインで [データベース] を選択します。
4. [データベースの作成] を選択します。

5. [データベースの作成] ページで、[標準作成] を選択します。
6. [エンジンオプション] で、[Aurora (PostgreSQL 互換)] を選択します。

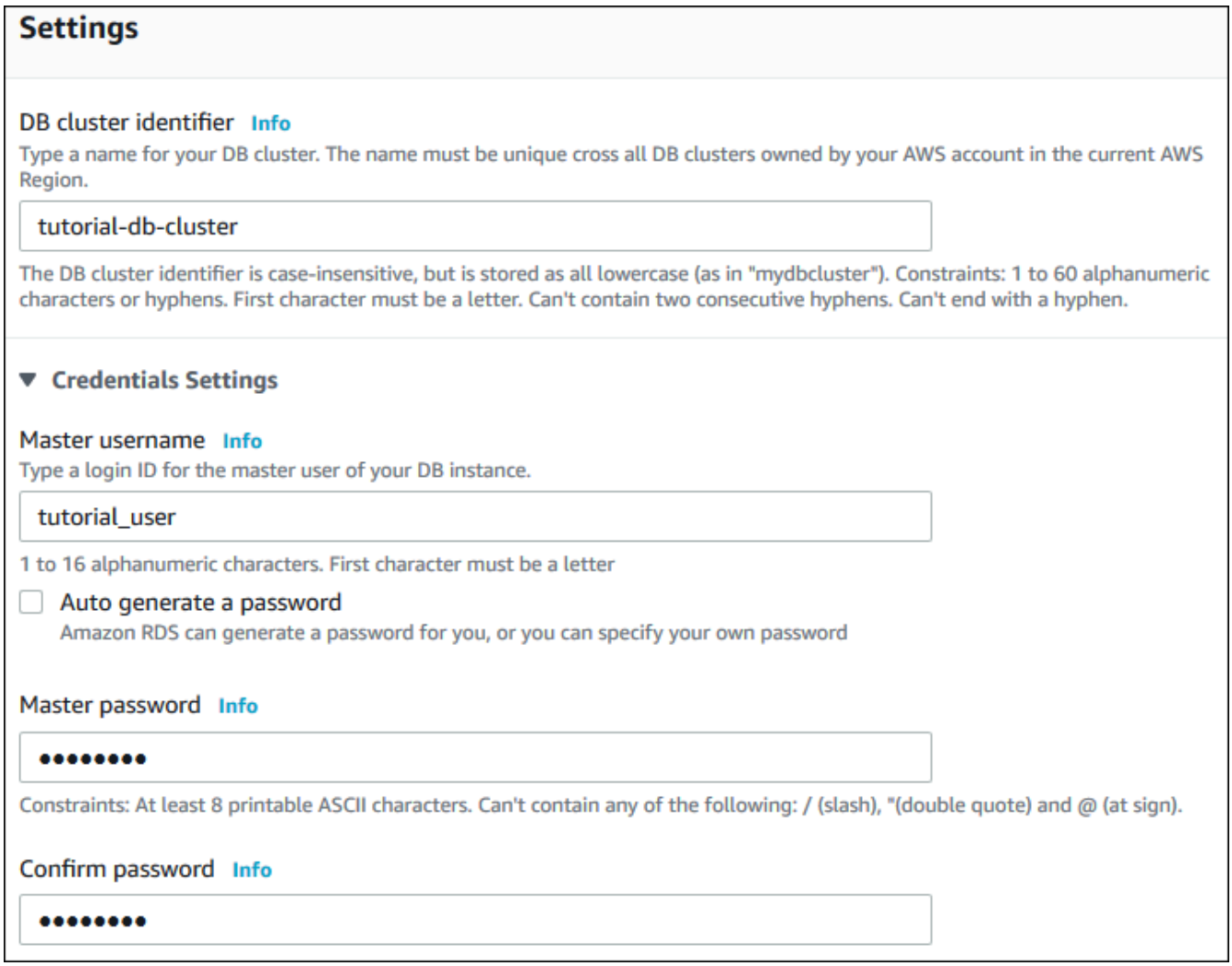


[バージョン] と他のエンジンオプションは、既定値のままにします。

7. [テンプレート] セクションで、[Dev/Test] を選択します。

8. [設定] セクションで、これらの値を設定します。

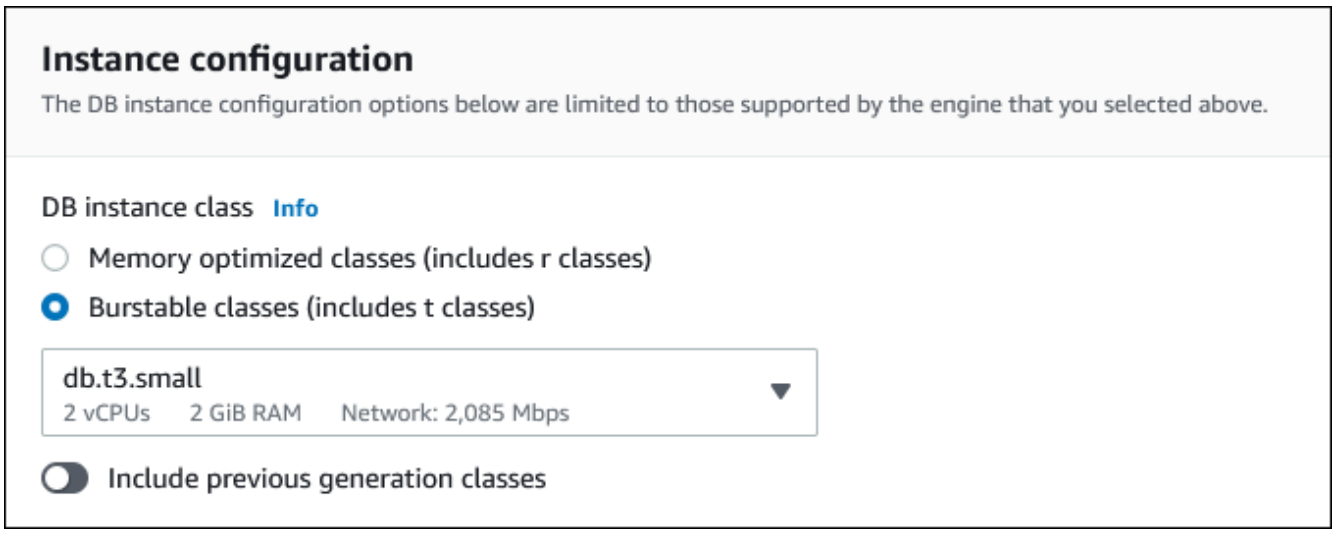
   - [DB cluster identifier](DB クラスター識別子): **`tutorial-db-cluster`** を入力します。
   - [Master username] (マスターユーザー名) — **`tutorial_user`** を入力します。
   - [Auto generate a password] (パスワードの自動生成) — オプションはオフのままにします。
   - [Master password] (マスターパスワード): パスワードを入力します。
   - [Confirm password] - パスワードを再入力します。



9. [Instance configuration] (インスタンス設定) セクションで、次の値を設定します。

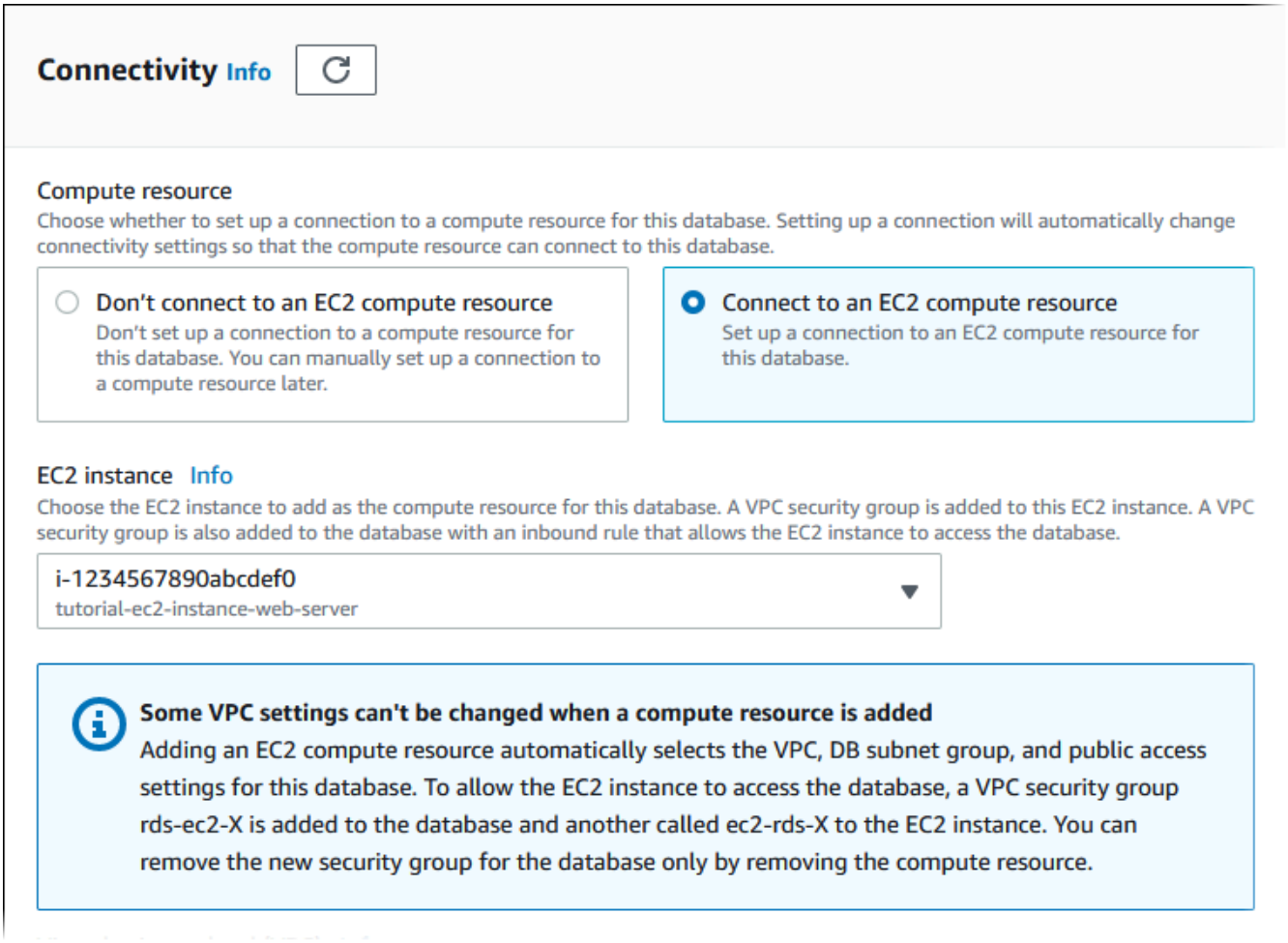
   - バースト可能クラス (t クラスを含む)
   - db.t3.small または db.t3.medium

> **Note**
>
> T DB インスタンスクラスを、開発サーバーおよびテストサーバー、または他の本稼働以外のサーバーにのみ使用することをお勧めします。T インスタンスクラスの詳細については、「DB インスタンスクラスタイプ」を参照してください。



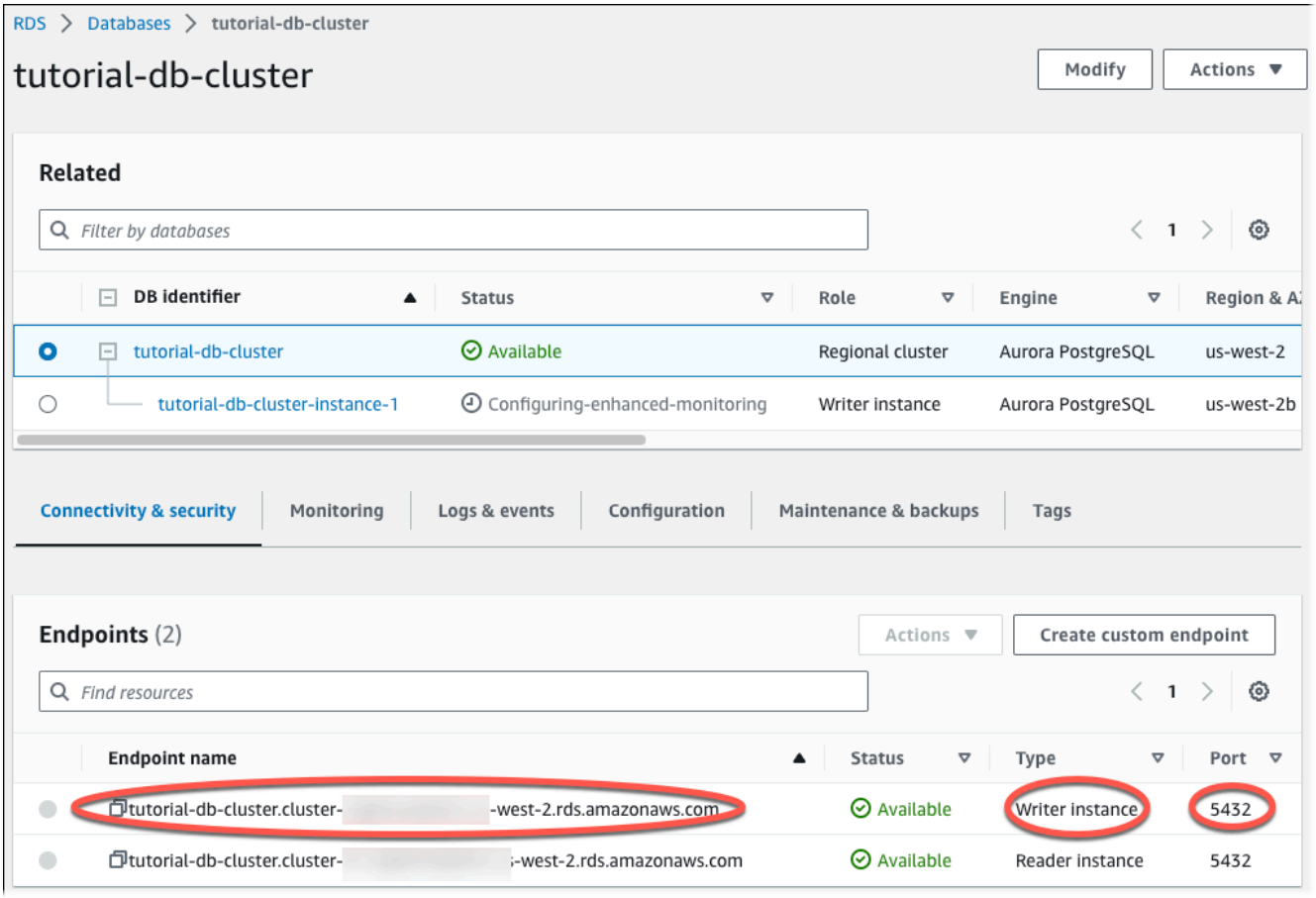
10. [Availability & durability] (可用性と耐久性) セクションで、デフォルト値を使用します。
11. [Connectivity] (接続) セクションで、次の値を設定し、他の値はデフォルトのままにします。

    - [Compute resource] (コンピューティングリソース) で、選択してください[Connect to an EC2 compute resource] (EC2 コンピューティングリソースに接続する) を選択します。
    - [EC2 instance] (EC2 インスタンス) で、以前に作成した [tutorial-ec2-instance-web-server] などの EC2 インスタンスを選択します。



12. [Additional configuration (追加設定)] セクションを開き、[Initial database name (初期データベース名)] に **`sample`** と入力します。その他のオプションについては、デフォルト設定のままにしておきます。
13. Aurora PostgreSQL DB クラスターを作成するには、[データベースの作成] を選択します。

    新しい DB クラスターは、[作成中] ステータスとして [データベース] リストに表示されます。
14. 新しい DB クラスターの [ステータス] が [Available (使用可能)] と表示されるまで待ちます。詳細を表示する DB クラスターの名前を選択します。
15. [接続とセキュリティ] セクションで、書き込み DB インスタンスの [エンドポイント] および [ポート] を表示します。



書き込み DB インスタンスのエンドポイントとポートを書き留めます。この情報を使用して、ウェブサーバーを DB クラスターに接続します。

16. EC2 インスタンスにウェブサーバーをインストールします を完了します。

EC2 インスタンスにウェブサーバーをインストールします

EC2 インスタンスの起動 で作成した EC2 インスタンスにウェブサーバーをインストールします。ウェブサーバーは、Amazon Aurora DB クラスターの作成 で作成した Amazon Aurora DB クラスターに接続します。

PHP と MariaDB を使用する Apache ウェブサーバーのインストール

EC2 インスタンスに接続し、ウェブサーバーをインストールします。

EC2 インスタンスに接続し、PHP を使用して Apache ウェブサーバーをインストールするには

1. Linux インスタンスに関する Amazon EC2 ユーザーガイドの「Linux インスタンスに接続する」のステップに従って、先ほど作成した EC2 インスタンスに接続します。

SSH を使用して EC2 インスタンスに接続することをお勧めします。SSH クライアントユーティリティが Windows、Linux、または Mac にインストールされている場合は、次のコマンド形式でインスタンスに接続できます。

```
ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
```

例えば、ec2-database-connect-key-pair.pem が Linux の /dir1 に保存されていて、EC2 インスタンスのパブリック IPv4 DNS が ec2-12-345-678-90.compute-1.amazonaws.com であるとします。SSH コマンドは次のようになります。

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

2. EC2 インスタンスのソフトウェアを更新して、最新のバグ修正とセキュリティ更新を入手します。これを行うには、次のコマンドを使用します。

> **Note**
>
> -y オプションを指定すると、確認メッセージを表示せずに更新をインストールします。インストール前に更新を確認するには、このオプションを省略します。

```
sudo dnf update -y
```

3. 更新が完了した後、次のコマンドを使用して、Apache ウェブサーバー、PHP、および MariaDB または PostgreSQL ソフトウェアをインストールします。このコマンドは、複数のソフトウェアパッケージおよび関連する依存関係を同時にインストールします。

   MariaDB & MySQL

   ```
   sudo dnf install -y httpd php php-mysqli mariadb105
   ```

   PostgreSQL

   ```
   sudo dnf install -y httpd php php-pgsql postgresql15
   ```

エラーが表示された場合、インスタンスはおそらく Amazon Linux 2023 AMI で起動していません。代わりに Amazon Linux 2 AMI を使用している可能性があります。次のコマンドを使用して、Amazon Linux のバージョンを表示できます。

```
cat /etc/system-release
```

詳細については、「インスタンスソフトウェアの更新」を参照してください。

4. 次に示すコマンドを使用してウェブサーバーを起動します。

```
sudo systemctl start httpd
```

ウェブサーバーが正しくインストールされ、起動されているかどうかをテストできます。これを行うには、ウェブブラウザのアドレスバーに EC2 インスタンスのパブリックドメインネームシステム (DNS) 名を入力します (例: `http://ec2-42-8-168-21.us-west-1.compute.amazonaws.com`)。ウェブサーバーが実行中の場合、Apache テストページが表示されます。

Apache テストページが表示されない場合は、チュートリアル: DB クラスターで使用する VPC を作成する (IPv4 専用) で作成した VPC セキュリティグループのインバウンドルールを確認してください。インバウンドルールに、ウェブサーバーへの接続に使用する IP アドレスに対する HTTP (ポート 80) アクセスを許可するルールが含まれていることを確認します。

> **Note**
>
> Apache テストページは、ドキュメントのルートディレクトリ `/var/www/html` にコンテンツがないときのみ表示されます。ドキュメントルートディレクトリにコンテンツを追加すると、そのコンテンツが EC2 インスタンスのパブリック DNS アドレスに表示されます。これより前では、Apache のテストページに表示されます。

5. `systemctl` コマンドを使用して、システムがブートするたびにウェブサーバーが起動するように設定します。

```
sudo systemctl enable httpd
```

ec2-user が Apache ウェブサーバーのデフォルトルートディレクトリにあるファイルを管理できるようにするには、/var/www ディレクトリの所有権とアクセス許可を変更します。このタスクを行うには、複数の方法があります。このチュートリアルでは、ec2-user を apache グループに追加し、apache ディレクトリの所有権を /var/www グループに付与し、グループへの書き込み権限を割り当てます。

Apache ウェブサーバーのファイルアクセス許可を設定するには

1. ec2-user ユーザーを apache グループに追加します。

```
sudo usermod -a -G apache ec2-user
```

2. ログアウトしてアクセス許可を更新して新しい apache グループを含めます。

```
exit
```

3. 再度ログインし、apache コマンドを使用して groups グループが存在していることを確認します。

```
groups
```

出力は次のようになります。

```
ec2-user adm wheel apache systemd-journal
```

4. /var/www ディレクトリとそのコンテンツのグループ所有権を apache グループに変更します。

```
sudo chown -R ec2-user:apache /var/www
```

5. /var/www およびそのサブディレクトリのディレクトリアクセス許可を変更して、グループの書き込みアクセス許可を追加し、後で作成するサブディレクトリのグループ ID を設定します。

```
sudo chmod 2775 /var/www
find /var/www -type d -exec sudo chmod 2775 {} \;
```

6. /var/www ディレクトリおよびそのサブディレクトリのファイルごとにアクセス許可を繰り返し変更し、グループの書き込みアクセス許可を追加します。

```
find /var/www -type f -exec sudo chmod 0664 {} \;
```

これで、ec2-user (および apache グループの将来のメンバー) は、Apache ドキュメントルートにファイルを追加、削除、編集できるようになります。これにより、静的ウェブサイトや PHP アプリケーションなどのコンテンツを追加できます。

> **Note**
>
> HTTP プロトコルを実行するウェブサーバーは、送受信したデータのトランスポートセキュリティを提供しません。ウェブブラウザを使用して HTTP サーバーに接続すると、ネットワーク経路上のどこからでも、多くの情報が誰でも閲覧できるようになります。この情報には、アクセスした URL、受信したウェブページのコンテンツ、HTML フォームの内容 (パスワードなど) が含まれます。
> ウェブサーバーを保護するためのベストプラクティスとして、HTTPS (HTTP Secure) のサポートをインストールしてください。このプロトコルでは、SSL/TLS を使用してデータを保護します。詳細については、Amazon EC2 ユーザーガイドの「チュートリアル: Amazon Linux AMI を使用した SSL/TLS の設定」を参照してください。

Apache ウェブサーバーを DB クラスターに接続する

次に、Amazon Aurora DB クラスターに接続する Apache ウェブサーバーへのコンテンツを追加します。

DB クラスターに接続する Apache ウェブサーバーにコンテンツを追加するには

1. EC2 インスタンスに接続したまま、ディレクトリを /var/www に変更し、inc という名前の新しいサブディレクトリを作成します。

   ```
   cd /var/www
   mkdir inc
   cd inc
   ```

2. inc という dbinfo.inc ディレクトリに新しいファイルを作成し、nano を呼び出して (または任意のテキストエディタで) ファイルを編集します。

   ```
   >dbinfo.inc
   ```

```
nano dbinfo.inc
```

3. dbinfo.inc ファイルに次のコンテンツを追加します。ここで、*db_instance_endpoint* は、DB クラスター の、ポートのない、DB クラスターライターエンドポイントです。

> **Note**
>
> ユーザー名とパスワード情報は、ウェブサーバーのドキュメントルートに含まれていないフォルダに配置することをお勧めします。これにより、セキュリティ情報が公開される可能性が低くなります。
> アプリケーションで master password を適切なパスワードに変更してください。

```
<?php

define('DB_SERVER', 'db_cluster_writer_endpoint');
define('DB_USERNAME', 'tutorial_user');
define('DB_PASSWORD', 'master password');
define('DB_DATABASE', 'sample');
?>
```

4. dbinfo.inc ファイルを保存して閉じます。nano を使用している場合は、Ctrl+S と Ctrl+X を使用してファイルを保存して閉じます。
5. ディレクトリを /var/www/html に変更します。

```
cd /var/www/html
```

6. html という SamplePage.php ディレクトリに新しいファイルを作成し、nano を呼び出して (または任意のテキストエディタで) ファイルを編集します。

```
>SamplePage.php
nano SamplePage.php
```

7. SamplePage.php ファイルに次のコンテンツを追加します。

   MariaDB & MySQL

```
<?php include "../inc/dbinfo.inc"; ?>
<html>
```

```
<body>
<h1>Sample page</h1>
<?php

  /* Connect to MySQL and select the database. */
  $connection = mysqli_connect(DB_SERVER, DB_USERNAME, DB_PASSWORD);

  if (mysqli_connect_errno()) echo "Failed to connect to MySQL: " .
 mysqli_connect_error();

  $database = mysqli_select_db($connection, DB_DATABASE);

  /* Ensure that the EMPLOYEES table exists. */
  VerifyEmployeesTable($connection, DB_DATABASE);

  /* If input fields are populated, add a row to the EMPLOYEES table. */
  $employee_name = htmlentities($_POST['NAME']);
  $employee_address = htmlentities($_POST['ADDRESS']);

  if (strlen($employee_name) || strlen($employee_address)) {
    AddEmployee($connection, $employee_name, $employee_address);
  }
?>

<!-- Input form -->
<form action="<?PHP echo $_SERVER['SCRIPT_NAME'] ?>" method="POST">
  <table border="0">
    <tr>
      <td>NAME</td>
      <td>ADDRESS</td>
    </tr>
    <tr>
      <td>
        <input type="text" name="NAME" maxlength="45" size="30" />
      </td>
      <td>
        <input type="text" name="ADDRESS" maxlength="90" size="60" />
      </td>
      <td>
        <input type="submit" value="Add Data" />
      </td>
    </tr>
  </table>
</form>
```

```
<!-- Display table data. -->
<table border="1" cellpadding="2" cellspacing="2">
  <tr>
    <td>ID</td>
    <td>NAME</td>
    <td>ADDRESS</td>
  </tr>

<?php

$result = mysqli_query($connection, "SELECT * FROM EMPLOYEES");

while($query_data = mysqli_fetch_row($result)) {
  echo "<tr>";
  echo "<td>",$query_data[0], "</td>",
       "<td>",$query_data[1], "</td>",
       "<td>",$query_data[2], "</td>";
  echo "</tr>";
}
?>

</table>

<!-- Clean up. -->
<?php

  mysqli_free_result($result);
  mysqli_close($connection);

?>

</body>
</html>


<?php

/* Add an employee to the table. */
function AddEmployee($connection, $name, $address) {
   $n = mysqli_real_escape_string($connection, $name);
   $a = mysqli_real_escape_string($connection, $address);

   $query = "INSERT INTO EMPLOYEES (NAME, ADDRESS) VALUES ('$n', '$a');";
```

```
   if(!mysqli_query($connection, $query)) echo("<p>Error adding employee data.</
p>");
}

/* Check whether the table exists and, if not, create it. */
function VerifyEmployeesTable($connection, $dbName) {
  if(!TableExists("EMPLOYEES", $connection, $dbName))
  {
     $query = "CREATE TABLE EMPLOYEES (
         ID int(11) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
         NAME VARCHAR(45),
         ADDRESS VARCHAR(90)
       )";

     if(!mysqli_query($connection, $query)) echo("<p>Error creating table.</
p>");
  }
}

/* Check for the existence of a table. */
function TableExists($tableName, $connection, $dbName) {
  $t = mysqli_real_escape_string($connection, $tableName);
  $d = mysqli_real_escape_string($connection, $dbName);

  $checktable = mysqli_query($connection,
      "SELECT TABLE_NAME FROM information_schema.TABLES WHERE TABLE_NAME = '$t'
 AND TABLE_SCHEMA = '$d'");

  if(mysqli_num_rows($checktable) > 0) return true;

  return false;
}
?>
```

PostgreSQL

```
<?php include "../inc/dbinfo.inc"; ?>

<html>
<body>
<h1>Sample page</h1>
```

```
<?php

/* Connect to PostgreSQL and select the database. */
$constring = "host=" . DB_SERVER . " dbname=" . DB_DATABASE . " user=" .
 DB_USERNAME . " password=" . DB_PASSWORD ;
$connection = pg_connect($constring);

if (!$connection){
 echo "Failed to connect to PostgreSQL";
 exit;
}

/* Ensure that the EMPLOYEES table exists. */
VerifyEmployeesTable($connection, DB_DATABASE);

/* If input fields are populated, add a row to the EMPLOYEES table. */
$employee_name = htmlentities($_POST['NAME']);
$employee_address = htmlentities($_POST['ADDRESS']);

if (strlen($employee_name) || strlen($employee_address)) {
  AddEmployee($connection, $employee_name, $employee_address);
}

?>

<!-- Input form -->
<form action="<?PHP echo $_SERVER['SCRIPT_NAME'] ?>" method="POST">
  <table border="0">
    <tr>
      <td>NAME</td>
      <td>ADDRESS</td>
    </tr>
    <tr>
      <td>
    <input type="text" name="NAME" maxlength="45" size="30" />
      </td>
      <td>
    <input type="text" name="ADDRESS" maxlength="90" size="60" />
      </td>
      <td>
    <input type="submit" value="Add Data" />
      </td>
    </tr>
  </table>
```

```
</form>
<!-- Display table data. -->
<table border="1" cellpadding="2" cellspacing="2">
  <tr>
    <td>ID</td>
    <td>NAME</td>
    <td>ADDRESS</td>
  </tr>

<?php

$result = pg_query($connection, "SELECT * FROM EMPLOYEES");

while($query_data = pg_fetch_row($result)) {
  echo "<tr>";
  echo "<td>",$query_data[0], "</td>",
       "<td>",$query_data[1], "</td>",
       "<td>",$query_data[2], "</td>";
  echo "</tr>";
}
?>
</table>

<!-- Clean up. -->
<?php

  pg_free_result($result);
  pg_close($connection);
?>
</body>
</html>


<?php

/* Add an employee to the table. */
function AddEmployee($connection, $name, $address) {
   $n = pg_escape_string($name);
   $a = pg_escape_string($address);
   echo "Forming Query";
   $query = "INSERT INTO EMPLOYEES (NAME, ADDRESS) VALUES ('$n', '$a');";

   if(!pg_query($connection, $query)) echo("<p>Error adding employee data.</
p>");
```

```
}

/* Check whether the table exists and, if not, create it. */
function VerifyEmployeesTable($connection, $dbName) {
  if(!TableExists("EMPLOYEES", $connection, $dbName))
  {
     $query = "CREATE TABLE EMPLOYEES (
         ID serial PRIMARY KEY,
         NAME VARCHAR(45),
         ADDRESS VARCHAR(90)
       )";

     if(!pg_query($connection, $query)) echo("<p>Error creating table.</p>");
  }
}
/* Check for the existence of a table. */
function TableExists($tableName, $connection, $dbName) {
  $t = strtolower(pg_escape_string($tableName)); //table name is case sensitive
  $d = pg_escape_string($dbName); //schema is 'public' instead of 'sample' db
 name so not using that

  $query = "SELECT TABLE_NAME FROM information_schema.TABLES WHERE TABLE_NAME =
 '$t';";
  $checktable = pg_query($connection, $query);

  if (pg_num_rows($checktable) >0) return true;
  return false;

}
?>
```

8. SamplePage.php ファイルを保存して閉じます。
9. ウェブブラウザを開いて http://*EC2 instance endpoint*/SamplePage.php (例: http://ec2-12-345-67-890.us-west-2.compute.amazonaws.com/SamplePage.php) を参照することで、ウェブサーバーが正常に DB クラスターに接続していることを確認します。

SamplePage.php を使用して、DB クラスターにデータを追加できます。これで、追加したデータがこのページに表示されます。データがテーブルに挿入されたことを確認するには、Amazon EC2 インスタンスに MySQL をインストールします。その後、DB クラスターに接続し、テーブルにクエリを実行します。

DB クラスターへの接続についての詳細は、Amazon Aurora DB クラスターへの接続 を参照してください。

DB クラスター の安全性を高めるために、VPC の外部にある出典が DB クラスターに接続できないことを確認します。

ウェブサーバーとデータベースのテストが完了したら、DB クラスターと Amazon EC2 インスタンスを削除する必要があります。

- DB クラスターを削除するには、Aurora DB クラスターと DB インスタンスを削除するの手順に従います。最終的なスナップショットを作成する必要はありません。
- Amazon EC2 インスタンスを終了するには、Amazon EC2 ユーザーガイドの「インスタンスの終了」の手順に従います。

Amazon Aurora チュートリアルおよびサンプルコード

AWS のドキュメントには、一般的な Amazon Aurora のユースケースをガイドするチュートリアルがいくつか含まれています。これらのチュートリアルの多くは、他の AWS のサービスとともに Amazon Aurora を使用する方法を説明しています。加えて、GitHub でサンプルコードにアクセスすることもできます。

> **Note**
>
> その他のチュートリアルについては、AWS データベースブログをご覧ください。トレーニングの詳細については、AWS トレーニングと認定を参照してください。

トピック

このガイドのチュートリアル

このガイドの次のチュートリアルは、Amazon Aurora を使用して一般的なタスクを実行する方法を示しています。

- チュートリアル: DB クラスターで使用する VPC を作成する (IPv4 専用)

  Amazon VPC サービスに基づく仮想プライベートクラウド (VPC) に、DB クラスターを含める方法について説明します。この場合、VPC は同じ VPC 内の Amazon EC2 インスタンスで実行しているウェブサーバーとデータを共有します。
- チュートリアル: DB クラスター用の VPC を作成する (デュアルスタックモード)

  Amazon VPC サービスに基づく仮想プライベートクラウド (VPC) に、DB クラスターを含める方法について説明します。この場合、VPC は同じ VPC 内の Amazon EC2 インスタンスとデータを

共有します。このチュートリアルでは、デュアルスタックモードで実行されているデータベースで動作する VPC を、このシナリオで作成します。

- チュートリアル: ウェブサーバーと Amazon Aurora DB クラスターを作成する

  PHP を使用する Apache ウェブサーバーのインストールと、MySQL データベースの作成を説明します。ウェブサーバーは Amazon Linux を使用して Amazon EC2 インスタンスで実行され、MySQL データベースは と Aurora MySQL DB クラスターです。Amazon EC2 インスタンスと DB クラスターの両方が Amazon VPC で実行されます。

- チュートリアル: DB クラスターのスナップショットから Amazon Aurora DB クラスターを復元する

  DB クラススナップショットから DB クラスターを復元する方法について説明します。

- チュートリアル: タグを使用して、停止する Aurora DB クラスターを指定します

  タグを使用して、停止する Aurora DB クラスターを指定する方法を学習します。

- チュートリアル: Amazon EventBridge を使用して DB インスタンスの状態変化を口グに記録する

  Amazon EventBridge および AWS Lambda を使用して DB インスタンスの状態変更を口グに記録する方法を説明します。

他の AWS ガイドのチュートリアル

他の AWS ガイドの次のチュートリアルは、Amazon Aurora を使用して一般的なタスクを実行する方法を説明しています。

> **Note**
>
> 一部のチュートリアルでは Amazon RDS DB インスタンスを使用しますが、Aurora DB クラスターを使用するように適合させることができます。

- AWS AppSync デベロッパーガイドのチュートリアル: Aurora サーバーレス

  AWS AppSync を使用して、Data API を有効にした Aurora Serverless DB クラスターに対して SQL コマンドを実行するためのデータソースを提供する方法を説明します。AWS AppSync リゾルバーで GraphQL クエリ、ミューテーション、サブスクリプションを使用して、Data API に対して SQL ステートメントを実行できます。

- AWS Secrets Manager ユーザーガイドの「チュートリアル: AWS データベース用のシークレットをローテーションする」

  AWS データベースのシークレットを作成し、スケジュールに従ってローテーションするよう設定します。1 つのローテーションを手動でトリガーし、新しいバージョンのシークレットが引き続きアクセスを提供していることを確認します。
- AWS Elastic Beanstalk デベロッパーガイドの「チュートリアルとサンプル」

  Amazon RDS データベースと AWS Elastic Beanstalk を使用するアプリケーションをデプロイする方法を説明します。
- Amazon Machine Learning デベロッパーガイド の Amazon RDS データベースのデータを使用して Amazon ML データソースを作成する

  MySQL DB インスタンスに格納されているデータから Amazon Machine Learning (Amazon ML) データソースオブジェクトを作成する方法を説明します。
- Amazon QuickSight ユーザーガイド の手動で VPC での Amazon RDS インスタンスへのアクセスを有効にする

  Amazon QuickSight が VPC 内の Amazon RDS DB インスタンスにアクセスできるようにする方法を説明します。

Amazon Aurora PostgreSQL の AWS ワークショップおよびラボコンテンツポータル

以下のワークショップやその他のハンズオンコンテンツのコレクションは、 Amazon Aurora PostgreSQL の機能と能力を理解するのに役立ちます。

- Aurora クラスターの作成

  Amazon Aurora PostgreSQL クラスターを手動で作成する方法について説明します。
- Cloud9 クラウドベース IDE 環境を作成して、データベースに接続する

  Cloud9 を設定して、PostgreSQL データベースを初期化する方法について説明します。
- 高速クローン

  Aurora 高速クローンを作成する方法について説明します。
- クエリプラン管理

クエリプラン管理を使用して、一連のステートメントの実行プランを制御する方法について説明します。

- クラスターキャッシュ管理

  Aurora PostgreSQL のクラスターキャッシュ管理機能について説明します。

- データベースアクティビティストリーミング

  この機能を使用してデータベースアクティビティを監視および監査する方法について説明します。

- Performance Insights の使用

  Performance Insights を使用して DB インスタンスを監視および調整する方法について説明します。

- RDS ツールによるパフォーマンスモニタリング

  AWS と Postgres ツール (Cloudwatch、拡張モニタリング、スロークエリログ、Performance Insights、PostgreSQL カタログビュー) を使用して、パフォーマンスの問題を理解し、データベースのパフォーマンスを改善する方法を特定する方法を説明します。

- リードレプリカの自動スケーリング

  負荷生成スクリプトを使用して、Aurora リードレプリカの自動スケーリングが実際にどのように機能するか説明します。

- 耐障害性のテスト

  DB クラスターがどのように障害に耐えられるかを説明します。

- Aurora グローバルデータベース

  Aurora グローバルデータベースについて説明します。

- 機械学習の使用

  Aurora 機械学習について説明します。

- Aurora Serverless v2

  Aurora Serverless v2 について説明します。

- Trusted Language Extensions for Aurora PostgreSQL

  Aurora PostgreSQL で安全に動作する高性能な拡張機能を構築する方法を学びます。

Amazon Aurora MySQL の AWS ワークショップおよびラボコンテンツポータル

以下のワークショップやその他のハンズオンコンテンツのコレクションは、 Amazon Aurora MySQL の機能と能力を理解するのに役立ちます。

- Aurora クラスターの作成

  Amazon Aurora MySQL クラスターを手動で作成する方法について説明します。
- Cloud9 クラウドベース IDE 環境を作成して、データベースに接続する

  Cloud9 を設定して MySQL データベースを初期化する方法について説明します。
- 高速クローン

  Aurora 高速クローンを作成する方法について説明します。
- クラスターのバックトラック

  DB クラスターをバックトラックする方法について説明します。
- Performance Insights の使用

  Performance Insights を使用して DB インスタンスを監視および調整する方法について説明します。
- RDS ツールによるパフォーマンスモニタリング

  AWS と SQL ツールを使用して、パフォーマンスの問題を理解し、データベースのパフォーマンスを向上させる方法を特定する方法について説明します。
- クエリパフォーマンスの分析

  さまざまなツールを使用して SQL パフォーマンス関連の問題をトラブルシューティングする方法について説明します。
- リードレプリカの自動スケーリング

  リードレプリカの自動スケーリングの仕組みを説明します。
- 耐障害性のテスト

  Aurora MySQL の高可用性と耐障害性の機能について説明します。
- Aurora グローバルデータベース

Aurora グローバルデータベースについて説明します。

- Aurora Serverless v2

  Aurora Serverless v2 について説明します。

- 機械学習の使用

  Aurora 機械学習について説明します。

GitHub のチュートリアルとサンプルコード

GitHub の次のチュートリアルとサンプルコードは、Amazon Aurora を使用して一般的なタスクを実行する方法を示しています。

- Aurora Serverless v2 貸出ライブラリの作成

  常連客が本を借りたり返却したりできる貸出ライブラリアプリケーションの作成方法について説明します。このサンプルでは、Aurora Serverless v2 と AWS SDK for Python (Boto3) を使用しています。

- SDK for Java 2.x を使用して Aurora Serverless v2 データを照会する Spring REST API を備えた Amazon Aurora アイテムトラッカーアプリケーションを作成する

  Aurora Serverless v2 データを照会する Spring REST API を作成する方法について説明します。これは、SDK for Java 2.x を使用する React アプリケーションで使用するためのものです。

- AWS SDK for PHP を使用して Aurora Serverless v2 データを照会する Amazon Aurora アイテムトラッカーアプリケーションを作成する

  Data API の `RdsDataClient` と Aurora Serverless v2 を使用するアプリケーションを作成し、作業項目を追跡して報告する方法について説明します。この例では、AWS SDK for PHP を使用します。

- AWS SDK for Python (Boto3) を使用して Aurora Serverless v2 データを照会する Amazon Aurora アイテムトラッカーアプリケーションを作成する

  Data API の `RdsDataClient` と Aurora Serverless v2 を使用するアプリケーションを作成し、作業項目を追跡して報告する方法について説明します。この例では、AWS SDK for Python (Boto3) を使用します。

このサービスを AWS SDK で使用する

AWS ソフトウェア開発キット (SDK) は、多くの一般的なプログラミング言語で使用できます。各 SDK には、デベロッパーが好みの言語でアプリケーションを簡単に構築できるようにする API、コード例、およびドキュメントが提供されています。

SDK ドキュメント	コードの例
AWS SDK for C++	AWS SDK for C++ コードの例
AWS CLI	AWS CLI コードの例
AWS SDK for Go	AWS SDK for Go コードの例
AWS SDK for Java	AWS SDK for Java コードの例
AWS SDK for JavaScript	AWS SDK for JavaScript コードの例
AWS SDK for Kotlin	AWS SDK for Kotlin コードの例
AWS SDK for .NET	AWS SDK for .NET コードの例
AWS SDK for PHP	AWS SDK for PHP コードの例
AWS Tools for PowerShell	Tools for PowerShell のコード例
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) コードの例
AWS SDK for Ruby	AWS SDK for Ruby コードの例
AWS SDK for Rust	AWS SDK for Rust コードの例
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP コードの例
AWS SDK for Swift	AWS SDK for Swift コードの例

このサービスに固有の例については、「AWS SDK を使用した Aurora のコード例」を参照してください。

可用性の例

必要なものが見つからなかった場合。このページの下側にある [Provide feedback (フィードバックを送信)] リンクから、コードの例をリクエストしてください。

Amazon Aurora DB クラスターの設定

このセクションでは、Aurora DB クラスターの設定方法を示します。Aurora DB クラスターを作成する前に、DB クラスターを実行する DB インスタンスクラスを決定します。また、AWS リージョンを選択して DB クラスターを実行する場所を決定します。次に、DB クラスターを作成します。Aurora 以外のデータがある場合は、データを Aurora DB クラスターに移行できます。

トピック

- Amazon Aurora DB クラスターの作成
- AWS CloudFormation での Amazon Aurora リソースの作成
- Amazon Aurora DB クラスターへの接続
- 「パラメータグループを使用する」
- Amazon Aurora DB クラスターへのデータの移行
- Aurora DB クラスター設定を使用して Amazon ElastiCache キャッシュを作成する

Amazon Aurora DB クラスターの作成

Amazon Aurora DB クラスターは、1 つの DB インスタンス (MySQL または PostgreSQL と互換) および 1 つのクラスターボリューム (3 つのアベイラビリティーゾーンにコピーされた DB クラスターのデータを単一の仮想ボリュームとして保持) で構成されます。デフォルトでは、Aurora DB クラスターには、読み取りと書き込みを実行するプライマリ DB インスタンスと、オプションで最大 15 個の Aurora レプリカ (リーダー DB インスタンス) が含まれます。Aurora DB クラスターの詳細については、「Amazon Aurora DB クラスター」を参照してください。

Aurora では、2 つの主要なタイプの DB クラスターがあります。

- Aurora プロビジョニング — 予想されるワークロードに基づいて、ライターインスタンスとリーダーインスタンスの DB インスタンスクラスを選択します。詳細については、「Aurora DB インスタンスクラス」を参照してください。プロビジョニングされた Aurora には、Aurora グローバルデータベースを含むいくつかのオプションがあります。詳細については、「Amazon Aurora Global Database の使用」を参照してください。
- Aurora Serverless – Aurora Serverless v1 および Aurora Serverless v2 は、Aurora 用のオンデマンドの自動スケーリング設定です。容量は、アプリケーションの需要に応じて自動的に調整されます。DB クラスターが消費するリソースに対してのみ課金されます。このオートメーションは、変動が大きく予測不可能なワークロードがある環境で特に役立ちます。詳細については、Amazon Aurora Serverless v1 の使用およびAurora Serverless v2 を使用するを参照してください。

次に、Aurora DB クラスターを作成する方法について説明します。スタートするには、まず「DB クラスターの前提条件」を参照してください。

Aurora DB クラスターへの接続については、「Amazon Aurora DB クラスターへの接続」を参照してください。

目次

- Aurora DB クラスターの設定
- DB クラスター用 Amazon Aurora に適用されない設定
- Amazon Aurora DB インスタンスには適用されない設定

DB クラスターの前提条件

> **Important**
>
> Aurora DB クラスターを作成する前に、「Amazon Aurora の環境をセットアップする」のタスクを完了する必要があります。

DB クラスターを作成する前に完了しておく必要がある前提条件を次に示します。

トピック

- DB クラスターのネットワークを設定する
- 追加の前提条件

DB クラスターのネットワークを設定する

Amazon Aurora DB クラスターは、Amazon VPC サービスに基づき、最低 2 つのアベイラビリティーゾーンのある AWS リージョンの 仮想プライベートクラウド (VPC) 内にのみ作成できます。DB クラスターに選択する DB サブネットグループは、少なくとも 2 つのアベイラビリティーゾーンを対象とする必要があります。この設定により、万一アベイラビリティーゾーンに障害が発生した場合でも、DB クラスターにはフェイルオーバーに使用できる DB インスタンスが常に 1 つ以上存在することが保証されます。

新しい DB クラスターと同じ VPC 内の EC2 インスタンス間の接続をセットアップする予定がある場合、DB クラスターの作成時に設定できます。同じ VPC 内の EC2 インスタンス以外のリソースから DB クラスターに接続する予定がある場合、ネットワーク接続を手動で設定できます。

トピック

- EC2 インスタンスとの自動ネットワーク接続を設定する
- ネットワークを手動で設定する

EC2 インスタンスとの自動ネットワーク接続を設定する

Aurora DB クラスターを作成する場合は、AWS Management Console を使用して Amazon EC2 インスタンスと新しい DB クラスター間の接続をセットアップできます。これを行うと、RDS では VPC とネットワークの設定を自動で行います。EC2 インスタンスが DB クラスターにアクセスできるように、EC2 インスタンスと同じ VPC 内に DB クラスターを作成します。

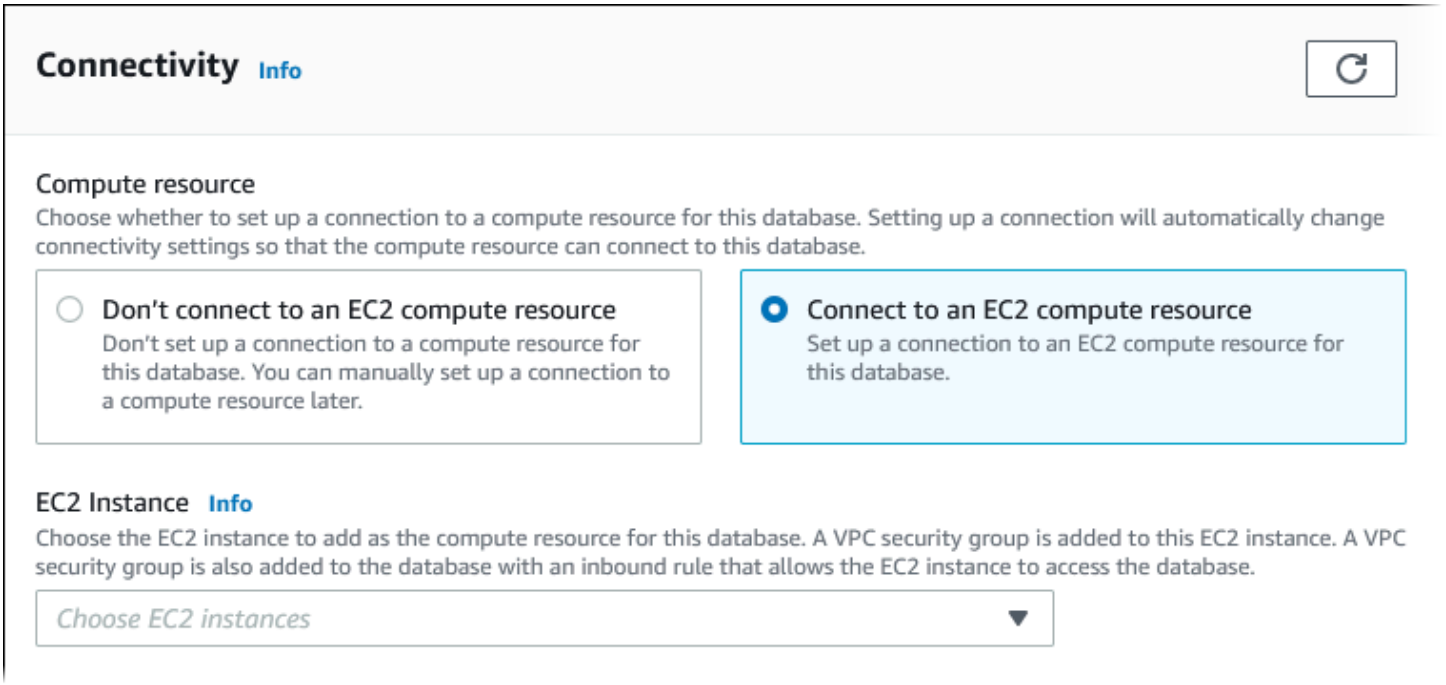
EC2 インスタンスと DB クラスターを接続するための要件は次のとおりです。

- DB クラスターを作成する前に、AWS リージョン に EC2 インスタンスが存在する必要があります。

  AWS リージョン に EC2 インスタンスが存在しない場合、コンソールには EC2 インスタンス作成用のリンクが表示されます。
- 現在、DB クラスターを Aurora Serverless DB クラスターまたは Aurora グローバルデータベースの一部にすることはできません。
- DB インスタンスを作成するユーザーには、次の操作を実行する権限が必要です。
  - `ec2:AssociateRouteTable`
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateRouteTable`
  - `ec2:CreateSubnet`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeRouteTables`
  - `ec2:DescribeSecurityGroups`
  - `ec2:DescribeSubnets`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

このオプションを使用すると、プライベート DB クラスターが作成されます。DB クラスターでは、プライベートサブネットのみを持つ DB サブネットグループを使用して、VPC 内のリソースへのアクセスを制限します。

EC2 インスタンスを DB クラスターに接続するには、[Create database] (データベースの作成) ページの [Connectivity] (接続) セクションで、[Connect to an EC2 compute resource] (EC2 コンピューティングリソースに接続する) を選択します。



[Connect to an EC2 compute resource] (EC2 コンピューティングリソースに接続する) を選択すると、RDS では次のオプションを自動的に設定します。[Don't connect to an EC2 compute resource] (EC2 コンピューティングリソースに接続しない) を選択して EC2 インスタンスとの接続をセットアップしない限り、これらの設定は変更できません。

コンソールオプション	自動ログ記録
ネットワークの種類	RDS はネットワークタイプを IPv4 に設定します。EC2 インスタンスと DB クラスター間の接続をセットアップする場合、デュアルスタックモードは現在サポートされていません。
仮想プライベートクラウド (VPC)	RDS は EC2 インスタンスに関連付けられている VPC に設定します。
DB サブネットグループ	RDS では、EC2 インスタンスと同じアベイラビリティーゾーンにプライベートサブネットを持つ DB サブネットグループが必要です。この要件を満たす DB サブネットグループが存在する場合、RDS は既存の DB サブネットグループを使用します。デ

コンソールオプション	自動ログ記録
	フォルトでは、このオプションは [Automatic setup] (自動セットアップ) に設定されています。 [Automatic setup] (自動セットアップ) を選択したとき、この要件を満たす DB サブネットグループがない場合、次のアクションが実行されます。RDS は 3 つのアベイラビリティーゾーンで 3 つの使用可能なプライベートサブネットを使用します。アベイラビリティーゾーンのうちの 1 つは EC2 インスタンスと同じです。プライベートサブネットがアベイラビリティーゾーンで使用できない場合、RDS はアベイラビリティーゾーンにプライベートサブネットを作成します。次に、RDS は DB サブネットグループを作成します。 プライベートサブネットが使用可能な場合、RDS はサブネットに関連付けられているルートテーブルを使用して、作成したサブネットをこのルートテーブルに追加します。プライベートサブネットが使用できない場合、RDS はインターネットゲートウェイにアクセスできないルートテーブルを作成し、作成したサブネットをルートテーブルに追加します。 RDS では、既存の DB サブネットグループを使用することもできます。既存の DB サブネットグループを使用する場合は、[Choose existing] (既存を選択) を選択します。
パブリックアクセス	RDS では [No] (いいえ) を選択して、DB クラスターがパブリックアクセス可能にならないようにします。 セキュリティ上の理由から、データベースは非公開として、インターネットからアクセスできないようにするのがベストプラクティスです。

コンソールオプション	自動ログ記録
VPC セキュリティグループ (ファイアウォール)	RDS では DB クラスターに関連付けられている新しいセキュリティグループを作成します。セキュリティグループの名前は `rds-ec2-`*n* で、*n* は数字です。このセキュリティグループには、EC2 VPC セキュリティグループ (ファイアウォール) をソースとするインバウンドルールが含まれています。DB クラスターに関連付けられているこのセキュリティグループにより、EC2 インスタンスが DB クラスターにアクセスできます。 また、RDS では EC2 インスタンス関連付けられている新しいセキュリティグループを作成します。セキュリティグループの名前は `ec2-rds-`*n* で、*n* は数字です。このセキュリティグループには、DB クラスターの VPC セキュリティグループをソースとするアウトバウンドルールが含まれています。このセキュリティグループにより、EC2 インスタンスは DB クラスターにトラフィックを送信できます。 [Create new] (新規作成) を選択して、新しいセキュリティグループの名前を入力すると、別のセキュリティグループを新規に追加できます。 既存のセキュリティグループを追加するには、[Choose existing] (既存を選択) を選択し、追加するセキュリティグループを選択します。

コンソールオプション	自動ログ記録
アベイラビリティーゾーン	DB クラスターの作成時に Availability & durability (可用性と耐久性) で Aurora レプリカを作成しない場合 (シングル AZ 配置) は、RDS では EC2 インスタンスのアベイラビリティーゾーンを選択します。 DB クラスターの作成時に Aurora レプリカを作成する場合 (マルチ AZ 配置) は、RDS では DB クラスター内の 1 つの DB インスタンスの EC2 インスタンスのアベイラビリティーゾーンを選択します。RDS は DB クラスター内の他の DB インスタンスに対し、異なるアベイラビリティーゾーンをランダムに選択します。プライマリ DB インスタンスまたは Aurora レプリカのいずれかが、EC2 インスタンスと同じアベイラビリティーゾーンに作成されます。プライマリ DB インスタンスと EC2 インスタンスが異なるアベイラビリティーゾーンにある場合、アベイラビリティーゾーン間のコストが発生する可能性があります。

これらの設定の詳細については、「Aurora DB クラスターの設定」をご参照ください。

DB クラスターの作成後にこれらの設定を変更すると、EC2 インスタンスと DB クラスター間の接続に影響する可能性があります。

ネットワークを手動で設定する

同じ VPC 内の EC2 インスタンス以外のリソースから DB クラスターに接続する予定がある場合、ネットワーク接続を手動で設定できます。AWS Management Console を使用して DB クラスターを作成する場合は、お客様に代わって Amazon RDS に VPC を自動的に作成させることができます。または、既存の VPC を使うか、Aurora DB クラスター用に新しい VPC を作成することができます。どの方法を使用する場合でも、VPC を Amazon Aurora DB クラスターで使用するには、少なくとも 2 つのアベイラビリティーゾーンのそれぞれに 1 つ以上のサブネットが必要です。

デフォルトでは、Amazon RDS は アベイラビリティーゾーンにプライマリ DB インスタンスと Aurora レプリカを自動的に作成します。特定のアベイラビリティーゾーンを選択するには、[Availability & durability] (可用性と耐久性) マルチ AZ 配置設定を [Don't create an Aurora Replica] (Aurora レプリカを作成しない) に変更する必要があります。これにより、アベイラビリティーゾーンの設定が表示され、VPC 内のアベイラビリティーゾーンの中から選択できます。ただし、デフォルト設定のままで、Amazon RDS によってマルチ AZ 配置を作成し、アベイラ

ビリティーゾーンを選択することを強くお勧めします。そうすることで、Aurora DB クラスターは、Aurora の 2 つの主な利点であるフェイルオーバーが速く可用性が高いという特徴を生かして作成されます。

デフォルト VPC を持っていない、または VPC を作成していない場合は、コンソールを使用して DB クラスターを作成する際に、お客様に代わって Amazon RDS に VPC を自動的に作成させることができます。それ以外の場合は、以下を実行する必要があります。

- DB クラスターをデプロイする AWS リージョンで、少なくとも 2 つのアベイラビリティーゾーンのそれぞれに 1 つ以上のサブネットを持つ VPC を作成します。詳細については、VPC 内の DB クラスターの使用およびチュートリアル: DB クラスターで使用する VPC を作成する (IPv4 専用)を参照してください。
- DB クラスターへの接続を許可する VPC セキュリティグループを指定します。詳細については、セキュリティグループを作成して VPC 内の DB クラスターへのアクセスを提供するおよびセキュリティグループによるアクセス制御を参照してください。
- DB クラスターが使用できる VPC 内の最低 2 つのサブネットを定義する RDS DB サブネットグループを指定します。詳細については、「DB サブネットグループの使用」を参照してください。

VPC の詳細については、「Amazon VPC VPC とAmazon Aurora」を参照してください。プライベート DB クラスターのネットワークを設定するチュートリアルについては、「チュートリアル: DB クラスターで使用する VPC を作成する (IPv4 専用)」を参照してください。

Aurora DB クラスターと同じ VPC にないリソースに接続する場合は、VPC の DB クラスターにアクセスするシナリオ の該当するシナリオを参照してください。

追加の前提条件

DB クラスターを作成する前に、以下に示す追加の前提条件を検討してください。

- AWS Identity and Access Management IAM 認証情報を使用して AWS に接続している場合、AWS アカウントに Amazon RDS オペレーションを実行するためのアクセス許可を付与する IAM ポリシーが必要です。詳細については、「Amazon Aurora での Identity and Access Management」を参照してください。

  IAM を使用して Amazon RDS コンソールにアクセスする場合は、まず、ユーザー認証情報で AWS Management Console にサインオンする必要があります。次に、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) に移動します。

- DB クラスターの設定パラメータを調整する場合は、必要なパラメータ設定を持つ DB クラスターパラメータグループと DB パラメータグループを指定する必要があります。DB クラスターパラメータグループまたは DB パラメータグループの作成または変更については、「「パラメータグループを使用する」」を参照してください。
- DB クラスター用に指定する TCP/IP ポート番号を確認します。会社のファイアウォールによっては、Aurora のデフォルトポート (MySQL の場合は 3306、PostgreSQL の場合は 5432) への接続がブロックされる場合があります。会社のファイアウォールがデフォルトのポートをブロックする場合は、お客様の DB クラスター用に別のポートを選択します。DB クラスターのすべてのインスタンスは同じポートを使用します。
- データベースのメジャーエンジンバージョンが RDS 標準サポート終了日に達した場合は、延長サポート CLI オプションまたは RDS API パラメータを使用する必要があります。詳細については、「Aurora DB クラスターの設定」の「RDS 延長サポート」を参照してください。

DB クラスターの作成

Aurora DB クラスターは、AWS Management Console、AWS CLI、または RDS API を使用して作成できます。

コンソール

[簡易作成] を有効または無効にし、AWS Management Console を使用して DB クラスターを作成できます。[Easy create] を有効にして、DB エンジンタイプ、DB インスタンスサイズ、および DB インスタンス識別子のみを指定します。[Easy create] では、他の設定オプションにデフォルト設定を使用します。[Easy create] が有効になっていない場合は、データベースの作成時に、可用性、セキュリティ、バックアップ、メンテナンスなどの設定オプションを追加指定します。

> **Note**
>
> この例では、[Standard Create (スタンダード作成)] が有効になっており、[Easy Create (簡易作成)] は有効になっていません。[簡易作成] を有効にして、DB クラスターを作成する方法については、「Amazon Aurora の開始方法」を参照してください。

コンソールを使用して Aurora DB クラスターを作成するには

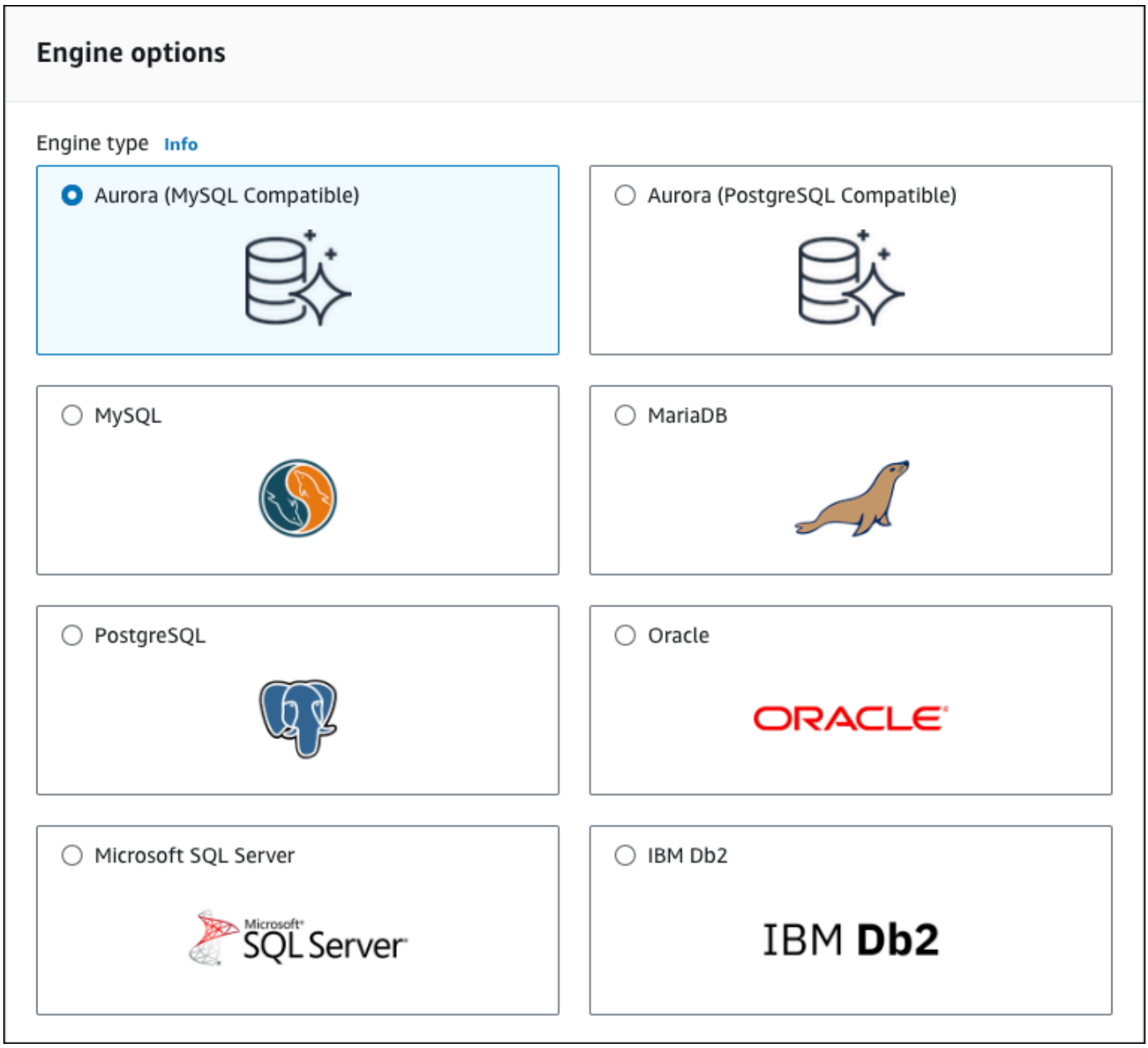
1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。

2. AWS Management Console の右上で、DB クラスターを作成する AWS リージョンを選択します。

   Aurora は、一部の AWS リージョンでは使用できません。Aurora を使用できる AWS リージョンのリストについては、「利用可能なリージョン」を参照してください。

3. ナビゲーションペインで、[データベース] を選択します。
4. [データベースの作成] を選択します。
5. [データベース作成方法を選択] で [標準作成] を選択します。
6. [エンジンタイプ] で、以下のいずれかを選択します。

   - [Aurora (MySQL 互換)]
   - [Aurora (PostgreSQL 互換)]



7. [エンジンバージョン] を選択します。

   詳細については、「Amazon Aurora バージョン」を参照してください。フィルターを使用して、Aurora Serverless v2 など、必要な機能と互換性のあるバージョンを選択できます。詳細については、「Aurora Serverless v2 を使用する」を参照してください。

8. [テンプレート] で、ユースケースに合うテンプレートを選択します。

9. マスターパスワードを入力するには、以下の操作を行います。

   a. [設定] セクションで、[認証情報の設定] を展開します。

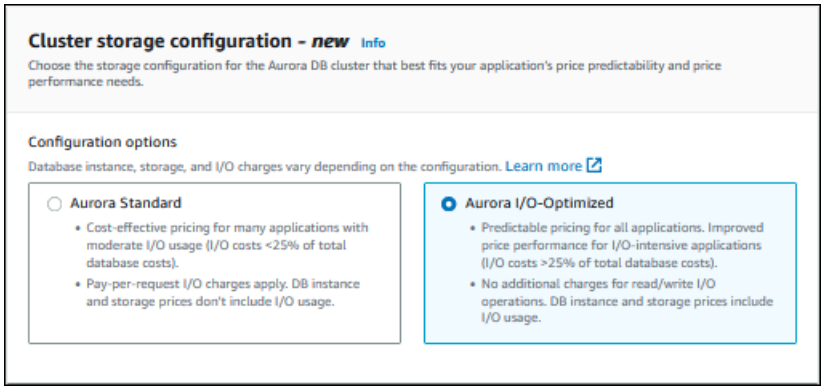
b. [Auto generate a password (パスワードの自動生成)] チェックボックスをオフにします。

c. (オプション) [Master username] 値を変更し、[Master password] および [Confirm password] に同じパスワードを入力します。

デフォルトでは、新しい DB インスタンスはマスターユーザー用に自動生成されたパスワードを使用します。

10. [VPC セキュリティグループ (ファイアウォール)] の [接続] セクションで [新規作成] を選択すると、ローカルコンピュータの IP アドレスにデータベースへのアクセスを許可するインバウンドルールを使用して VPC セキュリティグループが作成されます。

11. クラスターストレージ構成に対して、Aurora I/O-Optimized または Aurora Standard のいずれかを選択します。詳細については、「Amazon Aurora DB クラスターのストレージ設定」を参照してください。



12. (オプション) この DB クラスターのコンピューティングリソースへの接続を設定します。

DB クラスターの作成時に、Amazon EC2 インスタンスと新しい DB クラスター間の接続を設定できます。詳細については、「EC2 インスタンスとの自動ネットワーク接続を設定する」を参照してください。

13. 残りのセクションで、DB クラスター設定を指定します。各設定の詳細については、「Aurora DB クラスターの設定」を参照してください。

14. [データベースの作成] を選択します。

自動生成されたパスワードを使用することを選択した場合は、[データベース] ページに [認証情報の詳細の表示] ボタンが表示されます。

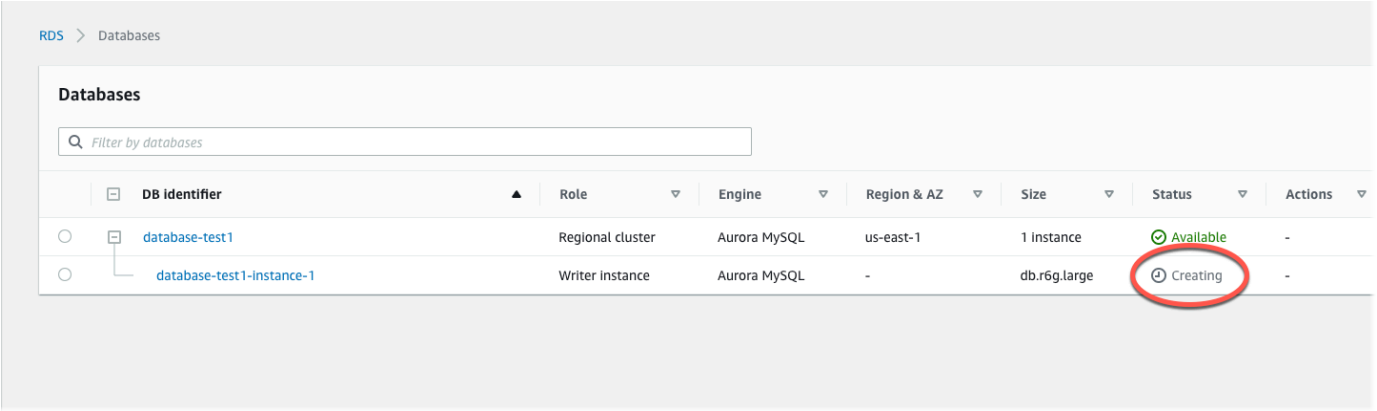
DB クラスターのマスターユーザー名およびパスワードを表示するには、[認証情報の詳細の表示] を選択します。

マスターユーザーとして DB インスタンスに接続するには、表示されているユーザー名およびパスワードを使用します。

> **Important**
>
> マスターユーザーのパスワードを再度表示することはできません。記録していない場合は、変更する必要がある場合があります。DB インスタンスが有効になった後にマスターユーザーのパスワードを変更する必要がある場合は、そのように DB インスタンスを変更することができます。DB インスタンスの変更の詳細については、「Amazon Aurora DB クラスターの変更」を参照してください。

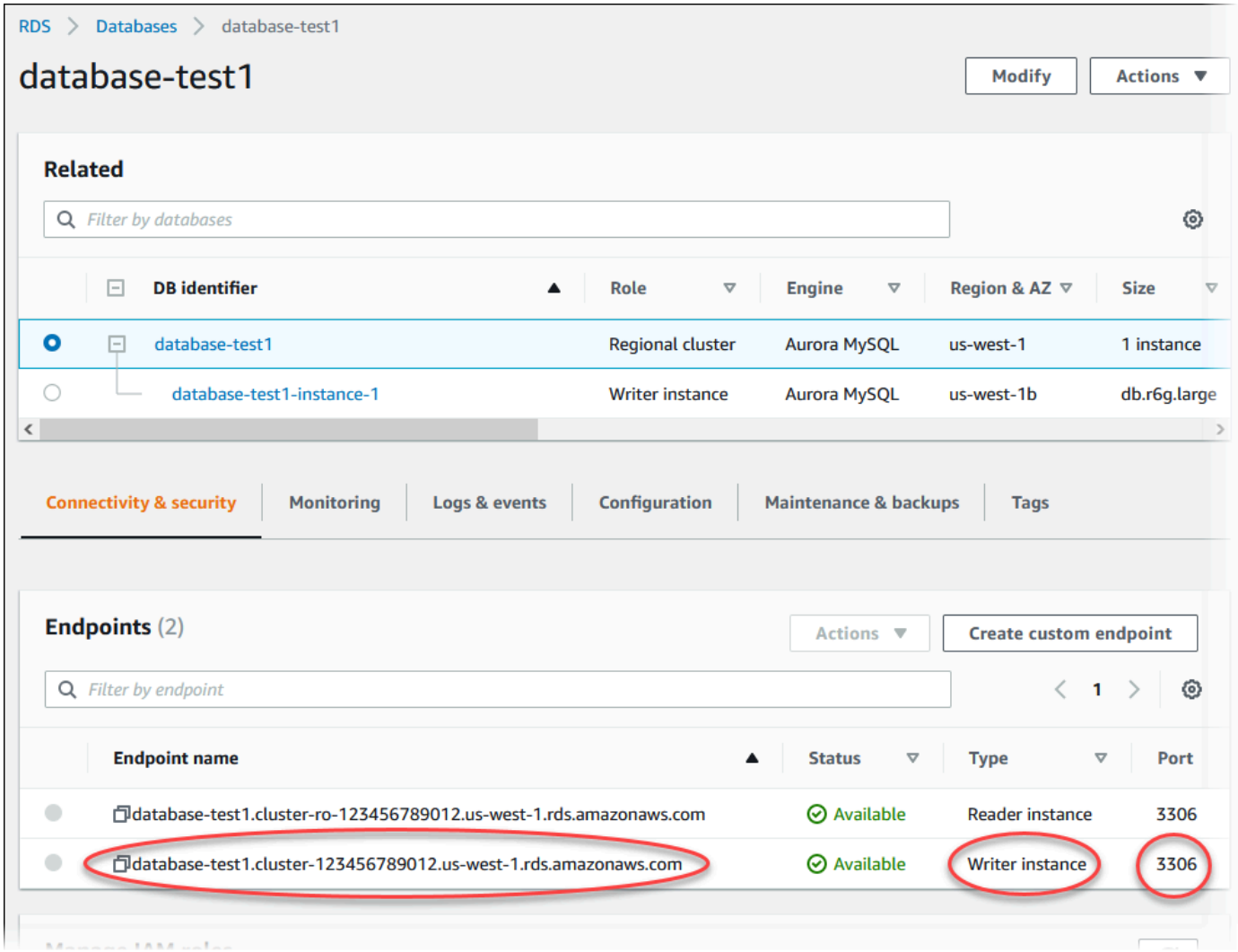
15. [データベース] で、新しい Aurora DB クラスターの名前を選択します。

RDS コンソールに、新しい DB クラスターの詳細が表示されます。DB クラスターが使用できるようになるまで、DB クラスターおよび DB インスタンスのステータスは [creating (作成中)] になります。



両方のステータスが [available] に変わると、DB クラスターに接続できます。DB インスタンスクラスとストレージの合計によっては、新しい DB クラスターを使用できるようになるまで最長 20 分かかることがあります。

新しく作成したクラスターを確認するには、Amazon RDS コンソールのナビゲーションペインから [データベース] を選択します。DB クラスターの詳細を表示する DB クラスターを選択します。詳細については、「Amazon Aurora DB クラスターの表示」を参照してください。



[接続とセキュリティ] タブで、書き込み DB インスタンスのポートおよびエンドポイントを書き留めます。クラスターのエンドポイントとポートは、書き込みまたは読み取りオペレーションを実行するすべてのアプリケーション用の JDBC 接続文字列と ODBC 接続文字列で使用します。

AWS CLI

> **Note**
>
> AWS CLI を使用して Aurora DB クラスターを作成する前に、VPC や RDS DB サブネットグループの作成など、必要な前提条件を満たす必要があります。詳細については、「DB クラスターの前提条件」を参照してください。

AWS CLI を使用して、Aurora MySQL DB クラスター、または Aurora PostgreSQL DB クラスターを作成できます。

AWS CLI を使用して Aurora MySQL DB クラスターを作成するには

Aurora MySQL 8.0 互換または 5.7 互換のDB クラスターまたは DB インスタンスを作成するときは、`--engine`オプションに`aurora-mysql`を指定する必要があります。

以下のステップを実行します。

1. 新しい DB クラスターの DB サブネットグループと VPC セキュリティグループ ID を認証し、AWS CLI の create-db-cluster コマンドを呼び出して Aurora MySQL DB クラスターを作成します。

   例えば、次のコマンドは `sample-cluster` という名前の新しい MySQL 8.0 互換 DB クラスターを作成します。クラスターは、デフォルトのエンジンバージョンと Aurora I/O-Optimized ストレージタイプを使用します。

   Linux、macOS、Unix の場合:

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster \
       --engine aurora-mysql --engine-version 8.0 \
       --storage-type aurora-iopt1 \
       --master-username user-name --manage-master-user-password \
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

   Windows の場合:

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
       --engine aurora-mysql --engine-version 8.0 ^
       --storage-type aurora-iopt1 ^
       --master-username user-name --manage-master-user-password ^
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

   次のコマンドは `sample-cluster` という名前の新しい MySQL 5.7 互換 DB クラスターを作成します。クラスターは、デフォルトのエンジンバージョンと Aurora Standard ストレージタイプを使用します。

   Linux、macOS、Unix の場合:

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster  \
   ```

```
    --engine aurora-mysql --engine-version 5.7 \
    --storage-type aurora \
    --master-username user-name --manage-master-user-password \
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

Windows の場合:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster sample-cluster  ^
    --engine aurora-mysql --engine-version 5.7 ^
    --storage-type aurora ^
    --master-username user-name --manage-master-user-password ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

2. コンソールを使用して DB クラスターを作成する場合、Amazon RDS は自動的に使用する DB クラスターのプライマリインスタンス (ライター) を作成します。AWS CLI を使用して DB クラスターを作成する場合、使用する DB クラスターのプライマリインスタンスを明示的に作成する必要があります。プライマリインスタンスは、DB クラスターで作成される初期の DB インスタンスです。プライマリ DB インスタンスを作成しない限り、DB クラスターエンドポイントは `Creating` ステータスのままです。

   DB クラスターのプライマリインスタンスを作成するには、create-db-instance AWS CLI コマンドを呼び出します。`--db-cluster-identifier` オプション値として DB クラスターの名前を含めます。

   > **Note**
   >
   > DB インスタンスに対して `--storage-type` オプションは設定できません。DB クラスターにのみ設定できます。

   例えば、次のコマンドは `sample-instance` という名前の新しい MySQL 5.7 互換または MySQL 8.0 互換の DB インスタンスを作成します。

   Linux、macOS、Unix の場合:

```
aws rds create-db-instance --db-instance-identifier sample-instance \
     --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-
class db.r5.large
```

Windows の場合:

```
aws rds create-db-instance --db-instance-identifier sample-instance ^
     --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-
class db.r5.large
```

AWS CLI を使用して Aurora PostgreSQL DB クラスターを作成するには

1. 新しい DB クラスターの DB サブネットグループと VPC セキュリティグループ ID を認証し、AWS CLI の create-db-cluster コマンドを呼び出して Aurora PostgreSQL DB クラスターを作成します。

   例えば、次のコマンドは `sample-cluster` という名前の新しい DB クラスターを作成します。クラスターは、デフォルトのエンジンバージョンと Aurora I/O-Optimized ストレージタイプを使用します。

   Linux、macOS、Unix の場合:

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster \
       --engine aurora-postgresql \
       --storage-type aurora-iopt1 \
       --master-username user-name --manage-master-user-password \
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

   Windows の場合:

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
       --engine aurora-postgresql ^
       --storage-type aurora-iopt1 ^
       --master-username user-name --manage-master-user-password ^
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

2. コンソールを使用して DB クラスターを作成する場合、Amazon RDS は自動的に使用する DB クラスターのプライマリインスタンス (ライター) を作成します。AWS CLI を使用して DB クラスターを作成する場合、使用する DB クラスターのプライマリインスタンスを明示的に作成する必要があります。プライマリインスタンスは、DB クラスターで作成される初期の DB インスタンスです。プライマリ DB インスタンスを作成しない限り、DB クラスターエンドポイントは `Creating` ステータスのままです。

DB クラスターのプライマリインスタンスを作成するには、create-db-instance AWS CLI コマンドを呼び出します。--db-cluster-identifier オプション値として DB クラスターの名前を含めます。

Linux、macOS、Unix の場合:

```
aws rds create-db-instance --db-instance-identifier sample-instance \
     --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-
instance-class db.r5.large
```

Windows の場合:

```
aws rds create-db-instance --db-instance-identifier sample-instance ^
     --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-
instance-class db.r5.large
```

これらの例では、--manage-master-user-password オプションを指定して、マスターユーザーパスワードを生成し、Secrets Manager で管理します。詳細については、「Amazon Aurora および AWS Secrets Manager によるパスワード管理」を参照してください。または、--master-password オプションを使用して、自分でパスワードを指定して管理することもできます。

RDS API

> **Note**
>
> AWS CLI を使用して Aurora DB クラスターを作成する前に、VPC や RDS DB サブネットグループの作成など、必要な前提条件を満たす必要があります。詳細については、「DB クラスターの前提条件」を参照してください。

新しい DB クラスターの DB サブネットグループと VPC セキュリティグループ ID を識別し、CreateDBCluster オペレーションを呼び出して DB クラスターを作成します。

Aurora MySQL バージョン 2 または 3 の DB クラスターまたは DB インスタンスを作成するときには、Engine パラメータとして aurora-mysql を指定してください。

Aurora PostgreSQL DB クラスターまたは DB インスタンスを作成するときは、aurora-postgresql パラメータに Engine を指定します。

コンソールを使用して DB クラスターを作成する場合、Amazon RDS は自動的に使用する DB クラスターのプライマリインスタンス (ライター) を作成します。RDS API を使用して DB クラスターを作成する場合は、CreateDBInstance を使って DB クラスターのプライマリインスタンスを明示的に作成する必要があります。プライマリインスタンスは、DB クラスターで作成される初期の DB インスタンスです。プライマリ DB インスタンスを作成しない限り、DB クラスターエンドポイントは `Creating` ステータスのままです。

プライマリ (ライター) DB インスタンスの作成

AWS Management Console を使用して DB クラスターを作成する場合、Amazon RDS は自動的に DB クラスターのプライマリインスタンス (ライター) を作成します。AWS CLI または RDS API を使用して DB クラスターを作成する場合、DB クラスターのプライマリインスタンスを明示的に作成する必要があります。プライマリインスタンスは、DB クラスターで作成される初期の DB インスタンスです。プライマリ DB インスタンスを作成しない限り、DB クラスターエンドポイントは `Creating` ステータスのままです。

詳細については、「DB クラスターの作成」を参照してください。

> **Note**
>
> ヘッドレスクラスターとも呼ばれるライター DB インスタンスのない DB クラスターがある場合、コンソールを使用してライターインスタンスを作成することはできません。これには、AWS CLI または RDS API を使用します。

次の例では、create-db-instance AWS CLI コマンドを使用して、`headless-test` という名前の Aurora PostgreSQL DB クラスターのライターインスタンスを作成します。

```
aws rds create-db-instance \
    --db-instance-identifier no-longer-headless \
    --db-cluster-identifier headless-test \
    --engine aurora-postgresql \
    --db-instance-class db.t4g.medium
```

Aurora DB クラスターの設定

次の表は、Aurora DB クラスターの作成時に選択する設定の詳細を示しています。

> **Note**
>
> Aurora Serverless v1 DB クラスターを作成する場合は、追加の設定を使用できます。これらの設定については、「Aurora Serverless v1 DB クラスターの作成」を参照してください。また、Aurora Serverless v1 の制限により、Aurora Serverless v1 では一部の設定を使用できません。詳細については、「Aurora Serverless v1 の制約事項」を参照してください。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
マイナーバージョン自動アップグレード	DB エンジンの指定したマイナーバージョンアップグレードをリリースと同時に Aurora DB クラスターに自動的に適用する場合は、[Enable auto minor version upgrade (マイナーバージョン自動アップグレードの有効化)] を選択します。 [マイナーバージョン自動アップグレード] の設定は Aurora PostgreSQL および Aurora MySQL DB クラスターの両方に適用されます。 Aurora PostgreSQL のエンジンに関するアップデートの詳細については、「Amazon Aurora PostgreSQL の更新」を参照してください。 Aurora MySQL のエンジンに関する更新の詳細については、「Amazon Aurora MySQL のデータベースエンジンの更新」を参照してください。	Aurora クラスター内のすべての DB インスタンスに対して、この値を設定します。この設定がオフになっている DB インスタンスがクラスター内にある場合、クラスターは自動アップグレードされません。 AWS CLI を使用して、`create-db-instance` を実行し、`--auto-minor-version-upgrade|--no-auto-minor-version-upgrade` オプションを設定します。 RDS API を使用して、`CreateDBInstance` を呼び出し、`AutoMinorVersionUpgrade` パラメータを設定します。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
AWS KMS key	[Encryption] が [Enable encryption] に設定されている場合にのみ使用できます。この DB クラスターの暗号化に使用する AWS KMS key を選択します。詳細については、「Amazon Aurora リソースの暗号化」を参照してください。	AWS CLI を使用して、create-db-cluster を実行し、--kms-key-id オプションを設定します。 RDS API を使用して、CreateDBCluster を呼び出し、KmsKeyId パラメータを設定します。
バックトラック	Aurora MySQL にのみ適用されます。バックトラックを有効にするには [バックトラックを有効にする] を選択し、バックトラックを無効にするには [バックトラックを無効にする] を選択します。バックトラックを使用して、新しい DB クラスターを作成せずに、特定時点に DB クラスターを巻き戻すことができます。デフォルトでは無効となっています。バックトラッキングを有効にする場合は、DB クラスターをバックトラックできる時間 (ターゲットのバックトラックウィンドウ) も指定します。詳細については、「Aurora DB クラスターのバックトラック」を参照してください。	AWS CLI を使用して、create-db-cluster を実行し、--backtrack-window オプションを設定します。 RDS API を使用して、CreateDBCluster を呼び出し、BacktrackWindow パラメータを設定します。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
認証局	DB クラスターの DB インスタンスによって使用されるサーバー証明書の認定機関 (CA)。 詳細については、「SSL/TLS を使用した DB クラスターへの接続の暗号化」を参照してください。	AWS CLI を使用して、create-db-instance を実行し、--ca-certificate-identifier オプションを設定します。 RDS API を使用して、CreateDBInstance を呼び出し、CACertificateIdentifier パラメータを設定します。
クラスターのストレージ設定	DB クラスターのストレージタイプ: Aurora I/O-Optimized または Aurora Standard。 詳細については、「Amazon Aurora DB クラスターのストレージ設定」を参照してください。	AWS CLI を使用して、create-db-cluster を実行し、--storage-type オプションを設定します。 RDS API を使用して、CreateDBCluster を呼び出し、StorageType パラメータを設定します。
Copy tags to snapshots	スナップショットの作成時に DB インスタンスタグを DB スナップショットへコピーするには、このオプションを選択します。 詳細については、「Amazon RDS リソースのタグ付け」を参照してください。	AWS CLI を使用して、create-db-cluster を実行し、--copy-tags-to-snapshot | --no-copy-tags-to-snapshot オプションを設定します。 RDS API を使用して、CreateDBCluster を呼び出し、CopyTagsToSnapshot パラメータを設定します。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
データベース認証	使用するデータベース認証。 MySQL の場合: • データベースパスワードのみを使用してデータベースのユーザーを認証するには、[パスワード認証] を選択します。 • IAM ユーザーおよびルールでデータベースパスワードとユーザー認証情報を使用してデータベースユーザーを認証するには、[パスワードと IAM データベース認証] を選択します。詳細については、「 の IAM データベース認証」を参照してください。 PostgreSQL の場合: • ユーザーおよびロールでデータベースパスワードとユーザー認証情報を使用してデータベースユーザーを認証するには、[IAM database authentication] (IAM データベース認証) を選択します。詳細については、「 の IAM データベース認証」を参照してください。 • Kerberos 認証を使用してデータベースパスワードとユーザー認証情報を認証するには、[Kerberos 認証] を選択します。詳細については、「Aurora	AWS CLI で IAM データベース認証を使用するには、create-db-cluster を実行し、--enable-iam-database-authentication | --no-enable-iam-database-authentication オプションを設定します。 RDS API で IAM データベース認証を使用するには、CreateDBCluster を呼び出し、EnableIAMDatabaseAuthentication パラメータを設定します。 AWS CLI で Kerberos 認証を使用するには、create-db-cluster を実行し、--domain および --domain-iam-role-name オプションを設定します。 RDS API で Kerberos 認証を使用するには、CreateDBCluster を呼び出し、Domain および DomainIAMRoleName パラメータを設定します。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
	PostgreSQL で Kerberos 認証を使用する」を参照してください。	
データベースポート	データベースのアクセスに使用するために、アプリケーションやユーティリティのポートを指定します。デフォルトでは、Aurora MySQL DB クラスターはデフォルトの MySQL ポート 3306 に設定され、Aurora PostgreSQL DB クラスターはデフォルトの PostgreSQL である 5432 に設定されます。会社のファイアウォールによっては、これらのデフォルトポートへの接続がブロックされます。会社のファイアウォールがデフォルトのポートをブロックする場合は、新しい DB クラスター用に別のポートを選択します。	AWS CLI を使用して、`create-db-cluster` を実行し、`--port` オプションを設定します。 RDS API を使用して、`CreateDBCluster` を呼び出し、`Port` パラメータを設定します。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
DB クラスター識別子	DB クラスターの名前を入力します。この名前は選択した AWS リージョン内で、アカウントに対して一意である必要があります。この識別子は、DB クラスターのクラスターエンドポイントアドレスで使用されます。クラスターエンドポイントの詳細については、「Amazon Aurora 接続管理」を参照してください。 DB クラスター識別子には以下の制約があります。 • 1 ~ 63 文字の英数字またはハイフンを使用する必要があります。 • 1 字目は文字である必要があります。 • ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません。 • 各 AWS リージョンの各 AWS アカウントのすべての DB クラスターの中で一意である必要があります。	AWS CLI を使用して、create-db-cluster を実行し、--db-cluster-identifier オプションを設定します。 RDS API を使用して、CreateDBCluster を呼び出し、DBClusterIdentifier パラメータを設定します。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
DB クラスターのパラメータグループ	DB クラスターのパラメータグループを選択します。Aurora にはデフォルトの DB クラスターパラメータグループが用意されています。また、独自の DB クラスターパラメータグループを作成することもできます。DB クラスターパラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。	AWS CLI を使用して、create-db-cluster を実行し、--db-cluster-parameter-group-name オプションを設定します。 RDS API を使用して、CreateDBCluster を呼び出し、DBClusterParameterGroupName パラメータを設定します。
DB インスタンスクラス	プロビジョニング済みの容量タイプにのみ適用されます。DB クラスターの各インスタンスに対する処理要件やメモリ要件を定義する DB インスタンスクラスを選択します。DB インスタンスクラスの詳細については、「Aurora DB インスタンスクラス」を参照してください。	Aurora クラスター内のすべての DB インスタンスに対して、この値を設定します。 AWS CLI を使用して、create-db-instance を実行し、--db-instance-class オプションを設定します。 RDS API を使用して、CreateDBInstance を呼び出し、DBInstanceClass パラメータを設定します。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
DB パラメータグループ	パラメータグループを選択します。Aurora にはデフォルトのパラメータグループが用意されています。また、独自のパラメータグループを作成することもできます。パラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。	Aurora クラスター内のすべての DB インスタンスに対して、この値を設定します。 AWS CLI を使用して、create-db-instance を実行し、--db-parameter-group-name オプションを設定します。 RDS API を使用して、CreateDBInstance を呼び出し、DBParameterGroupName パラメータを設定します。
DB サブネットグループ	DB クラスターで使用する DB サブネットグループ。 既存の DB サブネットグループを使用するには、[Choose existing] (既存を選択) を選択します。次に、[Existing DB subnet groups] (既存の DB サブネットグループ) ドロップダウンリストから必要なサブネットグループを選択します。 RDS が互換性のある DB サブネットグループを選択できるようにするには、[Automatic setup] (自動セットアップ) を選択します。存在しない場合、RDS はクラスターの新しいサブネットグループを作成します。 詳細については、「DB クラスターの前提条件」を参照してください。	AWS CLI を使用して、create-db-cluster を実行し、--db-subnet-group-name オプションを設定します。 RDS API を使用して、CreateDBCluster を呼び出し、DBSubnetGroupName パラメータを設定します。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
削除保護の有効化	DB クラスターが削除されないようにするには [Enable deletion protection (削除保護の有効化)] を選択します。コンソールで本稼働 DB クラスターを作成する場合、デフォルトで削除保護は有効です。	AWS CLI を使用して、`create-db-cluster` を実行し、`--deletion-protection | --no-deletion-protection` オプションを設定します。 RDS API を使用して、`CreateDBCluster` を呼び出し、`DeletionProtection` パラメータを設定します。
Enable encryption	この DB クラスターを保管時に暗号化するには、`Enable encryption` を選択します。詳細については、「Amazon Aurora リソースの暗号化」を参照してください。	AWS CLI を使用して、`create-db-cluster` を実行し、`--storage-encrypted | --no-storage-encrypted` オプションを設定します。 RDS API を使用して、`CreateDBCluster` を呼び出し、`StorageEncrypted` パラメータを設定します。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
拡張モニタリングを有効にする	DB クラスターが実行されているオペレーティングシステムに対してリアルタイムでのメトリクスの収集を有効にするには、[拡張モニタリングを有効にする] を選択します。詳細については、「拡張モニタリングを使用した OS メトリクスのモニタリング」を参照してください。	Aurora クラスター内のすべての DB インスタンスに対して、これらの値を設定します。 AWS CLI を使用して、`create-db-instance` を実行し、`--monitoring-interval` および `--monitoring-role-arn` オプションを設定します。 RDS API を使用して、`CreateDBInstance` を呼び出し、`MonitoringInterval` および `MonitoringRoleArn` パラメータを設定します。
RDS Data API を有効にする	[RDS Data API を有効にする] を選択して、RDS Data API (Data API) を有効にします。Data API は、接続を管理せずに SQL ステートメントを実行するための安全な HTTP エンドポイントを提供します。詳細については、「RDS Data API の使用」を参照してください。	AWS CLI を使用して、`create-db-cluster` を実行し、`--enable-http-endpoint | --no-enable-http-endpoint` オプションを設定します。 RDS API を使用して、`CreateDBCluster` を呼び出し、`EnableHttpEndpoint` パラメータを設定します。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
エンジンのタイプ	この DB クラスターに使用されるデータベースエンジンを選択します。	AWS CLI を使用して、create-db-cluster を実行し、--engine オプションを設定します。 RDS API を使用して、CreateDBCluster を呼び出し、Engine パラメータを設定します。
エンジンバージョン	プロビジョニング済みの容量タイプにのみ適用されます。DB エンジンのバージョン番号を選択します。	AWS CLI を使用して、create-db-cluster を実行し、--engine-version オプションを設定します。 RDS API を使用して、CreateDBCluster を呼び出し、EngineVersion パラメータを設定します。
フェイルオーバー優先順位	インスタンスのフェイルオーバー優先度を選択します。値を選択しない場合、デフォルト値は [tier-1] になります。この優先度により、プライマリインスタンスの障害からの復旧時に、Aurora レプリカを昇格する順序が決まります。詳細については、「Aurora DB クラスターの耐障害性」を参照してください。	Aurora クラスター内のすべての DB インスタンスに対して、この値を設定します。 AWS CLI を使用して、create-db-instance を実行し、--promotion-tier オプションを設定します。 RDS API を使用して、CreateDBInstance を呼び出し、PromotionTier パラメータを設定します。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
初期データベース名	デフォルトデータベースの名前を入力します。Aurora MySQL DB クラスターの名前を指定しないと、Amazon RDS は作成中の DB クラスターにデータベースを作成しません。Aurora PostgreSQL DB クラスターの名前を指定しないと、Amazon RDS は `postgres` という名前のデータベースを作成します。 Aurora MySQL の場合、デフォルトデータベース名には以下の制約があります。 • 1～64 個の英数字を使用する必要があります。 • データベースエンジンの予約語は使用できません。 Aurora PostgreSQL の場合、デフォルトデータベース名には以下の制約があります。 • 1～63 個の英数字を使用する必要があります。 • 先頭は文字にする必要があります。後続の文字には、英字、アンダースコア、または数字 (0～9) を使用できます。 • データベースエンジンの予約語は使用できません。	AWS CLI を使用して、`create-db-cluster` を実行し、`--database-name` オプションを設定します。 RDS API を使用して、`CreateDBCluster` を呼び出し、`DatabaseName` パラメータを設定します。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
	追加のデータベースを作成するには、DB クラスターに接続し、SQL コマンド CREATE DATABASE を使用します。DB クラスターへの接続の詳細については、「Amazon Aurora DB クラスターへの接続」を参照してください。	
ログのエクスポート	[ログのエクスポート] セクションで、Amazon CloudWatch Logs に公開するログを選択します。Aurora MySQL ログの Amazon CloudWatch Logs への公開に関する詳細については、Amazon CloudWatch Logs への Amazon Aurora MySQL ログの発行を参照してください。Aurora PostgreSQL logs の CloudWatch Logs への公開に関する詳細については、Amazon CloudWatch Logs への Aurora PostgreSQL ログの発行を参照してください。	AWS CLI を使用して、create-db-cluster を実行し、--enable-cloudwatch-logs-exports オプションを設定します。 RDS API を使用して、CreateDBCluster を呼び出し、EnableCloudwatchLogsExports パラメータを設定します。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
メンテナンスウィンドウ	[ウィンドウの選択] を選択し、システムメンテナンスが実行される期間を週単位で指定します。または、[指定なし] を選択して、Amazon RDS によって期間がランダムに割り当てられるようにします。	AWS CLI を使用して、create-db-cluster を実行し、--preferred-maintenance-window オプションを設定します。 RDS API を使用して、CreateDBCluster を呼び出し、PreferredMaintenanceWindow パラメータを設定します。
AWS Secrets Manager でマスター認証情報を管理する	[Manage master credentials in AWS Secrets Manager] (でマスター認証情報を管理する) を選択して、Secrets Manager でユーザーのパスワードをシークレットに管理します。 オプションで、シークレットを保護するために使用する KMS キーを選択します。お客様のアカウントの KMS キーから選択するか、別のアカウントからキーを入力します。 詳細については、「Amazon Aurora および AWS Secrets Manager によるパスワード管理」を参照してください。	AWS CLI を使用して、create-db-cluster を実行し、--manage-master-user-password | --no-manage-master-user-password および --master-user-secret-kms-key-id オプションを設定します。 RDS API を使用して、CreateDBCluster を呼び出し、ManageMasterUserPassword および MasterUserSecretKmsKeyId パラメータを設定します。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
マスターパスワード	DB クラスターにログオンするためのパスワードを入力します。 • Aurora MySQL の場合、パスワードには 8 ~ 41 個の印刷可能な ASCII 文字を使用する必要があります。 • Aurora PostgreSQL の場合は、8 ~ 99 個の印刷可能な ASCII 文字を使用する必要があります。 • /、"、@、またはスペースは使用できません。	AWS CLI を使用して、create-db-cluster を実行し、--master-user-password オプションを設定します。 RDS API を使用して、CreateDBCluster を呼び出し、MasterUserPassword パラメータを設定します。
マスターユーザーネーム	DB クラスターにログオンするためのマスターユーザー名として使用する名前を入力します。 • Aurora MySQL の場合、名前には 1 ~ 16 個の英数字を使用する必要があります。 • Aurora PostgreSQL の場合は、1 ~ 63 個の英数字を使用する必要があります。 • 1 字目は文字である必要があります。 • 名前にはデータベースエンジンの予約語を使用できません。 DB クラスターの作成後にマスターユーザー名を変更することはできません。	AWS CLI を使用して、create-db-cluster を実行し、--master-username オプションを設定します。 RDS API を使用して、CreateDBCluster を呼び出し、MasterUsername パラメータを設定します。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
マルチ AZ 配置	プロビジョニング済みの容量タイプにのみ適用されます。フェイルオーバーのサポート用に他のアベイラビリティーゾーンで Aurora レプリカを作成するかどうかを決めます。[Create Replica in Different Zone (別のゾーンでレプリカを作成)] を選択した場合、Amazon RDS は DB クラスターのプライマリインスタンスとは異なるアベイラビリティーゾーンに DB クラスターの Aurora レプリカを作成します。複数のアベイラビリティーゾーンの詳細については、「リージョンとアベイラビリティーゾーン」を参照してください。	AWS CLI を使用して、`create-db-cluster` を実行し、`--availability-zones` オプションを設定します。 RDS API を使用して、`CreateDBCluster` を呼び出し、`AvailabilityZones` パラメータを設定します。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
ネットワークの種類	DB クラスターでサポートされている IP アドレス設定プロトコル。 リソースが、IPv4 アドレス設定プロトコル経由でのみ DB クラスターと通信できるように指定する IPv4。 リソースが、IPv4、IPv6、またはその両方で DB クラスター通信できるように指定するデュアルスタックモード。IPv6 アドレス設定プロトコルで DB クラスターと通信する必要があるリソースがある場合は、デュアルスタックモードを使用します。デュアルスタックモードを使用するには、IPv4 と IPv6 の両方のネットワークプロトコルをサポートする 2 つのアベイラビリティーゾーンにまたがるサブネットが最低 2 つ必要です。また、IPv6 CIDR ブロックを、指定した DB サブネットグループのサブネットに関連付けてください。 詳細については、「Amazon Aurora IP アドレス指定」を参照してください。	AWS CLI を使用して、`create-db-cluster` を実行し、`--network-type` オプションを設定します。 RDS API を使用して、`CreateDBCluster` を呼び出し、`NetworkType` パラメータを設定します。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
パブリックアクセス	DB クラスターにパブリック IP アドレスを付与するには [パブリックアクセス可能] を選択します。付与しない場合は、[パブリックアクセス不可] を選択します。DB クラスターのインスタンスでは、パブリック DB インスタンスとプライベート DB インスタンスの両方を混在させることができます。パブリックアクセスからインスタンスを隠す方法については、「VPC 内の DB クラスターをインターネットから隠す」を参照してください。 Amazon VPC の外部から DB インスタンスに接続するには、DB インスタンスがパブリックにアクセス可能であり、アクセスが DB インスタンスのセキュリティグループのインバウンドルールで許可されているなど、いくつかの要件を満たす必要があります。詳細については、「Amazon RDS DB インスタンスに接続できない」を参照してください。 DB インスタンスがパブリックアクセス可能でない場合は、AWS Site-to-Site VPN 接続または AWS Direct Connect 接続を使用してプライベートネットワークからアクセスすることもできます。詳細については、「インターネットトラ	Aurora クラスター内のすべての DB インスタンスに対して、この値を設定します。 AWS CLI を使用して、`create-db-instance` を実行し、`--publicly-accessible | --no-publicly-accessible` オプションを設定します。 RDS API を使用して、`CreateDBInstance` を呼び出し、`PubliclyAccessible` パラメータを設定します。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
	フィックのプライバシー」を参照してください。	
RDS 延長サポート	Aurora 標準サポート終了日を過ぎてもサポートされているメジャーエンジンバージョンを 引き続き実行するには、[RDS 延長サポートを有効にする] を選択します。 DB クラスターを作成する場合、Amazon Aurora はデフォルトで RDS 延長サポートを選択します。Aurora 標準サポート 終了日後に新しい DB クラスターが作成されて RDS 延長サポートの料金が発生するのを避けるには、この設定を無効にします。既存の DB クラスターについて、RDS 延長サポートの課金開始日以前に料金が発生することはありません。 詳細については、「Amazon RDS 延長サポートの使用」を参照してください。	AWS CLI を使用して、create-db-cluster を実行し、--engine-lifecycle-support オプションを設定します。 RDS API を使用して、CreateDBCluster を呼び出し、EngineLifecycleSupport パラメータを設定します。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
RDS Proxy	[Create an RDS Proxy] (RDS Proxy の作成) を選択して、DB クラスターにプロキシを作成します。Amazon RDS は、プロキシの IAM ロールと Secrets Manager シークレットを自動的に作成します。 詳細については、「Amazon RDS Proxy for Aurora の使用」を参照してください。	DB クラスターの作成時には使用できません。
保持期間	Aurora がデータベースのバックアップコピーを保持する期間 (1〜35 日) を選択します。バックアップコピーは、2 番目のデータベースに対するポイントインタイム復元 (PITR) で使用できます。	AWS CLI を使用して、`create-db-cluster` を実行し、`--backup-retention-period` オプションを設定します。 RDS API を使用して、`CreateDBCluster` を呼び出し、`BackupRetentionPeriod` パラメータを設定します。
DevOps Guru をオンにする	[Turn on DevOps Guru] (DevOps Guru をオンにする) を選択して Aurora データベースの Amazon DevOps Guru をオンにします。DevOps Guru for RDS で、パフォーマンスの異常の詳細な分析を提供するには、[Performance Insights] (パフォーマンスインサイト) をオンにする必要があります。詳細については、「RDS 用の DevOps Guru のセットアップ」を参照してください。	RDS コンソールで DevOps Guru for RDS を有効にすることはできますが、RDS API または CLI を使用することはできません。「DevOps Guru をオンにする」詳細については、Amazon DevOps Guru ユーザーガイドを参照してください。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
Performance Insights をオンにする	[Turn on Performance Insights] (Performance Insights をオンにする) をクリックして Amazon RDS Performance Insights を有効にします。詳細については、「Amazon Aurora での Performance Insights を使用したDB 負荷のモニタリング」を参照してください。	Aurora クラスター内のすべての DB インスタンスに対して、これらの値を設定します。 AWS CLI を使用して、`create-db-instance` を実行し、`--enable-performance-insights | --no-enable-performance-insights` 、`--performance-insights-kms-key-id` 、および `--performance-insights-retention-period` オプションを設定します。 RDS API を使用して `CreateDBInstance` を呼び出し、`EnablePerformanceInsights` 、`PerformanceInsightsKMSKeyId` 、および `PerformanceInsightsRetentionPeriod` パラメータを設定します。
仮想プライベートクラウド (VPC)	DB クラスターをホストする VPC を選択します。[新しい VPC の作成] を選択して、Amazon RDS で VPC を作成します。詳細については、「DB クラスターの前提条件」を参照してください。	AWS CLIおよび API の場合は、VPC セキュリティグループ ID を指定します。

コンソール設定	設定の説明	CLI オプションと RDS API パラメータ
VPC セキュリティグループ (ファイアウォール)	[Create new] を選択して、Amazon RDS で VPC セキュリティグループを作成します。または、[既存の選択] を選択して、1 つ以上の VPC セキュリティグループを指定して、DB クラスターへのネットワークアクセスを保護します。 RDS コンソールで [Create new] を選択すると、新しく作成されるセキュリティグループのインバウンドルールにより、ブラウザで検出された IP アドレスから DB インスタンスへのアクセスが許可されます。 詳細については、「DB クラスターの前提条件」を参照してください。	AWS CLI を使用して、create-db-cluster を実行し、--vpc-security-group-ids オプションを設定します。 RDS API を使用して、CreateDBCluster を呼び出し、VpcSecurityGroupIds パラメータを設定します。

DB クラスター用 Amazon Aurora に適用されない設定

AWS CLI コマンドの create-db-cluster、および RDS API オペレーションの CreateDBCluster に関する以下の設定は、Amazon Aurora DB クラスターには適用されません。

> **Note**
>
> AWS Management Console には、Aurora DB クラスターのこれらの設定は表示されません。

AWS CLI の設定	RDS API の設定
--allocated-storage	AllocatedStorage
--auto-minor-version-upgrade | --no-auto-minor-version-upgrade	AutoMinorVersionUpgrade
--db-cluster-instance-class	DBClusterInstanceClass
--enable-performance-insights | --no-enable-performance-insights	EnablePerformanceInsights
--iops	Iops
--monitoring-interval	MonitoringInterval
--monitoring-role-arn	MonitoringRoleArn
--option-group-name	OptionGroupName
--performance-insights-kms-key-id	PerformanceInsightsKMSKeyId
--performance-insights-retention-period	PerformanceInsightsRetentionPeriod
--publicly-accessible | --no-publicly-accessible	PubliclyAccessible

Amazon Aurora DB インスタンスには適用されない設定

AWS CLI コマンドの create-db-instance、および RDS API オペレーションの CreateDBInstance に関する次の設定は、DB インスタンス Amazon Aurora DB クラスターには適用されません。

> **Note**
>
> AWS Management Console には、Aurora DB インスタンスのこれらの設定は表示されません。

AWS CLI の設定	RDS API の設定
`--allocated-storage`	`AllocatedStorage`
`--availability-zone`	`AvailabilityZone`
`--backup-retention-period`	`BackupRetentionPeriod`
`--backup-target`	`BackupTarget`
`--character-set-name`	`CharacterSetName`
`--character-set-name`	`CharacterSetName`
`--custom-iam-instance-profile`	`CustomIamInstanceProfile`
`--db-security-groups`	`DBSecurityGroups`
`--deletion-protection | --no-deletion-protection`	`DeletionProtection`
`--domain`	`Domain`
`--domain-iam-role-name`	`DomainIAMRoleName`
`--enable-cloudwatch-logs-exports`	`EnableCloudwatchLogsExports`
`--enable-customer-owned-ip | --no-enable-customer-owned-ip`	`EnableCustomerOwnedIp`
`--enable-iam-database-authentication | --no-enable-iam-database-authentication`	`EnableIAMDatabaseAuthentication`

AWS CLI の設定	RDS API の設定
--engine-version	EngineVersion
--iops	Iops
--kms-key-id	KmsKeyId
--master-username	MasterUsername
--master-user-password	MasterUserPassword
--max-allocated-storage	MaxAllocatedStorage
--multi-az | --no-multi-az	MultiAZ
--nchar-character-set-name	NcharCharacterSetName
--network-type	NetworkType
--option-group-name	OptionGroupName
--preferred-backup-window	PreferredBackupWindow
--processor-features	ProcessorFeatures
--storage-encrypted | --no-storage-encrypted	StorageEncrypted
--storage-type	StorageType
--tde-credential-arn	TdeCredentialArn
--tde-credential-password	TdeCredentialPassword
--timezone	Timezone
--vpc-security-group-ids	VpcSecurityGroupIds

AWS CloudFormation での Amazon Aurora リソースの作成

Amazon Aurora は AWS CloudFormation と統合されています。これは、リソースとインフラストラクチャの作成と管理の所要時間を短縮できるように AWS リソースをモデル化して設定するためのサービスです。必要なすべての AWS リソース (DB クラスターや DB クラスターパラメータグループなど) を記述するテンプレートを作成し、AWS CloudFormation はそれらのリソースをプロビジョニングして設定します。

AWS CloudFormation を使用すると、テンプレートを再利用して Aurora リソースを同じように繰り返してセットアップできます。リソースを一度記述すると、同じリソースを複数の AWS アカウントおよびリージョンで何度でも繰り返してプロビジョニングできます。

Aurora とAWS CloudFormationテンプレート

Aurora および関連サービスのリソースをプロビジョニングして設定するには、AWS CloudFormation テンプレートについて理解しておく必要があります。テンプレートは、JSON または YAML でフォーマットされたテキストファイルです。これらのテンプレートには、AWS CloudFormation スタックにプロビジョニングしたいリソースを記述します。JSONやYAMLに不慣れな方は、AWS CloudFormation Designerを使えば、 AWS CloudFormation テンプレートを使いこなすことができます。詳細については、AWS CloudFormation ユーザーガイドの「AWS CloudFormation Designer とは」を参照してください。

Aurora は、AWS CloudFormation でのリソースの作成をサポートしています。これらのリソースの JSON テンプレートと YAML テンプレートの例を含む詳細については、AWS CloudFormation ユーザーガイドの「RDS リソースタイプのリファレンス」を参照してください。

AWS CloudFormation の詳細はこちら

AWS CloudFormation の詳細については、以下のリソースを参照してください。

- AWS CloudFormation
- AWS CloudFormation ユーザーガイド
- AWS CloudFormation API リファレンス
- AWS CloudFormation コマンドラインインターフェイスユーザーガイド

Amazon Aurora DB クラスターへの接続

Aurora DB クラスターには、MySQL または PostgreSQL データベースへの接続に使用するものと同じツールを使用して接続することができます。MySQL または PostgreSQL DB インスタンスに接続するスクリプト、ユーティリティ、またはアプリケーションで接続文字列を指定します。Secure Sockets Layer (SSL) 接続の場合と同じパブリックキーを使用します。

接続文字列では、通常、DB クラスターに関連付けられた特別なエンドポイントからのホストとポートの情報を使用します。このようなエンドポイントでは、クラスター内の DB インスタンスの数に関係なく、同じ接続パラメータを使用できます。トラブルシューティングなどの特殊なタスクの場合、Aurora DB クラスター内の特定の DB インスタンスの、ホストとポート情報を使用できます。

> **Note**
>
> Aurora Serverless DB クラスターの場合、DB インスタンスにではなく、データベースエンドポイントに接続します。Aurora Serverless DB クラスターのデータベースエンドポイントは、AWS Management Console の [接続とセキュリティ] タブにあります。詳細については、「Amazon Aurora Serverless v1 の使用」を参照してください。

エンドポイントには、DB クラスターまたはインスタンスの操作に使用している Aurora DB エンジンや特定のツールに関係なく、アクセスできる必要があります。Aurora DB クラスターは、Amazon VPC サービスに基づいて仮想プライベートクラウド (VPC) のみで作成できます。つまり、エンドポイントには、VPC の内部または VPC の外部のいずれかから次のいずれかの方法を使用してアクセスします。

- VPC 内の Aurora DB クラスターにアクセスする - VPC 経由で Aurora DB クラスターへのアクセスを有効にします。そのためには、セキュリティグループ上の VPC のためのインバウンドルールを編集し、特定の Aurora DB クラスターへのアクセスを許可します。さまざまな Aurora DB クラスターのシナリオで VPC を設定する方法など、詳細については「Amazon Virtual Private Cloud (VPC) と Amazon Aurora」を参照してください。
- VPC 外部にある Aurora DB クラスターへのアクセス - VPC 外部から Aurora DB クラスターにアクセスするには、DB クラスターのパブリックエンドポイントアドレスを使用します。

詳細については、「Aurora 接続障害のトラブルシューティング」を参照してください。

目次

AWS ドライバーを使用した Aurora DB クラスターへの接続

AWS のドライバースイートは、スイッチオーバーとフェイルオーバーの時間の短縮、AWS Secrets Manager、AWS Identity and Access Management (IAM)、フェデレーティッド ID での認証をサポートするように設計されています。AWS ドライバーは、DB クラスターステータスをモニタリングし、クラスタートポロジを認識して新しいライターを決定することを前提としています。このアプローチにより、スイッチオーバーとフェイルオーバーの時間が 1 桁秒に短縮されます (オープンソースドライバーの場合は数十秒)。

次の表に、各ドライバーでサポートされている機能を示します。新しいサービス機能が導入されるにあたって、こうしたサービス機能を標準でサポートすることが AWS のドライバースイートの目標です。

機能	AWS JDBC ドライバー	AWS Python ドライバー
フェイルオーバーのサポート	はい	はい
フェイルオーバーモニタリングの強化	はい	はい
読み取り/書き込みの分割	はい	はい

機能	AWS JDBC ドライバー	AWS Python ドライバー
Aurora 接続トラッカー	はい	はい
ドライバーメタデータ接続	はい	該当なし
Telemetry	はい	はい
Secrets Manager	はい	はい
IAM 認証	はい	はい
フェデレーティッド ID (AD FS)	はい	はい
フェデレーティッド ID (Okta)	はい	いいえ

AWS ドライバーの詳細については、お使いの Aurora MySQL または Aurora PostgreSQL DB クラスターに対応する言語ドライバーを参照してください。

Amazon Aurora MySQL DB クラスターへの接続

Aurora MySQL DB クラスターを認証するには、MySQL のユーザー名とパスワード認証、または AWS Identity and Access Management (IAM) データベース認証を使用できます。MySQL のユーザー名およびパスワード認証を使用する方法については、MySQL ドキュメントの「Access control and account management」を参照してください。IAM データベース認証を使用する方法については、「 の IAM データベース認証」を参照してください。

MySQL 8.0 と互換性のある Amazon Aurora DB クラスターに接続すると、MySQL バージョン 8.0 と互換性のある SQL コマンドを実行できます。最小互換バージョンは MySQL 8.0.23 です。MySQL 8.0 の SQL 構文の詳細については、MySQL 8.0 リファレンスマニュアルを参照してください。Aurora MySQL バージョン 3 に適用される制限事項については、「Aurora MySQL バージョン 3 と MySQL 8.0 コミュニティエディションの比較」を参照してください。

MySQL 5.7 と互換性のある Amazon Aurora DB クラスターに接続すると、MySQL バージョン 5.7 と互換性のある SQL コマンドを実行できます。MySQL 5.7 の SQL 構文の詳細については、MySQL 5.7 リファレンスマニュアルを参照してください。Aurora MySQL 5.7 に適用される制限事項については、「MySQL 5.7 互換 Aurora MySQL バージョン 2」を参照してください。

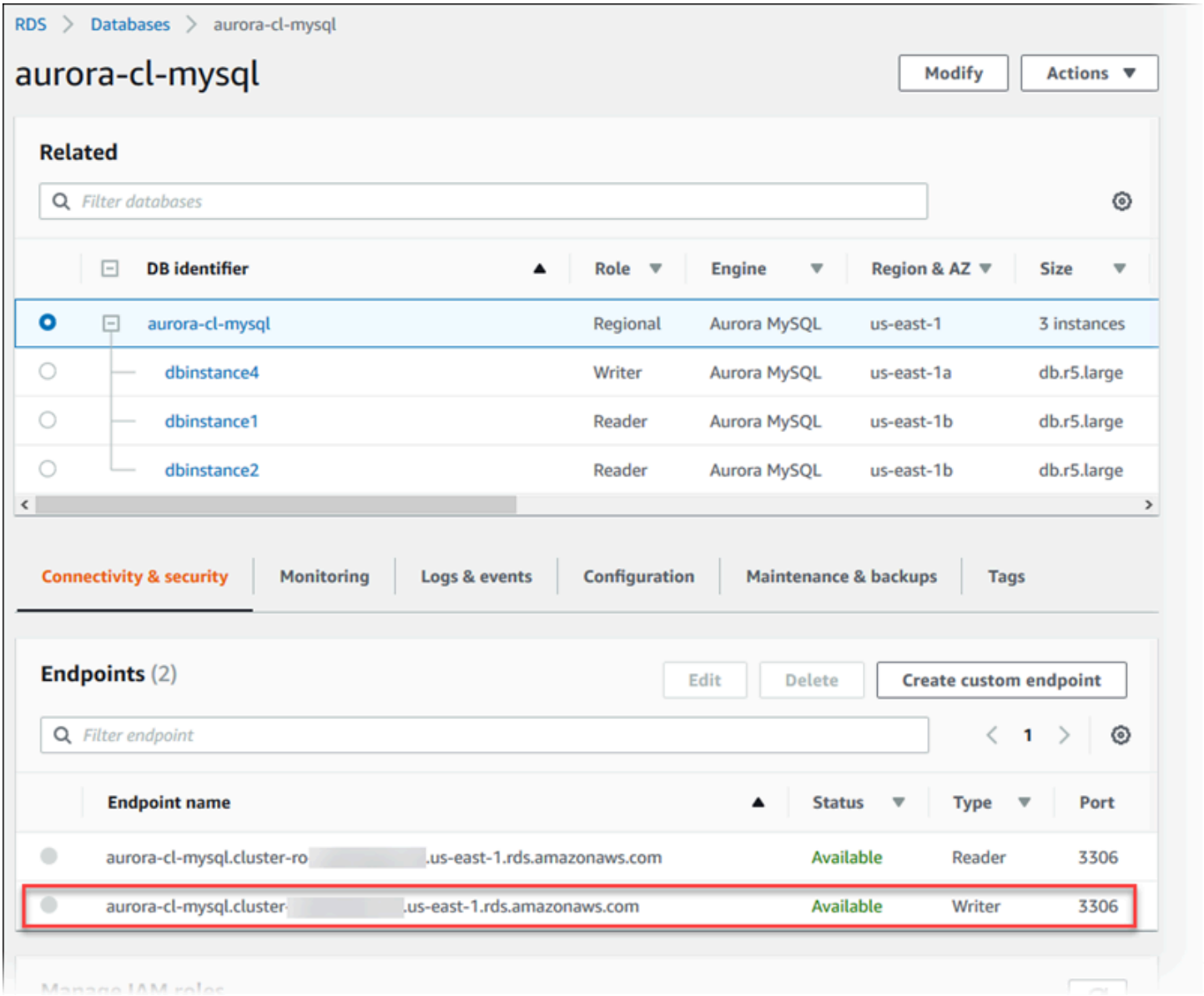
> **Note**
>
> Amazon Aurora MySQL DB クラスターへの接続に役立つ詳しいガイドについては、「Aurora 接続管理」ハンドブックを参照してください。

DB クラスターの詳細ビューで、MySQL 接続文字列で使用できるクラスターエンドポイントを確認できます。エンドポイントは DB クラスターのドメイン名とポートで構成されます。例えば、エンドポイントの値が `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com:3306` の場合、MySQL 接続文字列には次の値を指定します。

- ホストまたはホスト名には `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com` を指定します
- ポートには、`3306` を指定するか、DB クラスターの作成時に使用したポート値を指定します。

クラスターエンドポイントは DB クラスターのプライマリインスタンスに接続します。このクラスターエンドポイントを使用して、読み取りと書き込みの両方のオペレーションを実行できます。DB クラスターは、DB クラスターのデータへの読み取り専用アクセスをサポートする、最大 15 個の Aurora レプリカを使用できます。プライマリインスタンスと各 Aurora レプリカには、クラスターエンドポイントとは無関係の一意なエンドポイントがあり、そのエンドポイントにより、クラスター内の特定の DB インスタンスに直接接続できます。クラスターエンドポイントは常にプライマリインスタンスをポイントします。プライマリインスタンスが失敗し、置き換えられると、クラスターエンドポイントは新しいプライマリインスタンスをポイントします。

クラスターエンドポイント (ライターエンドポイント) を確認するには、Amazon RDS コンソールで [データベース] を選択し、DB クラスターの名前を選択して DB クラスターの詳細を表示します。



トピック

- Aurora MySQL の接続ユーティリティ
- MySQL ユーティリティを使用した Aurora MySQL との接続
- Amazon Web Services (AWS) JDBC ドライバーを使用した Aurora MySQL への接続
- Amazon Web Services (AWS) Python ドライバーを使用した Aurora MySQL への接続
- SSL を使用した Aurora MySQL への接続

Aurora MySQL の接続ユーティリティ

使用できる接続ユーティリティは次のとおりです。

- コマンドライン - MySQL コマンドラインユーティリティなどのツールを使用して、Amazon Aurora DB クラスターに接続できます。MySQL ユーティリティを使用する方法の詳細については、MySQL ドキュメントの「mysql — the MySQL command-line client」(mysql - MySQL コマンドラインクライアント) を参照してください。
- GUI - UI インターフェイスを使用することで、MySQL Workbench ユーティリティを使用して接続できます。詳細については、「MySQL Workbench のダウンロード」ページを参照してください。
- AWS ドライバー:
  - Amazon Web Services (AWS) JDBC ドライバーを使用した Aurora MySQL への接続
  - Amazon Web Services (AWS) Python ドライバーを使用した Aurora MySQL への接続

MySQL ユーティリティを使用した Aurora MySQL との接続

次の手順に従ってください。VPC のプライベートサブネットに DB クラスターを設定していることを前提としています。チュートリアル: ウェブサーバーと Amazon Aurora DB クラスターを作成するのチュートリアルに従って設定した Amazon EC2 インスタンスを使用して接続します。

> **Note**
>
> この手順では、チュートリアルで Web サーバーをインストールする必要はありませんが、MariaDB 10.5 のインストールは必要です。

MySQL ユーティリティを使用して DB クラスターに接続するには

1. DB クラスターへの接続に使用する EC2 インスタンスにログインします。

   次のような出力が表示されます。

```
Last login: Thu Jun 23 13:32:52 2022 from xxx.xxx.xxx.xxx

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-10-0-xxx.xxx ~]$
```

2. DB クラスターのプライマリインスタンスに接続するには、コマンドプロンプトで次のコマンドを入力します。

-h パラメータは、エンドポイント DNS 名をプライマリインスタンスの代わりに使用します。-u パラメータは、データベースユーザーアカウントのユーザー ID を置き換えます。

```
mysql -h primary-instance-endpoint.AWS_account.AWS_Region.rds.amazonaws.com -P 3306
 -u database_user -p
```

例:

```
mysql -h my-aurora-cluster-instance.c1xy5example.123456789012.eu-
central-1.rds.amazonaws.com -P 3306 -u admin -p
```

3. データベースユーザーのパスワードを入力します。

次のような出力が表示されます。

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 1770
Server version: 8.0.23 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>
```

4. SQL コマンドを入力します。

Amazon Web Services (AWS) JDBC ドライバーを使用した Aurora MySQL への接続

Amazon Web Services (AWS) JDBC ドライバーは、高度な JDBC ラッパーとして設計されています。このラッパーは、既存の JDBC ドライバーの機能を補完および拡張して、アプリケーションが Aurora MySQL などのクラスター化されたデータベースの機能を利用できるようにします。ドライバーには、コミュニティ MySQL Connector/J ドライバーおよびコミュニティ MariaDB Connector/J ドライバーとドロップイン互換性があります。

AWS JDBC ドライバーをインストールするには、AWS JDBC ドライバーの.jar ファイル (CLASSPATH アプリケーション内) を追加して、それぞれのコミュニティドライバーへの参照を保持します。対応する接続 URL プレフィックスを次のように更新します。

- `jdbc:mysql://` を `jdbc:aws-wrapper:mysql://` に

- `jdbc:mariadb://` を `jdbc:aws-wrapper:mariadb://` に

AWS JDBC ドライバーおよびその使用方法の詳細については、「Amazon Web Services (AWS) JDBC ドライバー GitHub リポジトリ」を参照してください。

> **Note**
>
> MariaDB Connector/J ユーティリティのバージョン 3.0.3 では Aurora DB クラスターがサポートされなくなるため、AWS JDBC ドライバーに移行することを強くお勧めします。

Amazon Web Services (AWS) Python ドライバーを使用した Aurora MySQL への接続

Amazon Web Services (AWS) Python ドライバーは、高度な Python ラッパーとして設計されています。このラッパーは、オープンソースの Psycopg ドライバーの機能を補完し、拡張します。AWS Python ドライバーは Python バージョン 3.8 以降をサポートしています。`aws-advanced-python-wrapper` パッケージは、`pip` コマンドと `psycopg` オープンソースパッケージを使用してインストールできます。

AWS Python ドライバーおよびその使用方法の詳細については、「Amazon Web Services (AWS) Python Driver GitHub repository」を参照してください。

SSL を使用した Aurora MySQL への接続

Aurora MySQL DB インスタンスへの接続には、SSL 暗号化を使用できます。詳細については、Aurora MySQL DB クラスターでの TLS の使用 を参照してください。

SSL を使用して接続するには、MySQL ユーティリティを次の手順に従って使用します。IAM データベース認証を使用している場合は、SSL 接続を使用する必要があります。詳細については、 の IAM データベース認証 を参照してください。

> **Note**
>
> SSL を使用してクラスターエンドポイントに接続するには、クライアント接続ユーティリティでサブジェクト代替名 (SAN) がサポートされている必要があります。クライアント接続ユーティリティで SAN がサポートされていない場合、Aurora DB クラスター内のインスタンスに直接接続することができます。Aurora エンドポイントの詳細については、「Amazon Aurora 接続管理」を参照してください。

MySQL ユーティリティを使用して SSL で DB クラスターに接続するには

1. Amazon RDS 署名証明書のパブリックキーをダウンロードします。

   証明書のダウンロードについては、SSL/TLS を使用した DB クラスターへの接続の暗号化 を参照してください。
2. MySQL ユーティリティを使用して SSL で DB クラスターのプライマリインスタンスに接続するには、コマンドプロンプトで次のコマンドを入力します。-h パラメータは、エンドポイント DNS 名をプライマリインスタンスの代わりに使用します。-u パラメータは、データベースユーザーアカウントのユーザー ID を置き換えます。--ssl-ca パラメータは、必要に応じて SSL 証明書のファイル名に置き換えます。プロンプトが表示されたら、マスターユーザーパスワードを入力します。

   ```
   mysql -h mycluster-primary.123456789012.us-east-1.rds.amazonaws.com -u admin_user -p --ssl-ca=[full path]global-bundle.pem --ssl-verify-server-cert
   ```

次のような出力が表示されます。

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 350
Server version: 8.0.26-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

RDS for MySQL 接続文字列の作成と SSL 接続用のパブリックキーの検索に関する一般的な手順については、「MySQL データベースエンジンを実行している DB インスタンスに接続する」を参照してください。

Amazon Aurora PostgreSQL DB クラスターへの接続

Amazon Aurora PostgreSQL DB クラスターの DB インスタンスに接続するには、PostgreSQL データベースに接続する場合と同じツールを使用できます。このプロセスの一部として、Secure Sockets Layer (SSL) 接続の場合と同じ発行鍵を使用します。Aurora PostgreSQL DB クラスターのプライマリインスタンスや Aurora レプリカのエンドポイントとポート情報を、PostgreSQL の DB インスタンスに接続するすべてのスクリプト、ユーティリティ、アプリケーションの接続文字列で使用できます。接続文字列では、プライマリインスタンスまたは Aurora レプリカのエンドポイントの DNS ア

ドレスをホストパラメータとして指定します。エンドポイントのポート番号をポートパラメータとして指定します。

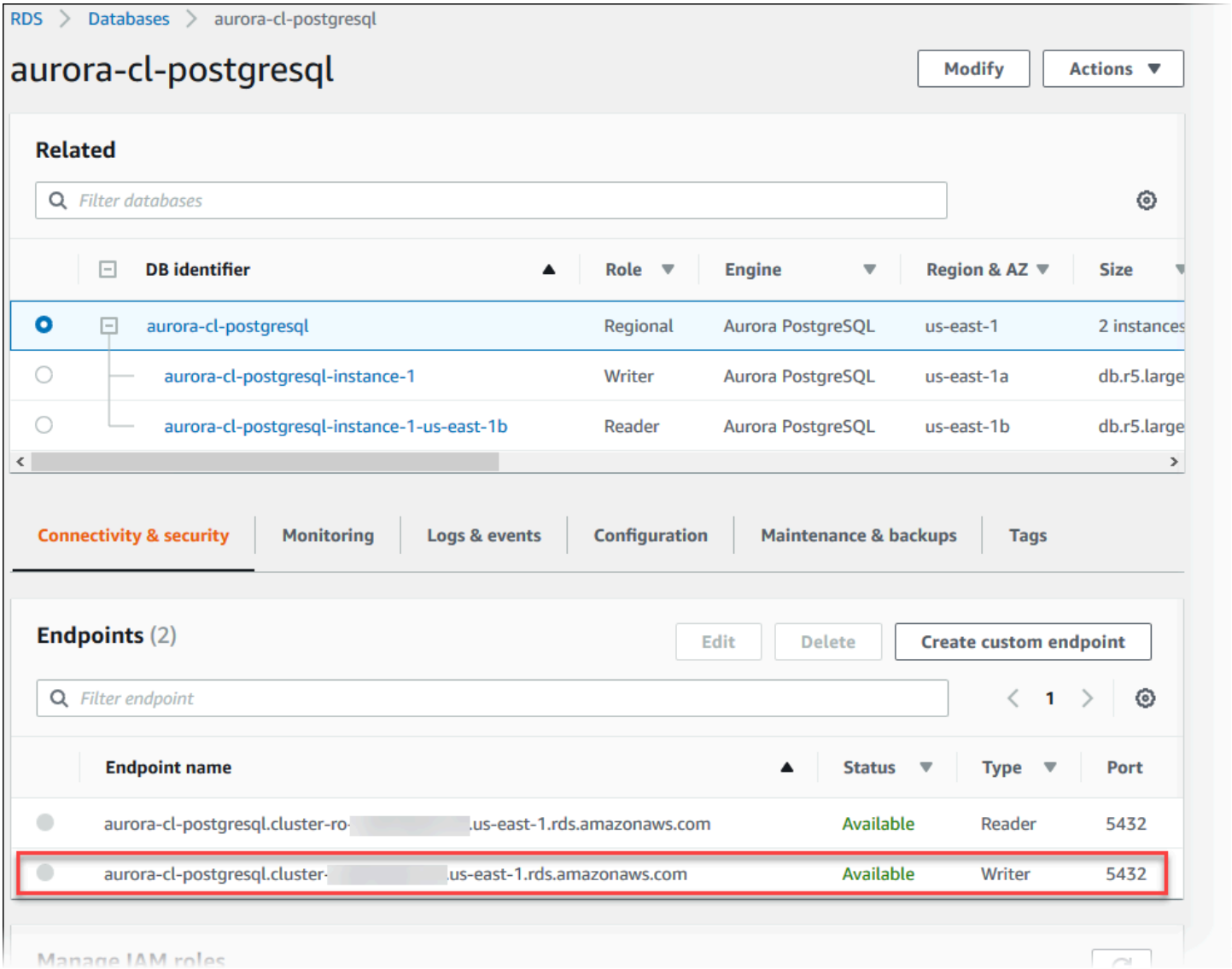
Amazon Aurora PostgreSQL DB クラスターの DB インスタンスに接続すると、PostgreSQL と互換性のある任意の SQL コマンドを実行できます。

Aurora PostgreSQL DB クラスターの詳細表示で、クラスターエンドポイントの名称、ステータス、タイプ、ポート番号を確認できます。このエンドポイントとポート番号は、PostgreSQL 接続文字列で使用します。例えば、エンドポイントの値が `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com` の場合、PostgreSQL 接続文字列には次の値を指定します。

- ホストまたはホスト名には `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com` を指定します
- ポートには、`5432` を指定するか、DB クラスターの作成時に使用したポート値を指定します。

クラスターエンドポイントは DB クラスターのプライマリインスタンスに接続します。このクラスターエンドポイントを使用して、読み取りと書き込みの両方のオペレーションを実行できます。DB クラスターは、DB クラスターのデータへの読み取り専用アクセスをサポートする、最大 15 個の Aurora レプリカを使用できます。Aurora クラスターの各 DB インスタンス (プライマリインスタンスと各 Aurora のレプリカ) には、それぞれ、クラスターエンドポイントとは別に一意のエンドポイントが含まれます。一意のエンドポイントを使用することで、クラスターの特定の DB インスタンスに直接接続できるようになります。クラスターエンドポイントは常にプライマリインスタンスをポイントします。プライマリインスタンスが失敗し、置き換えられると、クラスターエンドポイントは新しいプライマリインスタンスをポイントします。

クラスターエンドポイント (ライターエンドポイント) を確認するには、Amazon RDS コンソールで [データベース] を選択し、DB クラスターの名前を選択して DB クラスターの詳細を表示します。



Aurora PostgreSQL の接続ユーティリティ

使用できる接続ユーティリティは次のとおりです。

- コマンドライン - PostgreSQL インタラクティブターミナルである psql などのツールを使用して、Aurora PostgreSQL DB クラスターに接続できます。PostgreSQL インタラクティブターミナルの詳しい使用方法については、PostgreSQL ドキュメントの「psql」を参照してください。
- GUI - UI インターフェイスで pgAdmin ユーティリティを使用して、Aurora PostgreSQL DB クラスターに接続できます。詳細については、pgAdmin ウェブサイトのダウンロードページを参照してください。
- AWS ドライバー:
  - Amazon Web Services (AWS) JDBC ドライバーを使用した Aurora PostgreSQL への接続

- Amazon Web Services (AWS) Python ドライバーを使用した Aurora PostgreSQL への接続

Amazon Web Services (AWS) JDBC ドライバーを使用した Aurora PostgreSQL への接続

Amazon Web Services (AWS) JDBC ドライバーは、高度な JDBC ラッパーとして設計されています。このラッパーは、既存の JDBC ドライバーの機能を補完および拡張して、アプリケーションが Aurora PostgreSQL などのクラスター化されたデータベースの機能を利用できるようにします。ドライバーには、コミュニティ pgJDBC ドライバーとドロップイン互換性があります。

AWS JDBC ドライバーをインストールするには、AWS JDBC ドライバーの.jar ファイル (CLASSPATH アプリケーション内) を追加して、pgJDBC コミュニティドライバーへの参照を保持します。接続 URL プレフィックスを `jdbc:postgresql://` から `jdbc:aws-wrapper:postgresql://` に更新します。

AWS JDBC ドライバーおよびその使用方法の詳細については、「Amazon Web Services (AWS) JDBC ドライバー GitHub リポジトリ」を参照してください。

Amazon Web Services (AWS) Python ドライバーを使用した Aurora PostgreSQL への接続

Amazon Web Services (AWS) Python ドライバーは、高度な Python ラッパーとして設計されています。このラッパーは、オープンソースの Psycopg ドライバーの機能を補完し、拡張します。AWS Python ドライバーは Python バージョン 3.8 以降をサポートしています。`aws-advanced-python-wrapper` パッケージは、`pip` コマンドと `psycopg` オープンソースパッケージを使用してインストールできます。

AWS Python ドライバーおよびその使用方法の詳細については、「Amazon Web Services (AWS) Python Driver GitHub repository」を参照してください。

Aurora 接続障害のトラブルシューティング

新しい Aurora DB クラスターに対する接続障害の一般的な原因を次に示します。

- VPC のセキュリティグループがアクセスを許可していない場合、VPC では、VPC のセキュリティグループを適切に設定して、デバイスまたは Amazon EC2 インスタンスからの接続を許可する必要があります。障害を解決するには、VPC のセキュリティグループの、インバウンドルールを変更して接続を許可します。例については、「チュートリアル: DB クラスターで使用する VPC を作成する (IPv4 専用)」を参照してください。

- ファイアウォールルールによってブロックされたポート - Aurora DB クラスター用に設定されたポートの値を確認します。ファイアウォールルールがそのポートをブロックしている場合は、別のポートを使用してインスタンスを再作成できます。
- 不完全なまたは誤った IAM 設定 - Aurora DB インスタンスを IAM ベースの認証を使用するように作成している場合は、正しく設定されていることを確認します。詳細については、「 の IAM データベース認証」を参照してください。

Aurora DB 接続問題のトラブルシューティングの詳細については、「Amazon RDS DB インスタンスに接続できない」を参照してください。

「パラメータグループを使用する」

データベースパラメータは、データベースの設定方法を指定します。データベースパラメータでは、データベースに割り当てるメモリなどのリソースの量を指定できます。

DB インスタンスとAurora DB クラスターをパラメータグループに関連付けて、データベースの設定を管理します。Aurora は、デフォルト設定を使用してパラメータグループを定義します。カスタマイズした設定を使用して独自のパラメータグループを定義できます。

トピック

パラメータグループの概要

DB クラスターパラメータグループは、Aurora DB クラスター内のすべての DB インスタンスに適用されるエンジン設定値のコンテナとして機能します。例えば、Aurora 共有ストレージモデルでは、Aurora クラスター内のすべての DB インスタンスが `innodb_file_per_table` などのパラメータで同じ設定を使用することが要求されます。したがって、物理的なストレージレイアウトに影響するパラメータは、クラスターパラメータグループの一部です。DB クラスターパラメータグループには、インスタンスレベルのすべてのパラメータのデフォルト値も含まれています。

DB パラメータグループは、1 つ以上の DB インスタンスに適用されるエンジン設定値のコンテナとして機能します。DB パラメータグループは、Amazon RDS と Aurora の両方の DB インスタンスに適用されます。これらの構成設定が適用されるプロパティ (メモリバッファのサイズなど) は、Aurora クラスター内の DB インスタンス間で異なる場合があります。

トピック

- サポートされるパラメータとパラメータ値

デフォルトおよびカスタムパラメータグループ

DB パラメータグループを指定せずに DB インスタンスを作成すると、DB インスタンスはデフォルトの DB パラメータグループを使用します。同様に、DB クラスターパラメータグループを指定せずにAurora DB クラスターを作成すると、DB クラスターではデフォルトの DB クラスターパラメータグループが使用されます。デフォルトの各パラメータグループには、エンジン、コンピューティングクラス、およびインスタンスの割り当てストレージに基づいた、データベースエンジンのデフォルトと Amazon RDS システムのデフォルトが含まれています。

デフォルトのパラメータグループのパラメータ設定は変更できません。代わりに、以下を実行できます。

1. 新しいパラメータグループを作成します。
2. 必要なパラメータの設定を変更します。パラメータグループ内のすべての DB エンジンパラメータが変更できるわけではありません。
3. DB インスタンスまたは DB クラスターを変更して、新しいパラメータグループを関連付けます。

   DB クラスターまたは DB インスタンスの変更については、「Amazon Aurora DB クラスターの変更」を参照してください。

> **Note**
>
> カスタムパラメータグループを使用するように DB インスタンスを変更して、DB インスタンスを起動すると、RDS は起動プロセスの一環として DB インスタンスを自動的に再起動します。

RDS は、DB インスタンスの再起動後にのみ、変更された静的パラメータと動的パラメータを新しく関連付けられたパラメータグループに適用します。ただし、DB インスタンスに関連付けた後に DB パラメータグループの動的パラメータを変更すると、これらの変更は再起動せずに直ちに適用されます。DB パラメータグループの変更については、「Amazon Aurora DB クラスターの変更」を参照してください。

DB パラメータグループ内のパラメータを更新すると、このパラメータグループに関連付けられたすべての DB インスタンスに変更が適用されます。同様に、Aurora DB クラスターパラメータグループ

内のパラメータを更新すると、この DB クラスターパラメータグループに関連付けられたすべての Aurora DB クラスターに変更が適用されます。

パラメータグループを最初から作成したくない場合は、AWS CLI copy-db-parameter-group コマンドまたは copy-db-cluster-parameter-group コマンドを使用して、既存のパラメータグループをコピーできます。場合によっては、パラメータグループをコピーすると便利なことがあります。例えば、既存の パラメータグループのカスタムパラメータと値のほとんどを新しい パラメータグループに含めたい場合です。

静的および動的 DB クラスターパラメータ

DB クラスターパラメータは静的または動的に使用することができます。これらは次の点で異なります。

- 静的パラメータを変更して DB クラスターのパラメータグループを保存する場合、パラメータの変更は、関連付けられている各 DB クラスター内の DB インスタンスを手動で再起動した後に有効になります。AWS Management Console を使用して静的 DB クラスターパラメータ値を変更するときには、常に `ApplyMethod` として `pending-reboot` を使用します。
- 動的パラメータを変更すると、デフォルトでは、パラメータの変更は直ちに有効になり、再起動は不要です。コンソールを使用する場合、常に `ApplyMethod` として `immediate` を使用します。パラメータの変更を、関連付けられている DB クラスター内の DB インスタンスが再起動されるまで延期するには、AWS CLI または RDS API を使用します。パラメータを変更する場合は、`ApplyMethod` を `pending-reboot` に設定します。

AWS CLI を使用してパラメータ値を変更する方法については、「modify-db-cluster-parameter-group」を参照してください。RDS API を使用してパラメータ値を変更する方法については、「ModifyDBClusterParameterGroup」を参照してください。

DB クラスターに関連付けられている DB クラスターパラメータグループを変更した後、DB クラスター内の DB インスタンスを再起動します。再起動すると、DB クラスター内のすべての DB インスタンスに変更が適用されます。変更を適用するために DB クラスターの DB インスタンスを再起動する必要があるかどうかを判定するには、次の AWS CLI コマンドを実行します。

```
aws rds describe-db-clusters --db-cluster-identifier db_cluster_identifier
```

出力でプライマリ DB インスタンスの `DBClusterParameterGroupStatus` 値を確認します。値が `pending-reboot` の場合は、DB クラスターの DB インスタンスを再起動します。

静的および動的 DB インスタンスパラメータ

DB インスタンスパラメータは静的または動的に使用することができます。違いについては下記をご覧ください。

- 静的パラメータを変更して DB パラメータグループを保存すると、パラメータの変更は関連付けられている DB インスタンスを手動で再起動した後に有効になります。静的パラメータの場合、コンソールは常に `ApplyMethod` として `pending-reboot` を使用します。
- 動的パラメータを変更すると、デフォルトでは、パラメータの変更は直ちに有効になり、再起動は不要です。AWS Management Console を使用して DB インスタンスのパラメータ値を変更するときには、常に動的パラメータ向けの `ApplyMethod` として `immediate` を使用します。パラメータの変更を、関連付けられている DB インスタンスが再起動されるまで延期するには、AWS CLI または RDS API を使用します。パラメータを変更する場合は、`ApplyMethod` を `pending-reboot` に設定します。

AWS CLI を使用してパラメータ値を変更する方法については、「modify-db-parameter-group」を参照してください。RDS API を使用してパラメータ値を変更する方法については、「ModifyDBParameterGroup」を参照してください。

DB インスタンスが、その関連付けられた DB パラメータグループに対する最新の変更を使用していない場合、コンソールに DB パラメータグループの [pending-reboot] (再起動の保留中) のステータスが表示されます。このステータスにより、次回のメンテナンスウィンドウで自動的に再起動されることはありません。パラメータの最新の変更を DB インスタンスに適用するには、DB インスタンスを手動で再起動します。

文字セットパラメータ

DB クラスターを作成する前に、パラメータグループに含まれるデータベースの文字セットまたは照合に関するパラメータを設定します。また、その中にデータベースを作成する前にも行ってください。これにより、デフォルトのデータベースと新しいデータベースで、指定した文字セットと照合順序が使用されるようになります。文字セットまたは照合パラメータを変更した場合、パラメータの変更は既存のデータベースに適用されません。

一部の DB エンジンでは、`ALTER DATABASE` コマンドを使用して、既存のデータベースの文字セットまたは照合値を変更できます。次に例を示します。

```
ALTER DATABASE database_name CHARACTER SET character_set_name COLLATE collation;
```

データベースの文字セットまたは照合値の変更の詳細については、DB エンジンのドキュメントを参照してください。

サポートされるパラメータとパラメータ値

DB エンジンでサポートされているパラメータを決定するには、DB インスタンスまたは DB クラスターで使用される DB パラメータグループおよび DB クラスターパラメータグループのパラメータを表示します。詳細については、DB パラメータグループのパラメータ値を表示するおよびDB クラスターパラメータグループのパラメータ値を表示するを参照してください。

多くの場合、表現、数式、関数を使用して、整数およびブール型パラメータを指定することができます。関数には、数学的な口グ式を含めることができます。ただし、すべてのパラメータが、パラメータ値の表現、数式、関数をサポートしているわけではありません。詳細については、「DB パラメータの指定」を参照してください。

Aurora グローバルデータベースでは、Aurora クラスター別に異なる構成設定を指定できます。セカンダリクラスターをプライマリクラスターに昇格させる場合は、両者を同様の設定にして動作を一貫させてください。例えば、Aurora グローバルデータベースのすべてのクラスターでタイムゾーンと文字セットに同じ設定を使用します。

パラメータグループに不適切な設定のパラメータがあると、パフォーマンスが低下したりシステムが不安定になったり、予期しない悪影響が生じることがあります。データベースパラメータの変更時には常に注意が必要です。パラメータグループの変更前にはデータをバックアップしてください。テスト用 DB インスタンスまたは DB クラスターでパラメータグループの設定の変更を試してから、本番稼働用 DB インスタンスまたは DB クラスターにそれらの変更を適用してください。

DB クラスターパラメータグループを使用する

Amazon Aurora DB クラスターでは、DB クラスターパラメータグループが使用されます。次のセクションでは、DB クラスターパラメータグループの設定と管理について説明します。

トピック

- Amazon Aurora の DB クラスターパラメータと DB インスタンスパラメータ
- DB クラスターのパラメータグループの作成
- DB クラスターパラメータグループと DB クラスターの関連付け
- DB クラスターパラメータグループのパラメータの変更
- DB クラスターパラメータグループのパラメータのリセット
- DB クラスターのパラメータグループのコピー

- DB クラスターのパラメータグループのリスト化
- DB クラスターパラメータグループのパラメータ値を表示する
- DB クラスターパラメータグループの削除

Amazon Aurora の DB クラスターパラメータと DB インスタンスパラメータ

Aurora は、以下に示すように、2 レベルシステムの構成設定を使用します。

- DB クラスターパラメータグループのパラメータは、DB クラスター内のすべての DB インスタンスに適用されます。データは、Aurora 共有ストレージサブシステムに保存されます。このため、テーブルデータの物理レイアウトに関連するすべてのパラメータは、Aurora クラスター内のすべての DB インスタンスで同じにする必要があります。同様に、Aurora DB インスタンスはレプリケーションで接続されているため、レプリケーション設定のすべてのパラメータは Aurora クラスター全体で同じにする必要があります。
- DB パラメータグループのパラメータは、Aurora DB クラスター内の単一の DB インスタンスに適用されます。これらのパラメータは、同じ Aurora クラスター内の DB インスタンス間で変化させることができるメモリ使用量などの要素に関連しています。例えば、クラスターには、AWS インスタンスクラスが異なる DB インスタンスが含まれる場合がよくあります。

すべての Aurora クラスターは、DB クラスターパラメータグループに関連付けられます。このパラメータグループは、対応する DB エンジンのすべての設定値にデフォルト値を割り当てます。クラスターパラメータグループには、クラスターレベルとインスタンスレベル両方のパラメータのデフォルトも含まれています。プロビジョニングされたクラスターまたは Aurora Serverless v2 クラスター内の各 DB インスタンスは、その DB クラスターパラメータグループから設定を継承します。

各 DB インスタンスにも DB パラメータグループが関連付けられます。DB パラメータグループの値によって、クラスターパラメータグループのデフォルト値をオーバーライドできます。例えば、クラスター内の 1 つのインスタンスに問題が発生した場合、そのインスタンスにカスタム DB パラメータグループを割り当てることができます。カスタムパラメータグループには、デバッグまたはパフォーマンスチューニングに関連するパラメータとして特定の設定を含めることができます。

Aurora は、指定されたデータベースエンジンおよびバージョンに基づいて、クラスターまたは新しい DB インスタンスを作成すると、デフォルトのパラメータグループを割り当てます。カスタムパラメータグループを指定することもできます。これらのパラメータグループは自分で作成し、パラメータ値を編集できます。これらのカスタムパラメータグループは、作成時に指定できます。DB クラスターまたはインスタンスを後で変更して、カスタムパラメータグループを使用することもできます。

プロビジョニングされたインスタンスと Aurora Serverless v2 インスタンス化の場合、DB クラスターパラメータグループで変更した設定値は、DB パラメータグループのデフォルト値をオーバーライドします。DB パラメータグループ内の対応する値を編集すると、これらの値によって DB クラスターパラメータグループの設定が上書きされます。

変更した DB パラメータ設定は、構成設定を変更してデフォルト値に戻した場合でも、DB クラスターパラメータグループ値より優先されます。どのパラメータがオーバーライドされるかは、describe-db-parameters AWS CLI コマンドまたは DescribeDBParameters RDS API を使用して確認できます。`Source` フィールドには、該当するパラメータを変更した場合に、値 `user` が含まれます。DB クラスターパラメータグループの値が優先されるように、1 つ以上のパラメータをリセットするには、reset-db-parameter-group AWS CLI コンポーネントまたは ResetDBParameterGroup RDS API オペレーションを使用します。

DB クラスターと、Aurora で利用可能な DB インスタンスパラメータは、データベースエンジンの互換性に応じて異なります。

データベースエンジン	パラメータ
Aurora MySQL	「Aurora MySQL 設定パラメータ」を参照してください。 Aurora Serverless クラスターの詳細については、Aurora Serverless v2 のパラメータグループを使用する と Aurora Serverless v1 のパラメータグループ のその他の詳細を参照してください。
Aurora PostgreSQL	「Amazon Aurora PostgreSQL のパラメータ」を参照してください。 Aurora Serverless クラスターの詳細については、Aurora Serverless v2 のパラメータグループを使用する と Aurora Serverless v1 のパラメータグループ のその他の詳細を参照してください。

> **Note**
>
> Aurora Serverless v1 クラスターには、DB クラスターパラメータグループのみ関連付けられており、DB パラメータグループは関連付けられていません。Aurora Serverless v2 クラス

ターでは、すべての変更は、DB クラスターパラメータグループ内のカスタムパラメータに対して行います。
Aurora Serverless v2 は、DB クラスターパラメータグループと DB パラメータグループの両方を使用します。Aurora Serverless v2 では、ほぼすべての構成パラメータを変更できます。Aurora Serverless v2 は、一部の容量関連の構成パラメータの設定をオーバーライドして、Aurora Serverless v2 インスタンスがスケールダウンしたときにワークロードが中断されないようにします。
Aurora Serverless クラスターの構成設定と変更できる設定の詳細については、Aurora Serverless v2 のパラメータグループを使用する と Aurora Serverless v1 のパラメータグループ を参照してください。

DB クラスターのパラメータグループの作成

新しい DB クラスターパラメータグループは、AWS Management Console、AWS CLI、または RDS API を使って作成できます。

DB クラスターパラメータグループの作成後、その DB クラスターパラメータグループを使用する最初の DB クラスターが作成されるまで、5 分以上かかります。これにより、Amazon RDS は新しい DB クラスターによって使用される前に、パラメータグループを完全に作成することができます。DB クラスターパラメータグループが作成されたことを確認するには、Amazon RDS コンソールの [Parameter groups] (パラメータグループ) ページまたは describe-db-cluster-parameters コマンドを使用できます。

DB クラスターパラメータグループ名には、次の制限事項が適用されます。

- 名前は、1 ~ 255 の英字、数字、ハイフンである必要があります。

  デフォルトのパラメータグループ名には、`default.aurora-mysql5.7` のようなピリオドを含めることができます。ただし、カスタムパラメータグループ名にはピリオドを含めることはできません。
- 1 字目は文字である必要があります。
- 名前の最後にハイフンを使用したり、ハイフンを 2 つ続けて使用したりすることはできません。

コンソール

DB クラスターパラメータグループを作成するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[パラメータグループ] を選択します。
3. [Create parameter group] (パラメータグループの作成) を選択します。

   [パラメータグループの作成] ウィンドウが表示されます。
4. [パラメータグループファミリー] リストで、DB パラメータグループファミリーを選択します。
5. [タイプ] リストで、[DB クラスターのパラメータグループ] を選択します。
6. [グループ名] ボックスに、新しい DB クラスターパラメータグループの名前を入力します。
7. [説明] ボックスに、新しい DB クラスターパラメータグループの説明を入力します。
8. [Create] を選択します。

AWS CLI

DB クラスターのパラメータグループを作成するには、AWS CLI の `create-db-cluster-parameter-group` コマンドを使用します。

次の例では、Aurora MySQL バージョン 5.7 用に、mydbclusterparametergroup という名前で、「My new cluster parameter group」(新しいクラスターパラメータグループ) という説明の DB クラスターパラメータグループを作成しています。

以下の必須パラメータを含めます。

- `--db-cluster-parameter-group-name`
- `--db-parameter-group-family`
- `--description`

使用可能なすべてのパラメータグループファミリーを一覧表示するには、次のコマンドを使用します。

```
aws rds describe-db-engine-versions --query "DBEngineVersions[].DBParameterGroupFamily"
```

> **Note**
>
> 出力は重複が含まれます。

Example

Linux、macOS、Unix の場合:

```
aws rds create-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --db-parameter-group-family aurora-mysql5.7 \
    --description "My new cluster parameter group"
```

Windows の場合:

```
aws rds create-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --db-parameter-group-family aurora-mysql5.7 ^
    --description "My new cluster parameter group"
```

このコマンドでは、以下のような出力が生成されます。

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "mydbclusterparametergroup",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "My new cluster parameter group",
        "DBClusterParameterGroupArn": "arn:aws:rds:us-east-1:123456789012:cluster-
pg:mydbclusterparametergroup"
    }
}
```

RDS API

DB クラスターのパラメータグループを作成するには、RDS API の `CreateDBClusterParameterGroup` アクションを使用します。

以下の必須パラメータを含めます。

- `DBClusterParameterGroupName`

- `DBParameterGroupFamily`
- `Description`

DB クラスターパラメータグループと DB クラスターの関連付け

カスタマイズした設定を使用して、独自の DB クラスターパラメータグループを作成できます。AWS Management Console、AWS CLI、または RDS API を使用して、DB クラスターパラメータグループを DB クラスターに関連付けることができます。DB クラスターを作成または変更するときに行うことができます。

DB クラスターのパラメータグループの作成については、DB クラスターのパラメータグループの作成 を参照してください。DB クラスターの作成方法については、Amazon Aurora DB クラスターの作成 を参照してください。DB クラスターの変更については、「Amazon Aurora DB クラスターの変更」を参照してください。

> **Note**
>
> Aurora PostgreSQL 15.2、14.7、13.10、12.14、およびすべての 11 バージョンでは、DB クラスターに関連付けられている DB クラスターパラメータグループを変更する際、各レプリカインスタンスを再起動して変更を適用します。
> 変更を適用するために DB クラスターのプライマリ DB インスタンスを再起動する必要があるかどうかを判定するには、以下の AWS CLI コマンドを実行します。
> `aws rds describe-db-clusters --db-cluster-identifier db_cluster_identifier`
> 出力でプライマリ DB インスタンスの `DBClusterParameterGroupStatus` 値を確認します。値が `pending-reboot` の場合は、DB クラスターのプライマリ DB インスタンスを再起動します。

コンソール

DB クラスターパラメータグループを DB クラスターに関連付けるには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択し、変更する DB クラスターを選択します。
3. [Modify] を選択します。[DB クラスターの変更] ページが表示されます。

4. DB クラスターパラメータグループの設定を変更します。
5. [Continue] を選択して、変更の概要を確認します。

   [Scheduling of modifications (変更のスケジュール)] 設定に関係なく、変更はすぐに適用されます。
6. 確認ページで、変更内容を確認します。正しい場合は、[クラスターの変更] を選択して変更を保存します。

   または、[Back] を選択して変更を編集するか、[Cancel] を選択して変更をキャンセルします。

AWS CLI

DB クラスターのパラメータグループと DB クラスターを関連付けるには、以下のオプションを指定しながら AWS CLI の `modify-db-cluster` コマンドを使用します。

- `--db-cluster-name`
- `--db-cluster-parameter-group-name`

次の例では、`mydbclpg` DB パラメータグループを `mydbcluster` DB クラスターに関連付けます。

Example

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --db-cluster-parameter-group-name mydbclpg
```

Windows の場合:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --db-cluster-parameter-group-name mydbclpg
```

RDS API

DB クラスターパラメータグループと DB クラスターを関連付けるには、以下のパラメータを指定しながら RDS API の `ModifyDBCluster` オペレーションを使用します。

- `DBClusterIdentifier`

- `DBClusterParameterGroupName`

DB クラスターパラメータグループのパラメータの変更

ユーザーが作成した DB クラスターパラメータグループのパラメータ値は変更できます。デフォルト DB クラスターパラメータグループのパラメータ値は変更できません。ユーザー定義の DB クラスターパラメータグループのパラメータの変更は、その DB クラスターパラメータグループに関連付けられたすべての DB クラスターに適用されます。

コンソール

DB クラスターパラメータグループを変更するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[パラメータグループ] を選択します。
3. リストで、変更するパラメータグループを選択します。
4. [Parameter group actions (パラメータグループのアクション)] で、[編集] を選択します。
5. 変更するパラメータの値を変更します。ダイアログボックスの右上にある矢印キーを使用して、パラメータをスクロールできます。

   デフォルトパラメータグループの値を変更することはできません。
6. [Save changes] (変更の保存) をクリックします。
7. クラスター内のプライマリ (ライター) DB インスタンスを再起動して、変更を適用します。
8. 次に、リーダー DB インスタンスを再起動して、変更を適用します。

AWS CLI

DB クラスターパラメータグループを変更するには、以下の必須パラメータを指定しながら AWS CLI の `modify-db-cluster-parameter-group` コマンドを使用します。

- `--db-cluster-parameter-group-name`
- `--parameters`

以下の例では、mydbclusterparametergroup という名前の DB クラスターパラメータグループの `server_audit_logging` と `server_audit_logs_upload` の値を変更しています。

Example

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" \

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

Windows の場合:

```
aws rds modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" ^

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

このコマンドでは、以下のような出力が生成されます。

```
DBCLUSTERPARAMETERGROUP  mydbclusterparametergroup
```

RDS API

DB クラスターのパラメータグループを変更するには、以下の必須パラメータを指定ながら RDS API の ModifyDBClusterParameterGroup コマンドを使用します。

- DBClusterParameterGroupName
- Parameters

DB クラスターパラメータグループのパラメータのリセット

顧客が作成した DB クラスターパラメータグループの、デフォルト値のパラメータはリセットできます。ユーザー定義の DB クラスターパラメータグループのパラメータの変更は、その DB クラスターパラメータグループに関連付けられたすべての DB クラスターに適用されます。

> **Note**
>
> デフォルトの DB クラスターパラメータグループでは、パラメータは常にデフォルト値に設定されます。

コンソール

DB クラスターパラメータグループのパラメータをデフォルト値にリセットするには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[パラメータグループ] を選択します。
3. リストからパラメータグループを選択します。
4. [Parameter group actions (パラメータグループのアクション)] で、[編集] を選択します。
5. デフォルト値にリセットするパラメータを選択します。ダイアログボックスの右上にある矢印キーを使用して、パラメータをスクロールできます。

   デフォルトのパラメータグループの値をリセットすることはできません。
6. リセットを選択し、パラメータをリセットを選択して確定します。
7. DB クラスター内のプライマリ DB インスタンスを再起動して、DB クラスター内のすべての DB インスタンスに変更を適用します。

AWS CLI

DB クラスターのパラメータグループにおいて、パラメータをデフォルト値にリセットするには、以下の `--db-cluster-parameter-group-name` オプション (必須) を指定しながら AWS CLI の `reset-db-cluster-parameter-group` コマンドを使用します。

DB クラスターパラメータグループのパラメータをすべてリセットするには、`--reset-all-parameters` オプションを指定します。特定のパラメータをリセットするには、`--parameters`オプションを指定します。

次の例では、mydbparametergroupという名前の DB パラメータグループ内のすべてのパラメータをデフォルト値にリセットします。

Example

Linux、macOS、Unix の場合:

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbparametergroup \
    --reset-all-parameters
```

Windows の場合:

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbparametergroup ^
    --reset-all-parameters
```

以下の例では、mydbclusterparametergroup という名前の DB クラスターパラメータグループにある server_audit_logging と server_audit_logs_upload をデフォルト値にリセットしています。

Example

Linux、macOS、Unix の場合:

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --parameters "ParameterName=server_audit_logging,ApplyMethod=immediate" \
                 "ParameterName=server_audit_logs_upload,ApplyMethod=immediate"
```

Windows の場合:

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" ^

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

このコマンドでは、以下のような出力が生成されます。

```
DBClusterParameterGroupName  mydbclusterparametergroup
```

RDS API

DB クラスターパラメータグループのパラメータをデフォルト値にリセットするには、以下の必須パラメータを指定して、RDS API `ResetDBClusterParameterGroup`コマンドを使用します。`DBClusterParameterGroupName`

DB クラスターパラメータグループのパラメータをすべてリセットするには、`ResetAllParameters` パラメータを `true` に設定します。特定のパラメータをリセットするには、`Parameters`パラメータを指定します。

DB クラスターのパラメータグループのコピー

作成したカスタム DB クラスターパラメータグループをコピーできます。パラメータグループのコピーは、作成済みの DB クラスターパラメータグループがあり、そのグループの多くのカスタムパラメータと値を新しい DB クラスターパラメータグループに含める必要がある場合に便利な方法です。DB クラスターパラメータグループをコピーするには、AWS CLI copy-db-cluster-parameter-group コマンド、または RDS API CopyDBClusterParameterGroup オペレーションを使用できます。

DB クラスターパラメータグループをコピーした後で、この DB クラスターパラメータグループを使用する DB クラスターを作成するまで、5 分以上かかります。これにより、Amazon RDS は新しい DB クラスターによって使用される前に、パラメータグループを完全にコピーすることができます。DB クラスターパラメータグループが作成されたことを確認するには、Amazon RDS コンソールの [Parameter groups] (パラメータグループ) ページまたは describe-db-cluster-parameters コマンドを使用できます。

> **Note**
>
> デフォルトのパラメータグループをコピーすることはできません。ただし、デフォルトのパラメータグループに基づく新しいパラメータグループを作成できます。
> DB クラスターパラメータグループを別の AWS アカウント または AWS リージョン にコピーすることはできません。

コンソール

DB クラスターパラメータグループをコピーするには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[パラメータグループ] を選択します。

3. リストで、コピーするカスタムパラメータグループを選択します。
4. [Parameter group actions (パラメータグループのアクション)] で、[コピー] を選択します。
5. [新規の DB パラメータグループの識別子] に、新しいパラメータグループの名前を入力します。
6. [説明] に、新しいパラメータグループの説明を入力します。
7. [Copy (コピー)] を選択します。

AWS CLI

DB クラスターのパラメータグループをコピーするには、以下の必須パラメータを指定しながら AWS CLI の copy-db-cluster-parameter-group コマンドを使用します。

- `--source-db-cluster-parameter-group-identifier`
- `--target-db-cluster-parameter-group-identifier`
- `--target-db-cluster-parameter-group-description`

次の例は、DB クラスターパラメータグループ mygroup2 のコピーである mygroup1 という名前の新しい DB クラスターパラメータグループを作成します。

Example

Linux、macOS、Unix の場合:

```
aws rds copy-db-cluster-parameter-group \
    --source-db-cluster-parameter-group-identifier mygroup1 \
    --target-db-cluster-parameter-group-identifier mygroup2 \
    --target-db-cluster-parameter-group-description "DB parameter group 2"
```

Windows の場合:

```
aws rds copy-db-cluster-parameter-group ^
    --source-db-cluster-parameter-group-identifier mygroup1 ^
    --target-db-cluster-parameter-group-identifier mygroup2 ^
    --target-db-cluster-parameter-group-description "DB parameter group 2"
```

RDS API

DB クラスターパラメータグループをコピーするには、以下の必須パラメータを指定して、RDS API の CopyDBClusterParameterGroup オペレーションを使用します。

- `SourceDBClusterParameterGroupIdentifier`
- `TargetDBClusterParameterGroupIdentifier`
- `TargetDBClusterParameterGroupDescription`

DB クラスターのパラメータグループのリスト化

AWS アカウント用に作成した DB クラスターパラメータグループを一覧表示できます。

> **Note**
>
> デフォルトのパラメータグループは、特定の DB エンジンとバージョンの DB クラスターを作成するときに、デフォルトのパラメータテンプレートから自動的に作成されます。これらのデフォルトのパラメータグループには、優先されるパラメータ設定が含まれています。これを変更することはできません。カスタムパラメータグループを作成する場合、パラメータ設定を変更できます。

コンソール

AWS アカウントのすべての DB クラスターパラメータグループを一覧表示するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[パラメータグループ] を選択します。

   DB クラスターパラメータグループは、[Type] (タイプ) が [DB cluster parameter group] (DB クラスターパラメータグループ) のリストに表示されます。

AWS CLI

AWS アカウントにある、すべての DB クラスターのパラメータグループを一覧表示するには、AWS CLI の `describe-db-cluster-parameter-groups` コマンドを使用します。

Example

以下の例では、AWS アカウントに使用できるすべての DB クラスターパラメータグループを一覧表示しています。

```
aws rds describe-db-cluster-parameter-groups
```

次の例は、mydbclusterparametergroup パラメータグループを表しています。

Linux、macOS、Unix の場合:

```
aws rds describe-db-cluster-parameter-groups \
    --db-cluster-parameter-group-name mydbclusterparametergroup
```

Windows の場合:

```
aws rds describe-db-cluster-parameter-groups ^
    --db-cluster-parameter-group-name mydbclusterparametergroup
```

このコマンドでは次のようなレスポンスが返されます。

```
{
    "DBClusterParameterGroups": [
        {
            "DBClusterParameterGroupName": "mydbclusterparametergroup",
            "DBParameterGroupFamily": "aurora-mysql5.7",
            "Description": "My new cluster parameter group",
            "DBClusterParameterGroupArn": "arn:aws:rds:us-east-1:123456789012:cluster-
pg:mydbclusterparametergroup"
        }
    ]
}
```

RDS API

AWS アカウントにある、すべての DB クラスターのパラメータグループを一覧表示するには、RDS API の DescribeDBClusterParameterGroups アクションを使用します。

DB クラスターパラメータグループのパラメータ値を表示する

DB クラスターパラメータグループのすべてのパラメータとそれらの値のリストを取得できます。

コンソール

DB クラスターパラメータグループのパラメータ値を表示するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[パラメータグループ] を選択します。

   DB クラスターパラメータグループは、[Type] (タイプ) が [DB cluster parameter group] (DB クラスターパラメータグループ) のリストに表示されます。
3. パラメータを一覧表示する DB クラスターパラメータグループの名前を選択します。

AWS CLI

DB クラスターのパラメータグループについて、そのパラメータ値を表示するには、以下の必須パラメータを指定しながら AWS CLI の `describe-db-cluster-parameters` コマンドを使用します。

- `--db-cluster-parameter-group-name`

Example

以下の例では、JSON 形式の mydbclusterparametergroup という名前の DB クラスターパラメータグループのパラメータとその値を一覧表示しています。

このコマンドでは次のようなレスポンスが返されます。

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-
name mydbclusterparametergroup
```

```
{
    "Parameters": [
        {
            "ParameterName": "allow-suspicious-udfs",
            "Description": "Controls whether user-defined functions that have only an
 xxx symbol for the main function can be loaded",
            "Source": "engine-default",
            "ApplyType": "static",
            "DataType": "boolean",
            "AllowedValues": "0,1",
```

```
            "IsModifiable": false,
            "ApplyMethod": "pending-reboot",
            "SupportedEngineModes": [
                "provisioned"
            ]
        },
        {
            "ParameterName": "aurora_binlog_read_buffer_size",
            "ParameterValue": "5242880",
            "Description": "Read buffer size used by master dump thread when the switch
 aurora_binlog_use_large_read_buffer is ON.",
            "Source": "engine-default",
            "ApplyType": "dynamic",
            "DataType": "integer",
            "AllowedValues": "8192-536870912",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot",
            "SupportedEngineModes": [
                "provisioned"
            ]
        },

...
```

RDS API

DB クラスターパラメータグループのパラメータ値を表示するには、以下の必須パラメータを指定して、RDS API の DescribeDBClusterParameters コマンドを使用します。

- DBClusterParameterGroupName

場合によっては、パラメータに指定できる値が表示されないことがあります。これらは常にソースがデータベースエンジンのデフォルトであるパラメータです。

これらのパラメータの値を表示するには、次の SQL ステートメントを実行します。

- MySQL:

```
-- Show the value of a particular parameter
mysql$ SHOW VARIABLES LIKE '%parameter_name%';

-- Show the values of all parameters
```

```
mysql$ SHOW VARIABLES;
```

- PostgreSQL:

```
-- Show the value of a particular parameter
postgresql=> SHOW parameter_name;

-- Show the values of all parameters
postgresql=> SHOW ALL;
```

DB クラスターパラメータグループの削除

DB クラスターパラメータグループは、AWS Management Console、AWS CLI、または RDS API を使用して削除できます。DB クラスターパラメータグループのパラメータグループを削除できるのは、DB クラスターに関連付けられていない場合のみです。

コンソール

パラメータグループを削除するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、パラメータグループ を選択します。

   パラメータグループが一覧表示されます。
3. 削除する DB クラスターパラメータグループの名前を選択します。
4. [Actions (アクション)] を選択してから [Delete (削除)] を選択します。
5. パラメータグループ名を確認して、[削除] を選択します。

AWS CLI

DB クラスターのパラメータグループを削除するには、AWS CLI の `delete-db-cluster-parameter-group` コマンドを使用して、次の必須パラメータを指定します。

- `--db-parameter-group-name`

Example

次の例では、mydbparametergroup という名前の DB クラスターパラメータグループを削除します。

```
aws rds delete-db-cluster-parameter-group --db-parameter-group-name mydbparametergroup
```

RDS API

DB クラスターのパラメータグループを削除するには、RDS API の DeleteDBClusterParameterGroup コマンドを使用して、次の必須パラメータを指定します。

- DBParameterGroupName

DB インスタンスでの DB パラメータグループの使用

DB インスタンスでは DB パラメータグループが使用されます。次のセクションでは、DB インスタンスパラメータグループの設定と管理について説明します。

トピック

- DB パラメータグループを作成する
- DB パラメータグループを DB インスタンスに関連付ける
- DB パラメータグループのパラメータの変更
- DB パラメータグループのパラメータをデフォルト値にリセットします
- DB パラメータグループをコピーする
- DB パラメータグループを一覧表示する
- DB パラメータグループのパラメータ値を表示する
- DB パラメータグループの削除

DB パラメータグループを作成する

新しい DB パラメータグループは、AWS Management Console、AWS CLI、または RDS API を使って作成できます。

DB パラメータグループ名には、次の制限事項が適用されます。

- 名前は、1 ~ 255 の英字、数字、ハイフンである必要があります。

  デフォルトのパラメータグループ名には、default.mysql8.0 のようなピリオドを含めることができます。ただし、カスタムパラメータグループ名にはピリオドを含めることはできません。

- 1 字目は文字である必要があります。
- 名前の最後にハイフンを使用したり、ハイフンを 2 つ続けて使用したりすることはできません。

コンソール

DB パラメータグループを作成するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、パラメータグループ を選択します。
3. [Create parameter group] (パラメータグループの作成) を選択します。
4. [パラメータグループ名] に、新しい DB パラメータグループの名前を入力します。
5. [説明] に、新しい DB パラメータグループの説明を入力します。
6. [エンジンタイプ] で DB エンジンを選択します。
7. [パラメータグループファミリー] で、DB パラメータグループファミリーを選択します。
8. 該当する場合、[タイプ] で [DB パラメータグループ] を選択します。
9. [Create] (作成) を選択します。

AWS CLI

DB パラメータグループを作成するには、AWS CLI の `create-db-parameter-group` コマンドを使用します。以下の例では、MySQL バージョン 8.0 用に、mydbparametergroup という名前で、「My new parameter group」という説明の新しい DB パラメータグループを作成しています。

以下の必須パラメータを含めます。

- `--db-parameter-group-name`
- `--db-parameter-group-family`
- `--description`

使用可能なすべてのパラメータグループファミリーを一覧表示するには、次のコマンドを使用します。

```
aws rds describe-db-engine-versions --query "DBEngineVersions[].DBParameterGroupFamily"
```

> **Note**
>
> 出力は重複が含まれます。

Example

Linux、macOS、Unix の場合:

```
aws rds create-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --db-parameter-group-family aurora-mysql5.7 \
    --description "My new parameter group"
```

Windows の場合:

```
aws rds create-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --db-parameter-group-family aurora-mysql5.7 ^
    --description "My new parameter group"
```

このコマンドでは、以下のような出力が生成されます。

```
DBPARAMETERGROUP  mydbparametergroup  aurora-mysql5.7  My new parameter group
```

RDS API

DB パラメータグループを作成するには、RDS API の `CreateDBParameterGroup` オペレーションを使用します。

以下の必須パラメータを含めます。

- `DBParameterGroupName`
- `DBParameterGroupFamily`
- `Description`

DB パラメータグループを DB インスタンスに関連付ける

カスタマイズした設定を使用して、独自の DB パラメータグループを作成できます。AWS Management Console、AWS CLI、または RDS API を使用して、DB パラメータグループを DB イ

ンスタンスに関連付けることができます。DB インスタンスを作成または変更するときに、これを行うことができます。

DB パラメータグループの作成については、DB パラメータグループを作成する を参照してください。DB インスタンスの変更については、「DB クラスター内の DB インスタンスの変更」を参照してください。

> **Note**
>
> 新しい DB パラメータグループを DB インスタンスに関連付ける場合、変更された静的パラメータと動的パラメータは、DB インスタンスが再起動された後にのみ適用されます。ただし、DB インスタンスに関連付けた後に DB パラメータグループの動的パラメータを変更すると、これらの変更は再起動せずに直ちに適用されます。

コンソール

DB パラメータグループを DB インスタンスに関連付けるには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[データベース] を選択し、変更する DB インスタンスを選択します。
3. [Modify] を選択します。Modify DB instance ページが表示されます。
4. DB パラメータグループの設定を変更します。
5. [Continue] を選択して、変更の概要を確認します。
6. (オプション) 変更をすぐに適用するには、[Apply immediately] (すぐに適用) を選択します。このオプションを選択すると、停止状態になる場合があります。
7. 確認ページで、変更内容を確認します。正しい場合は、[Modify DB Instance (DB インスタンスを変更)] を選択して変更を保存します。

   または、[戻る] を選択して変更を編集するか、[キャンセル] を選択して変更をキャンセルします。

AWS CLI

DB パラメータグループを DB インスタンスに関連付けるには、以下のオプションを指定しながら AWS CLI の `modify-db-instance` コマンドを使用します。

- `--db-instance-identifier`
- `--db-parameter-group-name`

次の例では、mydbpg DB パラメータグループを database-1 DB インスタンスに関連付けます。`--apply-immediately` を使用すると変更はすぐに適用されます。`--no-apply-immediately` を使用して、次のメンテナンス時間中に変更を適用します。

Example

Linux、macOS、Unix の場合:

```
aws rds modify-db-instance \
    --db-instance-identifier database-1 \
    --db-parameter-group-name mydbpg \
    --apply-immediately
```

Windows の場合:

```
aws rds modify-db-instance ^
    --db-instance-identifier database-1 ^
    --db-parameter-group-name mydbpg ^
    --apply-immediately
```

RDS API

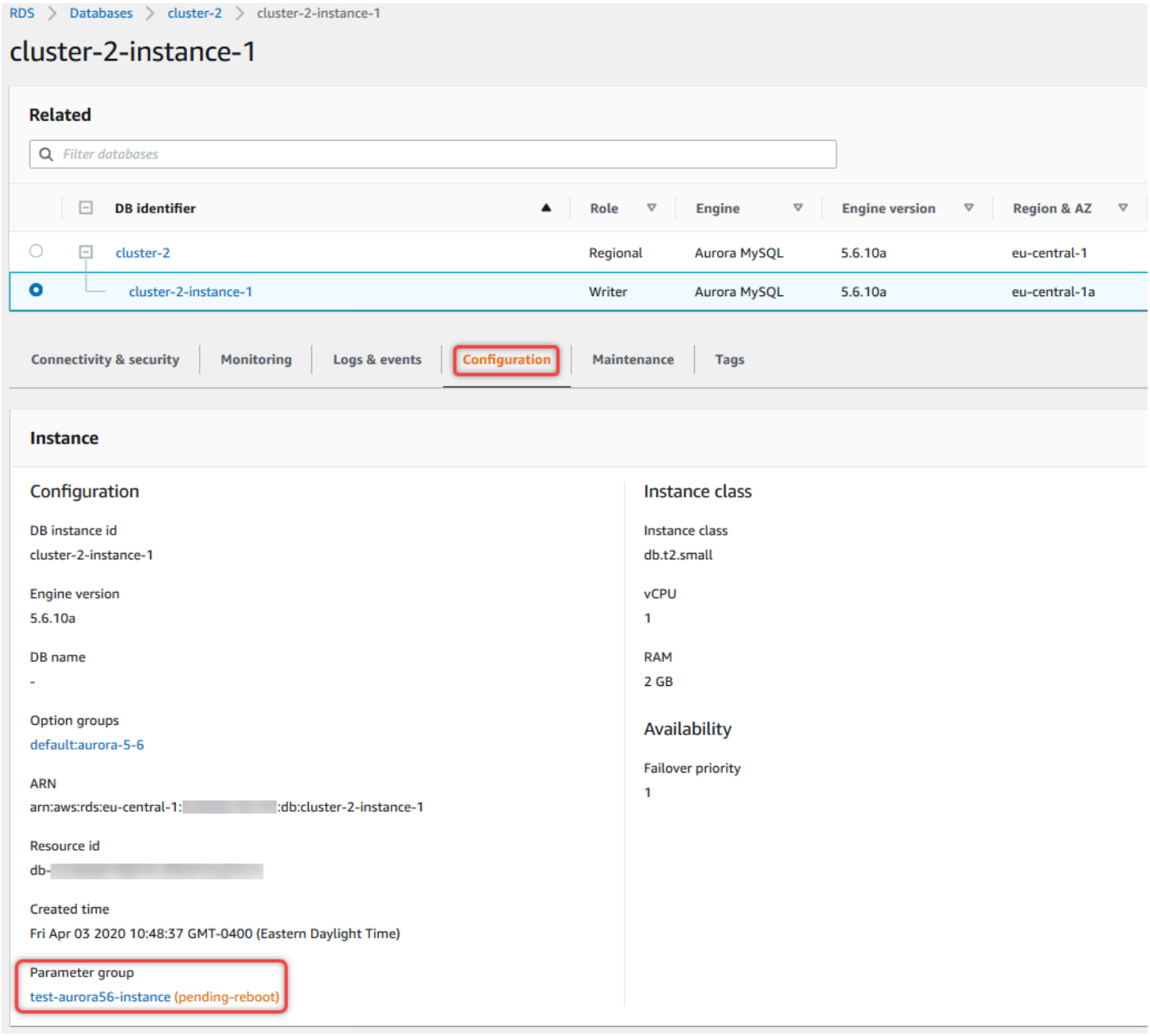
DB パラメータグループを DB インスタンスに関連付けるには、以下のパラメータを指定しながら RDS API の `ModifyDBInstance` オペレーションを使用します。

- `DBInstanceName`
- `DBParameterGroupName`

DB パラメータグループのパラメータの変更

ユーザー定義の DB パラメータグループのパラメータ値は変更できますが、デフォルトの DB パラメータグループのパラメータ値を変更することはできません。ユーザー定義の DB パラメータグループのパラメータの変更は、その DB パラメータグループに関連付けられたすべての DB インスタンスに適用されます。

一部のパラメータへの変更は、再起動せずに直ちに DB インスタンスに適用されます。他のパラメータの変更は、DB インスタンスの再起動後にのみ適用されます。DB インスタンスに関連付けられている DB パラメータグループのステータスは、RDS コンソールの [設定] タブに表示されます。例えば、DB インスタンスがその関連付けられた DB パラメータグループに対する最新の変更を使用していないとします。その場合、RDS コンソールは、DB パラメータグループのステータスを [pending-reboot] (再起動の保留中) と表示します。パラメータの最新の変更を DB インスタンスに適用するには、DB インスタンスを手動で再起動します。



コンソール

DB パラメータグループ内のパラメータを変更するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、パラメータグループ を選択します。
3. リストで、変更するパラメータグループの名前を選択します。
4. [Parameter group actions (パラメータグループのアクション)] で、[編集] を選択します。
5. 変更するパラメータの値を変更します。ダイアログボックスの右上にある矢印キーを使用して、パラメータをスクロールできます。

   デフォルトパラメータグループの値を変更することはできません。
6. [Save changes] (変更を保存) をクリックします。

AWS CLI

DB パラメータグループを変更するには、次の必須オプションを指定して、AWS CLI の `modify-db-parameter-group` コマンドを使用します。

- `--db-parameter-group-name`
- `--parameters`

以下の例では、mydbparametergroup という名前の DB パラメータグループの `max_connections` と `max_allowed_packet` の値を変更しています。

Example

Linux、macOS、Unix の場合:

```
aws rds modify-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --parameters
 "ParameterName=max_connections,ParameterValue=250,ApplyMethod=immediate" \

 "ParameterName=max_allowed_packet,ParameterValue=1024,ApplyMethod=immediate"
```

Windows の場合:

```
aws rds modify-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --parameters
 "ParameterName=max_connections,ParameterValue=250,ApplyMethod=immediate" ^

 "ParameterName=max_allowed_packet,ParameterValue=1024,ApplyMethod=immediate"
```

このコマンドでは、以下のような出力が生成されます。

```
DBPARAMETERGROUP  mydbparametergroup
```

RDS API

DB パラメータグループを変更するには、以下の必須パラメータを指定しながら RDS API の ModifyDBParameterGroup オペレーションを使用します。

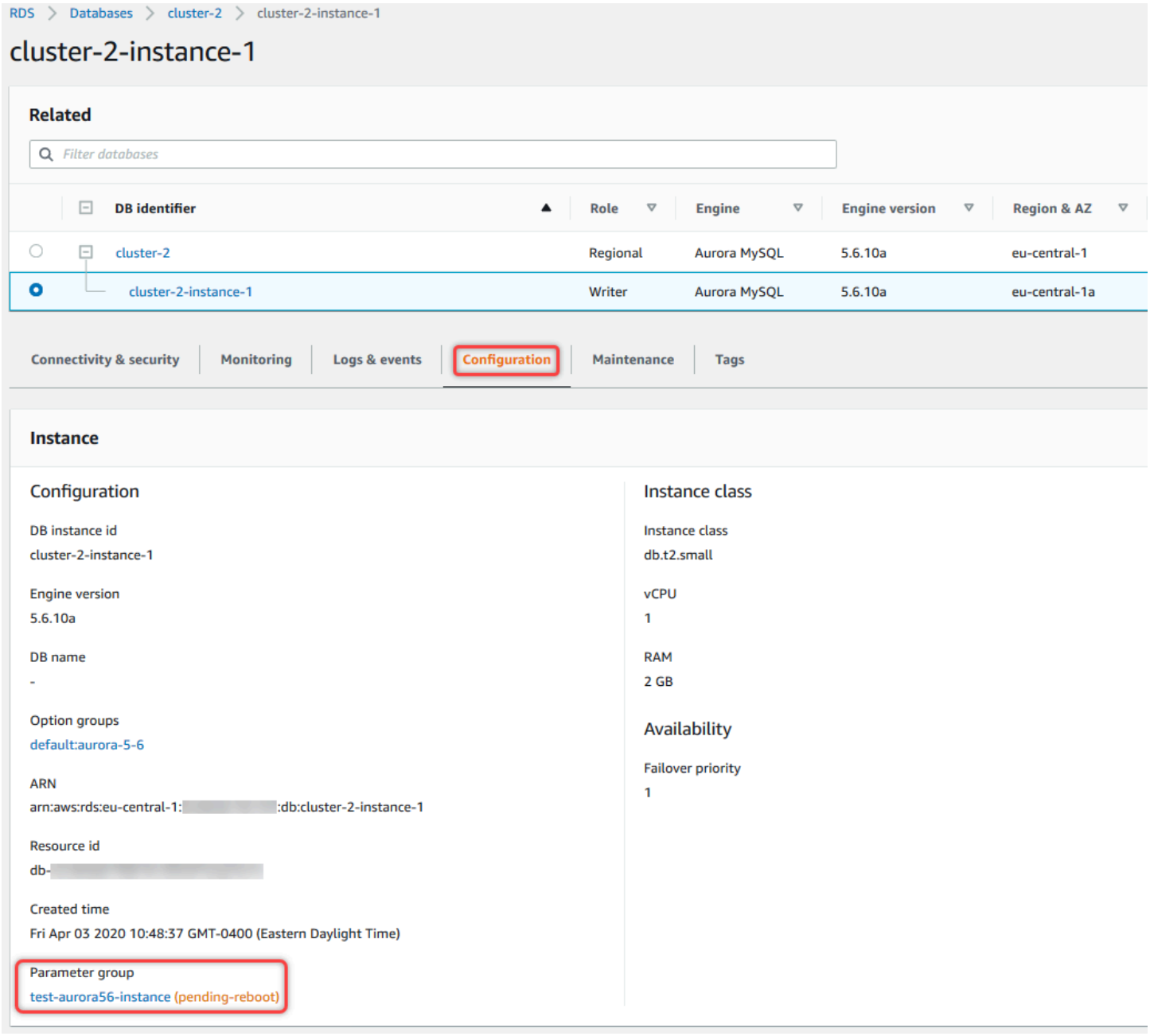
- DBParameterGroupName
- Parameters

DB パラメータグループのパラメータをデフォルト値にリセットします

顧客が作成した DB パラメータグループのパラメータ値をデフォルト値にリセットできます。ユーザー定義の DB パラメータグループのパラメータの変更は、その DB パラメータグループに関連付けられたすべての DB インスタンスに適用されます。

コンソールを使用すると、特定のパラメータをデフォルト値にリセットできます。ただし、DB パラメータグループのすべてのパラメータを一度にリセットするのは簡単ではありません。AWS CLI または RDS API を使用すると、特定のパラメータをデフォルト値にリセットできます。DB パラメータグループのすべてのパラメータを一度にリセットすることもできます。

一部のパラメータへの変更は、再起動せずに直ちに DB インスタンスに適用されます。他のパラメータの変更は、DB インスタンスの再起動後にのみ適用されます。DB インスタンスに関連付けられている DB パラメータグループのステータスは、RDS コンソールの [設定] タブに表示されます。例えば、DB インスタンスがその関連付けられた DB パラメータグループに対する最新の変更を使用していないとします。その場合、RDS コンソールは、DB パラメータグループのステータスを [pending-reboot] (再起動の保留中) と表示します。パラメータの最新の変更を DB インスタンスに適用するには、DB インスタンスを手動で再起動します。



> **Note**
>
> デフォルトの DB パラメータグループでは、パラメータは常にデフォルト値に設定されます。

コンソール

DB パラメータグループのパラメータをデフォルト値にリセットするには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、パラメータグループ を選択します。
3. リストからパラメータグループを選択します。
4. [Parameter group actions (パラメータグループのアクション)] で、[編集] を選択します。
5. デフォルト値にリセットするパラメータを選択します。ダイアログボックスの右上にある矢印キーを使用して、パラメータをスクロールできます。

   デフォルトのパラメータグループの値をリセットすることはできません。
6. リセットを選択し、パラメータをリセットを選択して確定します。

AWS CLI

DB パラメータグループの一部またはすべてのパラメータをリセットするには、AWS CLI`reset-db-parameter-group`コマンドに必須オプション`--db-parameter-group-name`を指定します。

DB パラメータグループのすべてのパラメータをリセットするには、`--reset-all-parameters`オプションを指定します。特定のパラメータをリセットするには、`--parameters`オプションを指定します。

次の例では、mydbparametergroup という名前の DB パラメータグループ内のすべてのパラメータをデフォルト値にリセットします。

Example

Linux、macOS、Unix の場合:

```
aws rds reset-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --reset-all-parameters
```

Windows の場合:

```
aws rds reset-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
```

```
    --reset-all-parameters
```

次の例では、mydbparametergroup という名前の DB パラメータグループのおよびmax_connectionsオプションmax_allowed_packetをデフォルト値にリセットします。

Example

Linux、macOS、Unix の場合:

```
aws rds reset-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --parameters "ParameterName=max_connections,ApplyMethod=immediate" \
                 "ParameterName=max_allowed_packet,ApplyMethod=immediate"
```

Windows の場合:

```
aws rds reset-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --parameters "ParameterName=max_connections,ApplyMethod=immediate" ^
                 "ParameterName=max_allowed_packet,ApplyMethod=immediate"
```

このコマンドでは、以下のような出力が生成されます。

```
DBParameterGroupName  mydbparametergroup
```

RDS API

DB パラメータグループ内のパラメータをデフォルト値にリセットするには、RDS API ResetDBParameterGroup コマンドに必須パラメータ DBParameterGroupName を指定します。

DB パラメータグループのすべてのパラメータをリセットするには、ResetAllParametersパラメータをtrueに設定します。特定のパラメータをリセットするには、Parametersパラメータを指定します。

DB パラメータグループをコピーする

作成したカスタム DB パラメータグループをコピーできます。パラメータグループのコピーは便利なソリューションです。例としては、作成済みの DB パラメータグループがあり、そのほとんどのカスタムパラメータと値を、新しい DB パラメータグループに含める場合です。AWS Management Console を使用して、DB パラメータグループをコピーできます。AWS CLI copy-db-parameter-group コマンド、または RDS API CopyDBParameterGroup オペレーションも使用できます。

DB パラメータグループをコピーした後で、その DB パラメータグループをデフォルトのパラメータグループとして使用する最初の DB インスタンスを作成するまで、最低 5 分待ちます。これにより、パラメータグループを使用する前に、Amazon RDS はコピーアクションが完全に終了できます。これは、DB インスタンスのデフォルトのデータベースを作成する際に不可欠となるパラメータで特に重要です。例としては、`character_set_database` パラメータで定義するデフォルトのデータベースの文字セットが挙げられます。DB パラメータグループが作成されたことを確認するには、Amazon RDS コンソールの [パラメータグループ] オプション、または describe-db-parameters コマンドを使用します。

> **Note**
>
> デフォルトのパラメータグループをコピーすることはできません。ただし、デフォルトのパラメータグループに基づく新しいパラメータグループを作成できます。
> DB パラメータグループを別の AWS アカウント または AWS リージョン にコピーすることはできません。

コンソール

DB パラメータグループをコピーするには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、パラメータグループ を選択します。
3. リストで、コピーするカスタムパラメータグループを選択します。
4. [Parameter group actions (パラメータグループのアクション)] で、[コピー] を選択します。
5. [新規の DB パラメータグループの識別子] に、新しいパラメータグループの名前を入力します。
6. [説明] に、新しいパラメータグループの説明を入力します。
7. [Copy (コピー)] を選択します。

AWS CLI

DB パラメータグループをコピーするには、次の必須オプションを指定しながら AWS CLI の `copy-db-parameter-group` コマンドを使用します。

- `--source-db-parameter-group-identifier`
- `--target-db-parameter-group-identifier`

- `--target-db-parameter-group-description`

次の例は、DB パラメータグループ mygroup2 のコピーである mygroup1 という名前の新しい DB パラメータグループを作成します。

Example

Linux、macOS、Unix の場合:

```
aws rds copy-db-parameter-group \
    --source-db-parameter-group-identifier mygroup1 \
    --target-db-parameter-group-identifier mygroup2 \
    --target-db-parameter-group-description "DB parameter group 2"
```

Windows の場合:

```
aws rds copy-db-parameter-group ^
    --source-db-parameter-group-identifier mygroup1 ^
    --target-db-parameter-group-identifier mygroup2 ^
    --target-db-parameter-group-description "DB parameter group 2"
```

RDS API

DB パラメータグループをコピーするには、以下の必須パラメータを指定して、RDS API の CopyDBParameterGroup オペレーションを使用します。

- `SourceDBParameterGroupIdentifier`
- `TargetDBParameterGroupIdentifier`
- `TargetDBParameterGroupDescription`

DB パラメータグループを一覧表示する

AWS アカウント用に作成した DB パラメータグループを一覧表示できます。

> **Note**
>
> デフォルトのパラメータグループは、特定の DB エンジンとバージョンの DB インスタンスを作成するときに、デフォルトのパラメータテンプレートから自動的に作成されます。これらのデフォルトのパラメータグループには、優先されるパラメータ設定が含まれています。

これを変更することはできません。カスタムパラメータグループを作成する場合、パラメータ設定を変更できます。

コンソール

AWS アカウントのすべての DB パラメータグループを一覧表示するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、パラメータグループ を選択します。

   DB パラメータグループがリストに表示されます。

AWS CLI

AWS アカウントのすべての DB パラメータグループを一覧表示するには、AWS CLI の `describe-db-parameter-groups` コマンドを使用します。

Example

以下の例では、AWS アカウントに使用できるすべての DB パラメータグループを一覧表示しています。

```
aws rds describe-db-parameter-groups
```

このコマンドでは次のようなレスポンスが返されます。

```
DBPARAMETERGROUP  default.mysql8.0     mysql8.0  Default parameter group for MySQL8.0
DBPARAMETERGROUP  mydbparametergroup   mysql8.0  My new parameter group
```

次の例は、mydbparamgroup1 パラメータグループを表しています。

Linux、macOS、Unix の場合:

```
aws rds describe-db-parameter-groups \
    --db-parameter-group-name mydbparamgroup1
```

Windows の場合:

```
aws rds describe-db-parameter-groups ^
    --db-parameter-group-name mydbparamgroup1
```

このコマンドでは次のようなレスポンスが返されます。

```
DBPARAMETERGROUP  mydbparametergroup1  mysql8.0  My new parameter group
```

RDS API

AWS アカウントのすべての DB パラメータグループを一覧表示するには、RDS API の DescribeDBParameterGroups オペレーションを使用します。

DB パラメータグループのパラメータ値を表示する

DB パラメータグループのすべてのパラメータとそれらの値のリストを取得できます。

コンソール

DB パラメータグループのパラメータ値を表示するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、パラメータグループ を選択します。

   DB パラメータグループがリストに表示されます。
3. パラメータを一覧表示するパラメータグループの名前を選択します。

AWS CLI

DB パラメータグループのパラメータ値を表示するには、以下の必須パラメータを指定して、AWS CLI の describe-db-parameters コマンドを使用します。

- `--db-parameter-group-name`

Example

以下の例では、mydbparametergroup という名前の DB パラメータグループのパラメータとその値を一覧表示しています。

```
aws rds describe-db-parameters --db-parameter-group-name mydbparametergroup
```

このコマンドでは次のようなレスポンスが返されます。

```
DBPARAMETER  Parameter Name            Parameter Value  Source           Data Type
 Apply Type  Is Modifiable
DBPARAMETER  allow-suspicious-udfs                      engine-default   boolean
 static      false
DBPARAMETER  auto_increment_increment                   engine-default   integer
 dynamic     true
DBPARAMETER  auto_increment_offset                      engine-default   integer
 dynamic     true
DBPARAMETER  binlog_cache_size         32768            system           integer
 dynamic     true
DBPARAMETER  socket                    /tmp/mysql.sock  system           string
 static      false
```

RDS API

DB パラメータグループのパラメータ値を表示するには、以下の必須パラメータを指定して、RDS API の `DescribeDBParameters` コマンドを使用します。

- `DBParameterGroupName`

DB パラメータグループの削除

DB パラメータグループは、AWS Management Console、AWS CLI、または RDS API を使用して削除できます。パラメータグループは、DB インスタンスに関連付けられていない場合にのみ削除できます。

コンソール

DB パラメータグループを削除するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、パラメータグループ を選択します。

   DB パラメータグループがリストに表示されます。
3. 削除するパラメータグループの名前を選択します。

4. [Actions (アクション)] を選択してから [Delete (削除)] を選択します。
5. パラメータグループ名を確認して、[削除] を選択します。

AWS CLI

DB パラメータグループを削除するには、AWS CLI の `delete-db-parameter-group` コマンドを使用して、次の必須パラメータを指定します。

- `--db-parameter-group-name`

Example

次の例では、mydbparametergroup という名前の DB パラメータグループを削除します。

```
aws rds delete-db-parameter-group --db-parameter-group-name mydbparametergroup
```

RDS API

DB パラメータグループを削除するには、RDS API `DeleteDBParameterGroup` コマンドを使用して、次の必須パラメータを指定します。

- `DBParameterGroupName`

DB パラメータグループを比較する

AWS Management Console を使用して、2 つの DB パラメータグループ間の差異を表示できます。

指定されたパラメータグループは、両方とも DB パラメータグループであるか、両方とも DB クラスターパラメータグループである必要があります。DB エンジンとバージョンが同じであっても当てはまります。例えば、`aurora-mysql8.0` (Aurora MySQL バージョン 3) DB パラメータグループと `aurora-mysql8.0` DB クラスターパラメータグループを比較することはできません。

Aurora MySQL と RDS for MySQL DB のパラメータグループは、バージョンが異なっていても比較できますが、Aurora PostgreSQL と RDS for PostgreSQL DB のパラメータグループは比較できません。

2 つの DB パラメータグループを比較するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[パラメータグループ] を選択します。
3. リストで、比較する 2 つのパラメータグループを選択します。

> **Note**
>
> デフォルトのパラメータグループとカスタムパラメータグループを比較するには、まず、[デフォルト] タブでデフォルトパラメータグループを選択し、次に [カスタム] タブでカスタムパラメータグループを選択します。

4. [アクション] で、[比較] を選択します。

DB パラメータの指定

DB パラメータの種類には、次のものがあります。

- 整数
- ブール値
- 文字列
- Long
- ダブル
- タイムスタンプ
- 他の定義されたデータ型オブジェクト
- 整数、ブール値、文字列、長整数、倍精度、タイムスタンプ、オブジェクト型の値の配列

表現、数式、関数を使用して、整数およびブール型パラメータを指定することもできます。

目次

- DB パラメータログ式
- DB パラメータ値の例

DB パラメータ式

DB パラメータ式は整数値に解決される式あるいはブール値です。式は中かっこ {} で囲みます。式は、DB パラメータ値、または DB パラメータ関数の引数として使用できます。

構文

```
{FormulaVariable}
{FormulaVariable*Integer}
{FormulaVariable*Integer/Integer}
{FormulaVariable/Integer}
```

DB パラメータ式の変数

各式の変数は整数あるいはブール値を返します。変数名では大文字と小文字が区別されます。

AllocatedStorage

データボリュームのサイズ (バイト単位) を表す整数を返します。

DBInstanceClassMemory

データベースプロセスに対し、使用可能なメモリのバイト数を整数で返します。この数値は、DB インスタンスクラスの合計メモリ量から始めて内部的に計算されます。この数から、インスタンスを管理する RDS プロセスとオペレーティングシステム用に予約されているメモリ量を減算します。したがって、この数値は、「Aurora DB インスタンスクラス」のインスタンスクラステーブルに示されているメモリ値よりも、常にやや低くなります。正確な値は、複数の要因の組み合わせによって異なります。これらには、インスタンスクラスと DB エンジンが含まれ、この値が適用されるのが RDS インスタンスなのか、Aurora クラスターに含まれているインスタンスなのかによっても異なります。

EndPointPort

DB インスタンスに接続するときに使用されるポートを表す整数を返します。

TrueIfReplica

DB インスタンスがリードレプリカである場合は `1`、リードレプリカでない場合は `0` が返されます。これは Aurora MySQL の`read_only` パラメータのデフォルト値です。

DB パラメータ式の演算子

DB パラメータ式では、2 つ演算子 (除算と乗算) がサポートされています。

除算演算子: /

被除数を除数で割り、整数の商を返します。商の小数部分は四捨五入されず切り捨てられます。

構文

```
dividend / divisor
```

被除数と除数の引数は整数式である必要があります。

乗算演算子: *

式を乗算し、式の積を返します。式の小数部分は四捨五入されず切り捨てられます。

構文

```
expression * expression
```

両方の式は整数である必要があります。

DB パラメータ関数

DB パラメータ関数の引数は、整数または数式で指定します。各関数には 1 つ以上の引数が必要です。複数の引数をカンマ区切りのリストで指定します。リストには、argument1,,argument3 など、空のメンバーを使用することはできません。関数名では大文字と小文字は区別されません。

IF

引数を返します。

構文

```
IF(argument1, argument2, argument3)
```

最初の引数が true と評価する場合に、2 番目の引数を返します。それ以外の場合には、3 番目の引数を返します。

GREATEST

整数またはパラメータ式のリストから最大値を返します。

構文

```
GREATEST(argument1, argument2,...argumentn)
```

整数を返します。

LEAST

整数またはパラメータ式のリストから最小値を返します。

構文

```
LEAST(argument1, argument2,...argumentn)
```

整数を返します。

SUM

指定した整数またはパラメータ式の値を加算します。

構文

```
SUM(argument1, argument2,...argumentn)
```

整数を返します。

DB パラメータログ式

整数 DB パラメータ値を log 式に設定できます。式は中かっこ {} で囲みます。次に例を示します。

```
{log(DBInstanceClassMemory/8187281418)*1000}
```

log 関数は log ベース 2 を表します。この例では、DBInstanceClassMemory 数式変数も使用しています。「DB パラメータ式の変数」を参照してください。

DB パラメータ値の例

これらの例は、DB パラメータの値に対して数式、関数、および式を使用していることを示しています。

> **Warning**
>
> DB パラメータグループのパラメータを不適切に設定すると、意図しない悪影響が生じる可能性があります。これには、パフォーマンスの低下やシステムの不安定化が含まれます。データベースパラメータの変更時には注意が必要です。DB パラメータグループの変更前にはデータをバックアップしてください。パラメータグループの変更は、テスト DB インスタンス (ポイントインタイム復元を使用して作成) で試してから、本番稼働用 DB インスタンスに適用してください。

Example DB パラメータ関数 LEAST の使用

Aurora MySQL LEAST パラメータ値で table_definition_cache 関数を指定できます。これを使用して、定義キャッシュに保管できるテーブル定義の数を DBInstanceClassMemory/393040 または 20,000 のいずれか少ない方に設定します。

```
LEAST({DBInstanceClassMemory/393040}, 20000)
```

Amazon Aurora DB クラスターへのデータの移行

データベースエンジンの互換性に応じて、既存のデータベースから Amazon Aurora DB クラスターにデータを移行するオプションは異なります。また、移行オプションは、移行元のデータベースおよび移行するデータのサイズによっても異なります。

Amazon Aurora MySQL DB クラスターへのデータの移行

以下のいずれかのソースから Amazon Aurora MySQL DB クラスターにデータを移行できます。

- RDS for MySQL DB インスタンス
- Amazon RDS 外部の MySQL データベース
- MySQL と互換性がないデータベース

詳細については、「Amazon Aurora MySQL DB クラスターへのデータの移行」を参照してください。

Amazon Aurora PostgreSQL DB クラスターへのデータの移行

以下のいずれかのソースから Amazon Aurora PostgreSQL DB クラスターにデータを移行できます。

- Amazon RDS PostgreSQL DB インスタンス
- PostgreSQL と互換性がないデータベース

詳細については、「PostgreSQL と互換性がある Amazon Aurora にデータを移行する」を参照してください。

Aurora DB クラスター設定を使用して Amazon ElastiCache キャッシュを作成する

ElastiCache は、フルマネージドのインメモリキャッシュサービスであり、マイクロ秒単位の読み取り/書き込みレイテンシーを提供し、柔軟なリアルタイムユースケースをサポートします。ElastiCache は、アプリケーションとデータベースのパフォーマンスを向上させるのに役立ちます。ElastiCache は、ゲームのリーダーボード、ストリーミング、データ分析など、データの耐久性を必要としないユースケースのプライマリデータストアとして使用できます。ElastiCache は、分散コンピューティング環境のデプロイと管理に関連する複雑さを排除するのに役立ちます。詳細については、Memcached の場合「ElastiCache の一般的なユースケースおよび ElastiCache がどのように役立つか」を、Redis の場合「ElastiCache の一般的なユースケースおよび ElastiCache がどのように役立つか」を参照してください。ElastiCache キャッシュを作成する際に、Amazon RDS コンソールを使用できます。

Amazon ElastiCache は 2 つの形式で運用できます。サーバーレスキャッシュで始めるか、独自のキャッシュクラスターを設計するかを選択できます。独自のキャッシュクラスターを設計する場合、ElastiCache は Redis エンジンと Memcached エンジンの両方で動作します。使用するエンジンが不明な場合は、「Memcached と Redis の比較」を参照してください。Amazon ElastiCache の詳細については、「Amazon ElastiCache ユーザーガイド」を参照してください。

トピック

- Aurora DB クラスター設定による ElastiCache キャッシュ作成の概要
- 既存の Aurora DB クラスターの設定で ElastiCache キャッシュを作成する

Aurora DB クラスター設定による ElastiCache キャッシュ作成の概要

新規作成または既存の Aurora DB クラスターと同じ設定を使用して、Amazon RDS から ElastiCache キャッシュを作成できます。

ElastiCache キャッシュを DB クラスターに関連付けるためのいくつかのユースケース:

- RDS で ElastiCache を使用すると、RDS だけで実行するよりもコストを節約し、パフォーマンスを向上させることができます。
- ElastiCache キャッシュは、データの耐久性を必要としないプライマリデータストアとしてアプリケーションで使用できます。Redis または Memcached を使用するアプリケーションでは ElastiCache を利用できます。ほとんど変更はありません。

RDS から ElastiCache キャッシュを作成すると、ElastiCache キャッシュは、関連する Aurora DB クラスターから以下の設定を継承します。

- ElastiCache 接続設定
- ElastiCache セキュリティ設定

要件に応じて、キャッシュ設定を指定できます。

アプリケーションで ElastiCache を設定する

アプリケーションは ElastiCache キャッシュを利用するように設定する必要があります。また、要件に応じてキャッシュ戦略を使用するようにアプリケーションを設定することで、キャッシュのパフォーマンスを最適化して改善できます。

- ElastiCache キャッシュにアクセスして開始するには、「Amazon ElastiCache for Redis を使い始める」と「Amazon ElastiCache for Memcached を使い始める」を参照してください。
- キャッシュ戦略の詳細については、Memcached の場合は「キャッシュ戦略とベストプラクティス」、Redis の場合は「キャッシュ戦略とベストプラクティス」を参照してください。
- ElastiCache for Redis クラスターの高可用性の詳細については、「レプリケーショングループを使用した高可用性」を参照してください。
- バックアップストレージ、リージョン内またはリージョン間のデータ転送、または使用に関連するコストが発生する可能性があります。AWS Outposts価格設定の詳細については、「Amazon ElastiCache 料金表」を参照してください。

既存の Aurora DB クラスターの設定で ElastiCache キャッシュを作成する

DB クラスターから継承された設定を使用して、Aurora DB クラスター の ElastiCache キャッシュを作成できます。

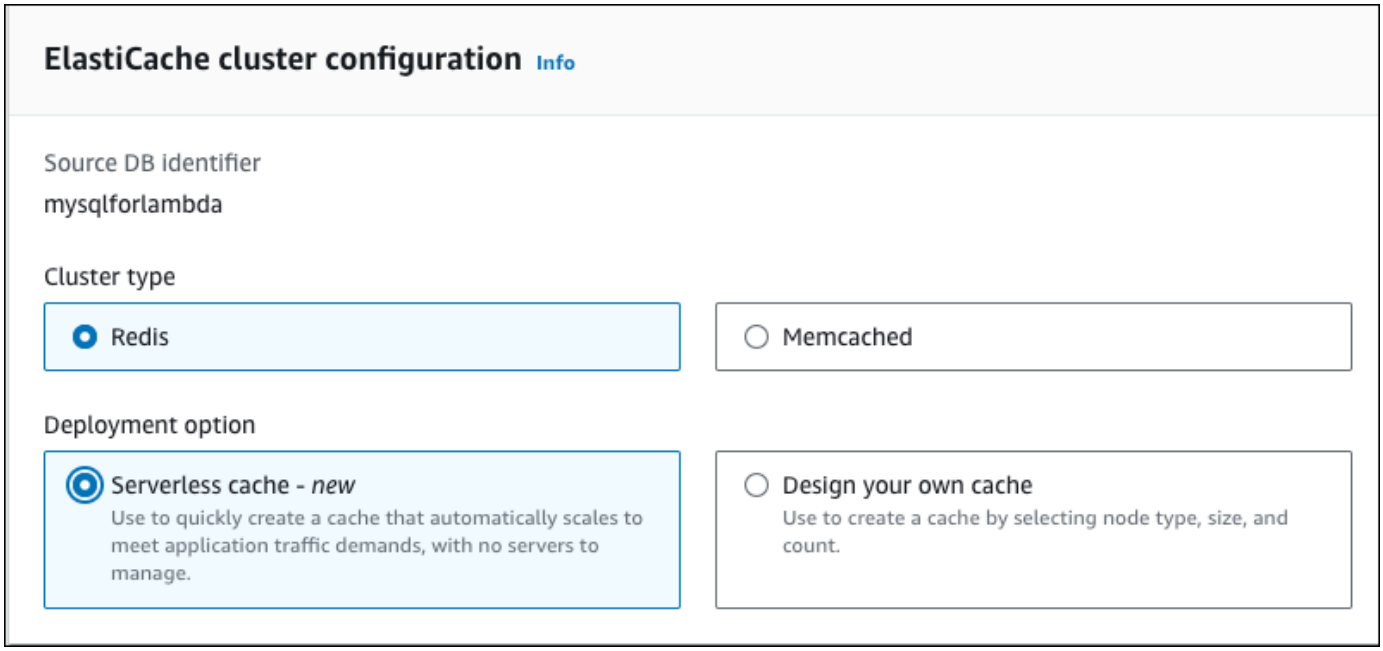
DB クラスターの設定で ElastiCache キャッシュを作成する

1. DB クラスターを作成するには、「Amazon Aurora DB クラスターの作成」の手順に従います。
2. Aurora DB クラスターを作成すると、コンソールに [推奨されるアドオン] ウィンドウが表示されます。[DB 設定を使用して RDS から ElastiCache クラスターを作成する] を選択します。

既存のデータベースでは、[データベース] ページで、必要な DB クラスターを選択します。[アクション] ドロップダウンメニューで [ElastiCache クラスターの作成] を選択して、既存の Aurora DB クラスターと同じ設定の ElastiCache キャッシュを RDS に作成します。

[ElastiCache 設定セクション] で、[ソース DB 識別子] に ElastiCache キャッシュが設定を継承する DB クラスターが表示されます。

3. Redis または Memcached クラスターを作成するかどうかを選択します。詳細については、「Memcached と Redis の比較」を参照してください。



4. その後、[サーバーレスキャッシュ] を作成するか、[独自のキャッシュを設計] するかを選択します。詳細については、「デプロイオプションの選択」を参照してください。

   [サーバーレスキャッシュ] を選択した場合：

   a. [キャッシュ設定] で、[名前] と [説明] に値を入力します。

   b. [デフォルト設定の表示] ではデフォルト設定のままにして、キャッシュと DB クラスター間の接続を確立します。

   c. [デフォルト設定をカスタマイズ] を選択して、デフォルトの設定を編集することもできます。[ElastiCache の接続設定]、[ElastiCache のセキュリティ設定]、[使用量の上限] を選択します。

5. [独自のキャッシュ を設計] を選択した場合：

a. [Redis クラスター] を選択した場合は、クラスターモードを [有効] にするか [無効] にするかを選択します。詳細については、「レプリケーション: Redis (クラスターモードが無効) 対 Redis (クラスターモードが有効)」を参照してください。

b. [名前]、[説明]、[エンジンバージョン] の値を入力します。

   [エンジンバージョン] については、推奨されるデフォルト値は最新のエンジンバージョンです。要件に最も合う ElastiCache キャッシュの [エンジンバージョン] を選択することもできます。

c. [ノードタイプ] オプションでノードタイプを選択します。詳細については、「ノードの管理」を参照してください。

   [クラスターモード] を [有効] に設定して Redis クラスターを作成する場合、[シャード数] オプションにシャード (パーティション/ノードグループ) の数を入力します。

   [レプリカ数] に各シャードのレプリカ数を入力します。

   > **Note**
   >
   > 選択したノードタイプ、シャード数、レプリカ数はすべて、キャッシュのパフォーマンスとリソースコストに影響します。これらの設定がデータベースのニーズに合っていることを確認してください。料金情報については、「Amazon ElastiCache 料金表」を参照してください。

d. [ElastiCache の接続設定] と [ElastiCache のセキュリティ設定] を選択します。デフォルト設定のままにすることも、要件に応じて設定をカスタマイズすることもできます。

6. ElastiCache キャッシュのデフォルト設定と継承された設定を確認します。一部の設定は、作成後に変更できません。

   > **Note**
   >
   > RDS は、60 分の最小ウィンドウ要件を満たすように ElastiCache キャッシュのバックアップウィンドウを調整する場合があります。ソースデータベースのバックアップウィンドウは変わりません。

7. 準備が完了したら、[ElastiCache キャッシュの作成] を選択します。

コンソールに、ElastiCache キャッシュの作成に関する確認バナーが表示されます。バナーにあるリンクから ElastiCache コンソールにアクセスすると、キャッシュの詳細が表示されます。ElastiCache コンソールには、新しく作成された ElastiCache キャッシュが表示されます。

Amazon Aurora DB クラスターの管理

このセクションでは、Aurora DB クラスターを管理し維持する方法を示します。Aurora は、レプリケーショントポロジに接続されているデータベースサーバーのクラスターに関連します。そのため、Aurora を管理するには、複数のサーバーへの変更をデプロイし、すべての Aurora レプリカがマスターサーバーで維持されていることを確認する必要があります。Aurora では、データの増加に伴い、基本となるストレージを透過的にスケーリングしているため、Aurora の管理に必要なディスクストレージの管理は比較的わずかです。同様に、Aurora では、継続的バックアップが自動的に行われるため、Aurora クラスターでは、バックアップの実行に伴う過度な計画やダウンタイムは必要ありません。

トピック

Amazon Aurora DB クラスターの停止と開始

Amazon Aurora クラスターの停止と開始は、開発とテスト環境のコスト管理に役立ちます。クラスターを使用するたびに、すべての DB インスタンスを設定および解放するのではなく、クラスターですべての DB インスタンスを一時的に停止することができます。

トピック

Aurora DB クラスターの停止と開始の概要

Aurora クラスターが必要ではない期間は、そのクラスターですべてのインスタンスを一度に停止することができます。クラスターを使用する必要がある時はいつでもクラスターを開始できます。開始と停止は、継続的な可用性を必要としない開発、テスト、または類似のアクティビティに使用されるクラスターのセットアップと解放のプロセスを簡素化します。クラスターのインスタンス数に関係なく、1 つのアクションだけに関連するすべての AWS Management Console の手順を実行できます。

DB クラスターの停止中は、指定された保持期間内のクラスターストレージ、手動のスナップショット、および自動化されたバックアップストレージに対してのみ課金されます。DB インスタンス時間に対しては請求されません。

> **Important**
>
> DB クラスターは最大 7 日間停止できます。7 日後に DB クラスターを手動で起動しなかった場合、DB クラスターは自動的に起動されるため、必要なメンテナンス更新が遅れることはありません。

ライトにロードされた Aurora クラスターの料金を最小限に抑えるために、その Aurora レプリカのすべてを削除するのはなくクラスターを停止することができます。1 つまたは 2 つ以上のインスタンスを持つクラスターの場合、頻繁に DB インスタンスを削除して再作成するためには、AWS CLI または Amazon RDS API を使用することが唯一実用的です。またこのような一連のオペレーション

は、正しい順序で実行することが難しくなります。例えば、フェイルオーバーメカニズムのアクティブ化を回避するために、プライマリインスタンスを削除する前にすべての Aurora レプリカを削除します。

DB クラスターの実行を維持する必要がありますが、必要以上に容量がある場合は、開始と停止を使用しないでください。クラスターのコストが高すぎる、または非常に多忙でない場合は、1 つまたは複数の DB インスタンスを削除するか、すべての DB インスタンスを小さなインスタンスクラスに変更します。個々の Aurora DB インスタンスを停止できません。

Aurora DB クラスターの停止と起動に関する制約事項

一部の Aurora クラスターは、停止および起動することができません。

- Aurora グローバルデータベースの一部であるクラスターを停止および起動することはできません。
- クロスリージョンリードレプリカを持つクラスターを停止および開始することはできません。
- ブルー/グリーンデプロイの一部であるクラスターを停止および開始することはできません。
- Aurora 並列クエリ機能を使用するクラスターの場合、最小の Aurora MySQL バージョンは 2.09.0 です。
- Aurora Serverless v1 クラスターを停止および開始することはできません。Aurora Serverless v2 では、クラスターを停止および開始できます。

既存のクラスターを停止および開始できない場合、[データベース] ページや詳細ページで[アクション] メニューの [停止] アクションは使用できません。

Aurora DB クラスターの停止

Aurora DB クラスターを使用するまたは管理を実行するには、常に実行している Aurora DB クラスターから始め、クラスターを停止し、クラスターを再び開始します。クラスターの停止中に、指定された保持期間内のクラスターストレージ、手動のスナップショット、および自動化されたバックアップストレージに対しては課金されますが、DB インスタンス時間に対しては課金されません。

停止オペレーションは、まず Aurora レプリカインスタンスを停止し、次にプライマリインスタンスを停止して、フェイルオーバーメカニズムのアクティブ化を回避します。

別の DB クラスターからのデータのレプリケーションターゲットとして機能する DB クラスターを停止することはできません。また、レプリケーションマスターとして機能し、別のクラスターにデータを送信することもできません。

特定の種類のクラスターを停止することはできません。現在、Aurora グローバルデータベースの一部であるクラスターを停止することはできません。

コンソール

Aurora クラスターを停止するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[クラスター] を選択し、クラスターを選択します。このページで停止オペレーションを実行するか、停止する DB クラスターの詳細ページに移動します。
3. [Actions] (アクション) で、[Stop temporarily] (一時的に停止) を選択します。

   DB クラスターの停止と開始ができない場合、[Databases] (データベース) ページまたは詳細ページの [Actions] (アクション) メニューから [Stop temporarily] (一時的に停止) アクションを使用することはできません。起動および停止できないクラスターの種類については、「Aurora DB クラスターの停止と起動に関する制約事項」を参照してください。
4. [Stop DB cluster temporarily] (DB クラスターを一時的に停止) ウィンドウで、DB クラスターが 7 日後に自動的に再起動することを確認するメッセージを選択します。
5. [Stop temporarily] (一時的に停止) を選択して DB クラスターを停止するか、[Cancel] (キャンセル) を選択して操作をキャンセルします。

AWS CLI

AWS CLI を使用して DB インスタンスを停止するには、以下のパラメータを指定して stop-db-cluster コマンドを呼び出します。

- `--db-cluster-identifier` – Aurora クラスターの名前。

Example

```
aws rds stop-db-cluster --db-cluster-identifier mydbcluster
```

RDS API

Amazon RDS API を使用して DB インスタンスを停止するには、以下のパラメータを指定して StopDBCluster オペレーションを呼び出します。

- `DBClusterIdentifier` – Aurora クラスターの名前。

Aurora DB クラスターの停止中に実行できるオペレーション

Aurora クラスター停止中に、指定された自動バックアップ保持期間内であれば、ポイントインタイムの復元を実行できます。ポイントインタイムの復元の実行の詳細については、「データの復元」を参照してください。

クラスターを停止している間は、Aurora DB クラスターまたはその DB インスタンスの設定を変更することはできません。クラスターに DB インスタンスを追加または削除することも、関連する DB インスタンスがまだある場合はクラスターを削除することもできません。そのような管理アクションを実行する前に、クラスターを開始する必要があります。

DB クラスターを停止すると、DB クラスターのパラメータグループまたは DB クラスターインスタンスの DB パラメータグループ以外の保留中のアクションが削除されます。

Aurora は、スケジュールされたメンテナンスを、再び開始した後に停止したクラスターに適用します。Aurora は、7 日後に停止したクラスターを自動的に開始し、メンテナンスのステータスが遅くなり過ぎないようにすることに注意してください。

Aurora はまた、クラスターが停止している間に基礎となるデータを変更できないため、自動バックアップを実行できません。Aurora は、クラスターの停止中にそのバックアップ保持期間を延長しません。

Aurora DB クラスターの開始

既に停止状態になっている Aurora クラスターで始まる Aurora DB クラスターを常に開始します。クラスターを開始すると、すべての DB インスタンスが再び利用可能になります。クラスターは、エンドポイント、パラメータグループ、および VPC セキュリティグループなどの構成設定を維持します。

通常、DB クラスターの作成には数分かかります。

コンソール

Aurora クラスターを開始するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。

2. ナビゲーションペインで、[クラスター] を選択し、クラスターを選択します。このページで開始オペレーションを実行するか、開始する DB クラスターの詳細ページに移動します。
3. [アクション] で [開始] を選択します。

AWS CLI

AWS CLI を使用して DB クラスターを開始するには、以下のパラメータを指定して start-db-cluster コマンドを呼び出します。

- `--db-cluster-identifier` – Aurora クラスターの名前。この名前は、クラスターを作成した時に選択した特定のクラスター識別子、または最後に `-cluster` を付加して選択した DB インスタンス識別子のどちらかです。

Example

```
aws rds start-db-cluster --db-cluster-identifier mydbcluster
```

RDS API

Amazon RDS API を使用して Aurora DB クラスターを開始するには、以下のパラメータを指定して StartDBCluster オペレーションを呼び出します。

- `DBCluster` – Aurora クラスターの名前。この名前は、クラスターを作成した時に選択した特定のクラスター識別子、または最後に `-cluster` を付加して選択した DB インスタンス識別子のどちらかです。

AWS コンピューティングリソースと オーロラ DB クラスターを自動的に接続する

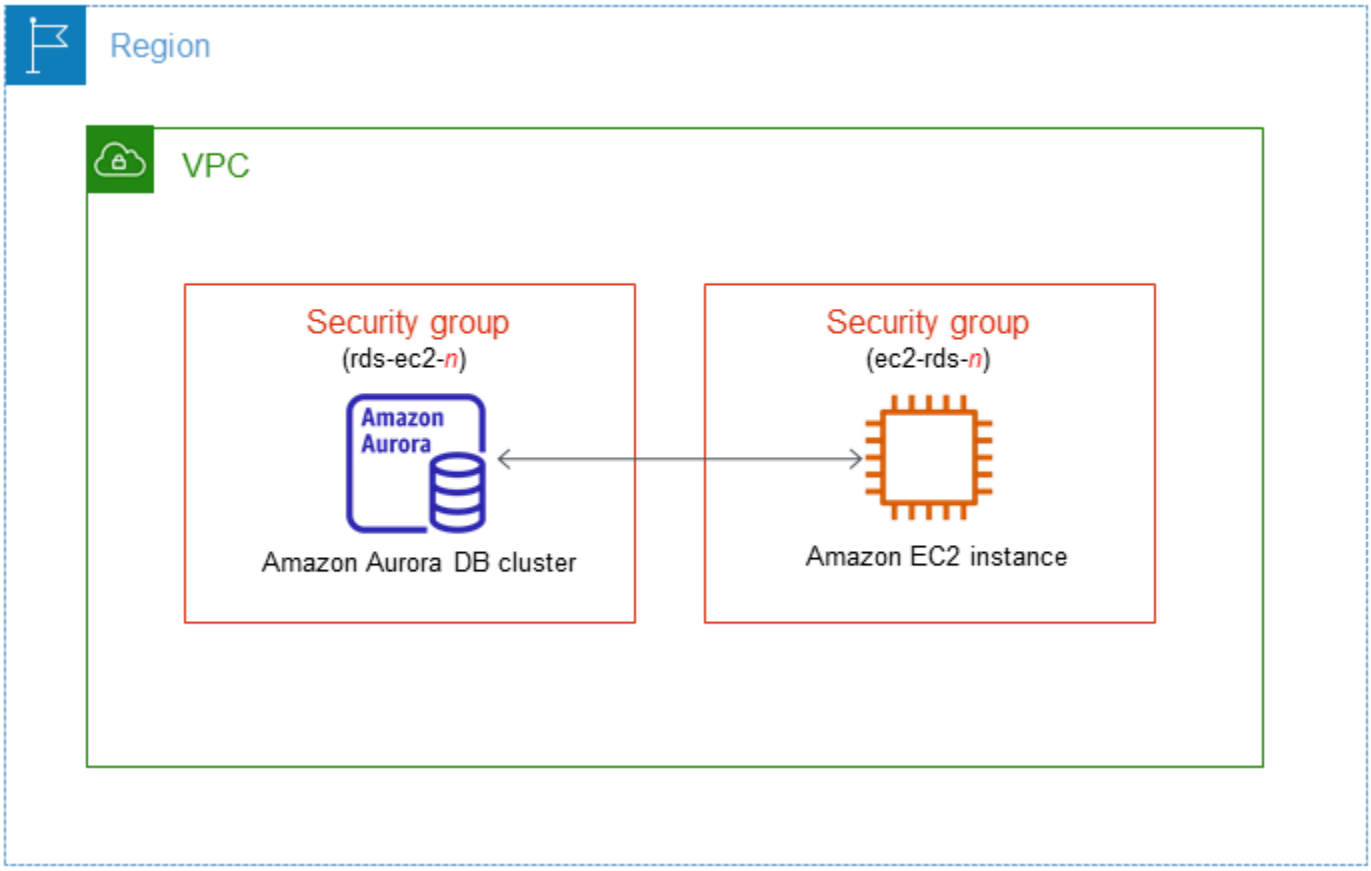
Aurora DB クラスター と、Amazon Elastic Compute Cloud (Amazon EC2) インスタンスや AWS Lambda 関数などの AWS コンピューティングリソースを自動的に接続できます。

トピック

- EC2 インスタンスと Aurora DB クラスターを自動的に接続する
- Lambda 関数と Aurora DB クラスターを自動的に接続する

EC2 インスタンスと Aurora DB クラスターを自動的に接続する

RDS コンソールを使用して、Amazon Elastic Compute Cloud (Amazon EC2) インスタンスと Aurora DB クラスターとの接続を簡単に設定できます。多くの場合、DB クラスターはプライベートサブネットにあり、EC2 インスタンスは VPC 内のパブリックサブネットにあります。EC2 インスタンスの SQL クライアントを使用して、DB クラスターに接続できます。EC2 インスタンスは、プライベート DB クラスターにアクセスするウェブサーバーやアプリケーションを実行することもできます。



Aurora DB クラスターと同じ VPC にない EC2 インスタンスに接続する場合は、「VPC の DB クラスターにアクセスするシナリオ」のシナリオを参照してください。

トピック

- EC2 インスタンスとの自動接続の概要
- EC2 インスタンスと Aurora DB クラスターを自動的に接続する
- 接続中のコンピューティングリソースを表示する
- 特定の DB エンジンを実行している DB インスタンスに接続する

EC2 インスタンスとの自動接続の概要

EC2 インスタンスと Aurora DB クラスター間の接続を設定すると、Amazon RDS は EC2 インスタンスと DB クラスター の VPC セキュリティグループを自動的に設定します。

EC2 インスタンスと Aurora DB クラスター を接続するための要件は次のとおりです。

- EC2 インスタンスは DB クラスター と同じ VPC に存在する必要があります。

同じ VPC に EC2 インスタンスが存在しない場合、コンソールには EC2 インスタンス作成用のリンクが表示されます。

- 現在、DB クラスターを Aurora Serverless DB クラスターまたは Aurora グローバルデータベースの一部にすることはできません。
- 接続を設定するユーザーには、以下の Amazon EC2 オペレーションを実行するアクセス許可が必要です。
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeSecurityGroups`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

DB インスタンスと EC2 インスタンスが異なるアベイラビリティーゾーンにある場合、アベイラビリティーゾーン間のコストが発生する可能性があります。

EC2 インスタンスへの接続を設定すると、次の表で示されているように、Amazon RDS は、DB クラスターと EC2 インスタンスに関連付けられているセキュリティグループの現在の設定に基づいて動作します。

現在の RDS セキュリティグループの設定	現在の EC2 セキュリティグループの設定	RDS アクション
DB クラスターに関連付けられたセキュリティグループには、パターン `rds-ec2-`*n* (*n* は数字) に一致する名前のセキュリティグループが 1 つまたは複数あります。パターンに一致するセキュリティグループは変更されていません。このセキュリティグルー	パターン `ec2-rds-`*n* (*n* は数字) に一致する名前の EC2 インスタンスに関連付けられたセキュリティグループが 1 つまたは複数存在します。パターンに一致するセキュリティグループは変更されていません。このセキュリティグループには、 DB クラスター	RDS は何のアクションも実行しません。 EC2 インスタンスと DB クラスターとの接続は、既に自動で設定されています。EC2 インスタンスと RDS データベースの間には既に接続が存在す

現在の RDS セキュリティグループの設定	現在の EC2 セキュリティグループの設定	RDS アクション
プには、EC2 インスタンスの VPC セキュリティグループをソースとするインバウンドルールが 1 つのみ存在します。	の VPC セキュリティグループをソースとするアウトバウンドルールが 1 つのみ存在します。	るため、セキュリティグループは変更されません。

現在の RDS セキュリティグループの設定	現在の EC2 セキュリティグループの設定	RDS アクション
次の条件のいずれかが適用されます。 • DB クラスターには、パターン `rds-ec2-`*n* と名前が一致する関連付けられたセキュリティグループはありません。 • DB クラスターに関連付けられたセキュリティグループには、パターン `rds-ec2-`*n* に一致する名前のセキュリティグループが 1 つまたは複数あります。ただし、Amazon RDS は、これらのセキュリティグループのいずれも、EC2 インスタンスとの接続には使用できません。Amazon RDS は、EC2 インスタンスの VPC セキュリティグループをソースとするインバウンドルールが 1 つも存在しないセキュリティグループを使用できません。また、Amazon RDS は、変更されたセキュリティグループを使用できません。変更の例としては、ルールの追加や、既存ルールのポート変更などがあります。	次の条件のいずれかが適用されます。 • EC2 インスタンスに関連付けされた、パターン `ec2-rds-`*n* に一致する名前のセキュリティグループは存在しません。 • EC2 インスタンスに関連付けられた、パターン `ec2-rds-`*n* に一致する名前のセキュリティグループが 1 つまたは複数存在します。ただし、Amazon RDS は、これらのセキュリティグループのいずれも、DB クラスターとの接続には使用できません。Amazon RDS は、DB クラスター の VPC セキュリティグループをソースとするアウトバウンドルールが 1 つもないセキュリティグループを使用できません。また、Amazon RDS は、変更されたセキュリティグループを使用できません。	RDS action: create new security groups

現在の RDS セキュリティグループの設定	現在の EC2 セキュリティグループの設定	RDS アクション
DB クラスターに関連付けられたセキュリティグループには、パターン `rds-ec2-`*n* に一致する名前のセキュリティグループが 1 つまたは複数あります。パターンに一致するセキュリティグループは変更されていません。このセキュリティグループには、EC2 インスタンスの VPC セキュリティグループをソースとするインバウンドルールが 1 つのみ存在します。	EC2 インスタンスに関連付けられた、パターン `ec2-rds-`*n* に一致する名前のセキュリティグループが 1 つまたは複数存在します。ただし、Amazon RDS は、これらのセキュリティグループのいずれも、DB クラスターとの接続には使用できません。Amazon RDS は、DB クラスターの VPC セキュリティグループをソースとするアウトバウンドルールが 1 つもないセキュリティグループを使用できません。また、Amazon RDS は、変更されたセキュリティグループを使用できません。	RDS action: create new security groups
DB クラスターに関連付けられたセキュリティグループには、パターン `rds-ec2-`*n* に一致する名前のセキュリティグループが 1 つまたは複数あります。パターンに一致するセキュリティグループは変更されていません。このセキュリティグループには、EC2 インスタンスの VPC セキュリティグループをソースとするインバウンドルールが 1 つのみ存在します。	接続に有効な EC2 セキュリティグループは存在しますが、EC2 インスタンスに関連付けられていません。このセキュリティグループには、パターン `ec2-rds-`*n* に一致する名前が付いています。これは変更されていません。また、これには DB クラスターの VPC セキュリティグループをソースとするアウトバウンドルールが 1 つのみ存在します。	RDS action: associate EC2 security group

現在の RDS セキュリティグループの設定	現在の EC2 セキュリティグループの設定	RDS アクション
次の条件のいずれかが適用されます。 • DB クラスターには、パターン `rds-ec2-`*n* と名前が一致する関連付けられたセキュリティグループはありません。 • DB クラスターに関連付けられたセキュリティグループには、パターン `rds-ec2-`*n* に一致する名前のセキュリティグループが 1 つまたは複数あります。ただし、Amazon RDS は、これらのセキュリティグループのいずれも、EC2 インスタンスとの接続には使用できません。Amazon RDS は、EC2 インスタンスの VPC セキュリティグループをソースとするインバウンドルールが 1 つも存在しないセキュリティグループを使用できません。また、Amazon RDS は、変更されたセキュリティグループを使用できません。	EC2 インスタンスに関連付けられた、パターン `ec2-rds-`*n* に一致する名前のセキュリティグループが 1 つまたは複数存在します。パターンに一致するセキュリティグループは変更されていません。このセキュリティグループには、DB クラスター の VPC セキュリティグループをソースとするアウトバウンドルールが 1 つのみ存在します。	RDS action: create new security groups

RDS アクション: 新しいセキュリティグループを作成する

Amazon RDS は以下のアクションを実行します。

- パターン `rds-ec2-n` に一致する新しいセキュリティグループを作成します。このセキュリティグループには、EC2 インスタンスの VPC セキュリティグループをソースとするインバウンドルールが存在します。このセキュリティグループでは、 DB クラスターに関連付けられており、EC2 インスタンスが DB クラスターへのアクセスを許可します。
- パターン `ec2-rds-n` に一致する新しいセキュリティグループを作成します。このセキュリティグループには、DB クラスターの VPC セキュリティグループをターゲットとするアウトバウンドルールが存在します。このセキュリティグループには EC2 インスタンスに関連付けられ、EC2 インスタンスが DB クラスターにトラフィックの送信を許可します。

RDS アクション: EC2 セキュリティグループを関連付ける

Amazon RDS は、有効な既存の EC2 セキュリティグループを EC2 インスタンスに関連付けます。このセキュリティグループにより、EC2 インスタンスは DB クラスターにトラフィックの送信を許可します。

EC2 インスタンスと Aurora DB クラスターを自動的に接続する

EC2 インスタンスと Aurora DB クラスターとの接続を設定する前に、「EC2 インスタンスとの自動接続の概要」で説明されている要件を満たしていることを確認してください。

接続の設定後にこれらのセキュリティグループを変更すると、EC2 インスタンスと Aurora DB クラスターとの接続に影響する可能性があります。

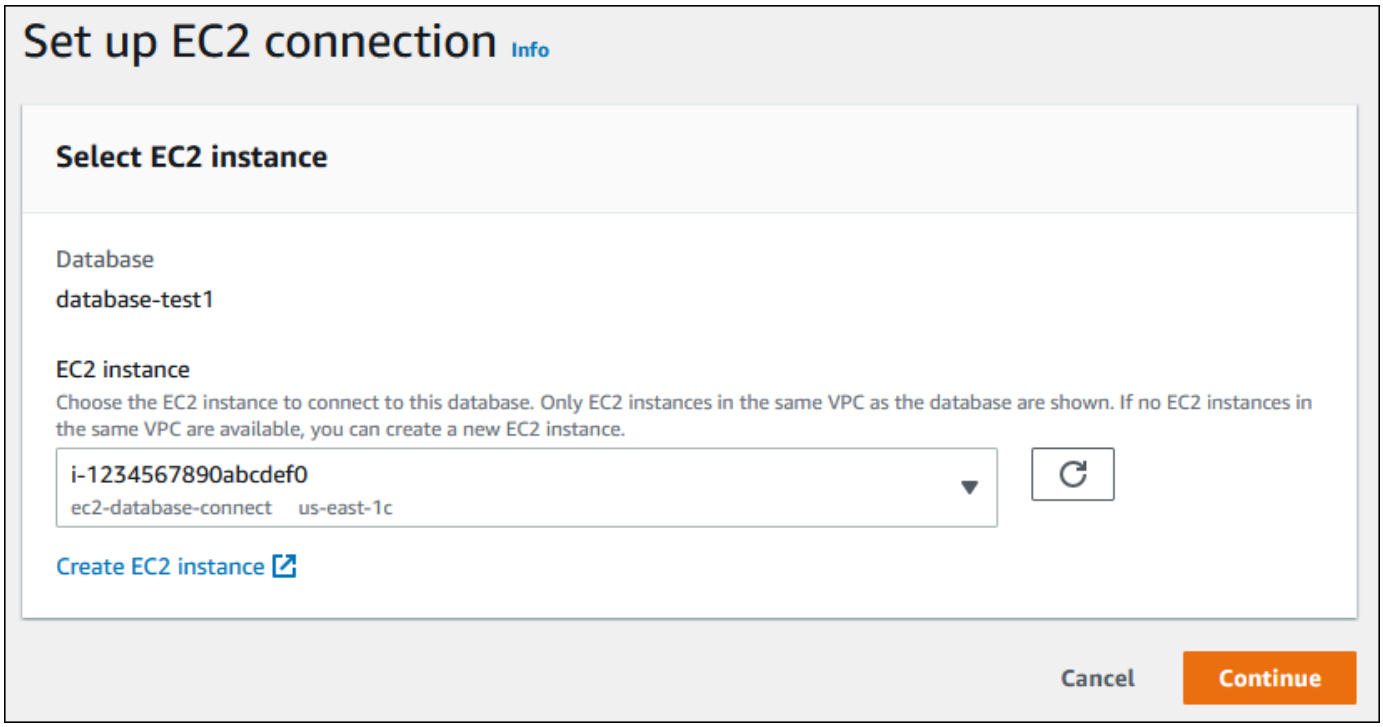
> **Note**
>
> AWS Management Console を使用することでのみ、EC2 インスタンスと Aurora DB クラスターとの接続を自動で設定できます。AWS CLI または RDS API を使用して自動で接続を設定することはできません。

EC2 インスタンスと Aurora DB クラスター を自動的に接続するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Databases] (データベース) を選択し、 DB クラスターのリンクを選択します。
3. [アクション] から [EC2 接続の設定] を選択します。

[Set up EC2 connection] (EC2 接続の設定) ページが表示されます。

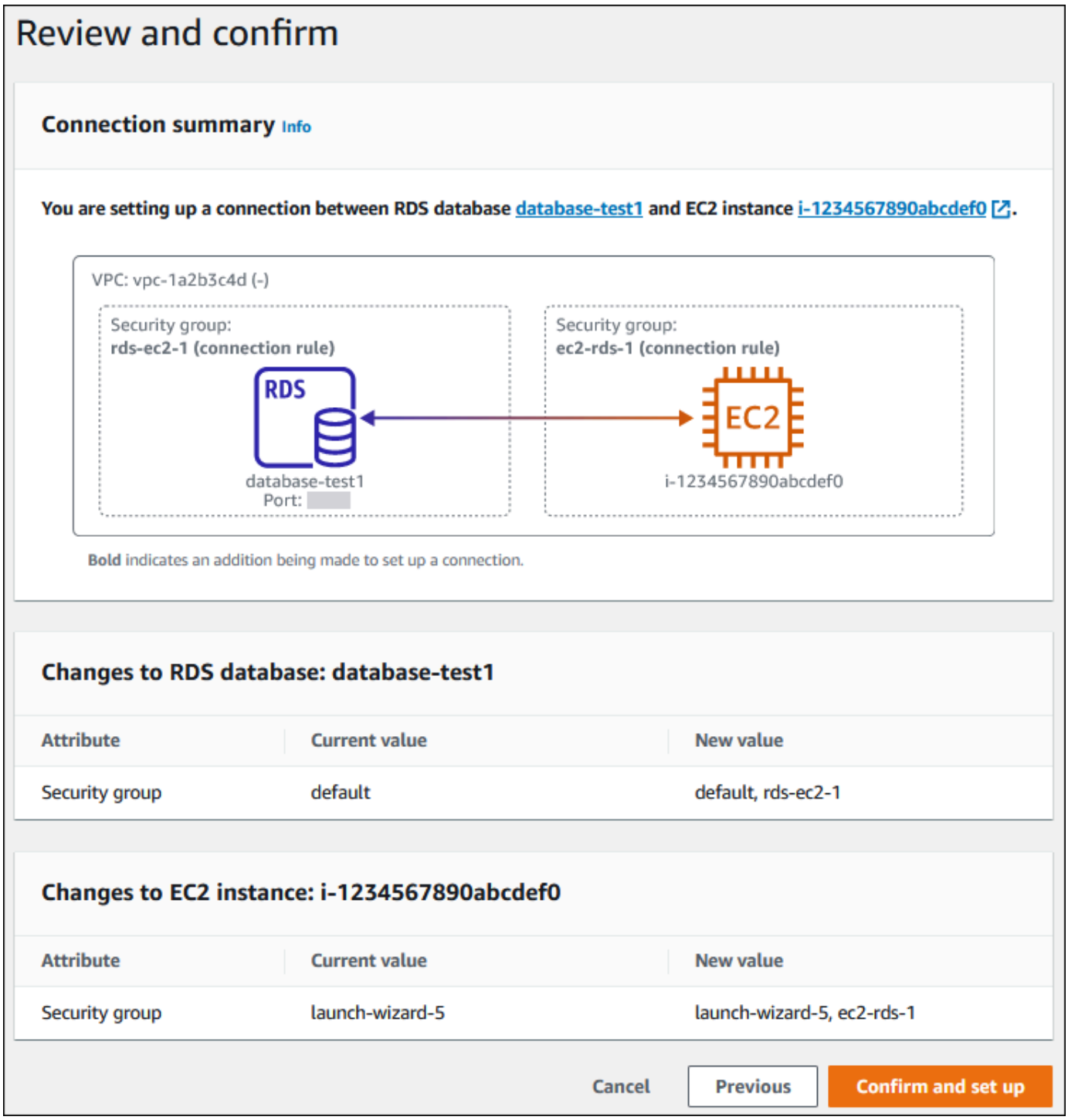
4. [Set up EC2 connection] (EC2 接続の設定) ページで、[EC2 instance] (EC2 インスタンス) を選択します。



同じ VPC に EC2 インスタンスが存在しない場合は、[Create EC2 instance] (EC2 インスタンスの作成) を選択します。この場合、新しい EC2 インスタンスが DB クラスターと同じ VPC にあることを確認してください。

5. Continue (続行) をクリックします。

[Review and confirm] (確認と確定) ページが表示されます。



6. [Review and confirm] (確認と確定) ページで、EC2 インスタンスとの接続を設定するために RDS が行う変更を確認します。

   変更が正しければ、[確認とセットアップ] を選択します。

   変更内容が正しくない場合は、[Previous] (前へ) または [Cancel] (キャンセル) を選択します。

接続中のコンピューティングリソースを表示する

AWS Management Console を使用して、Aurora DB クラスターに接続されているコンピューティングリソースを表示できます。表示されるリソースには、自動的に設定されたコンピューティングリソース接続が含まれます。コンピューティングリソースとの接続は、次の方法で自動的に設定できます。

- データベースを作成するときに、コンピューティングリソースを選択できます。

  詳細については、「Amazon Aurora DB クラスターの作成」を参照してください。
- 既存のデータベースとコンピューティングリソース間の接続を設定できます。

  詳細については、「EC2 インスタンスと Aurora DB クラスターを自動的に接続する」を参照してください。

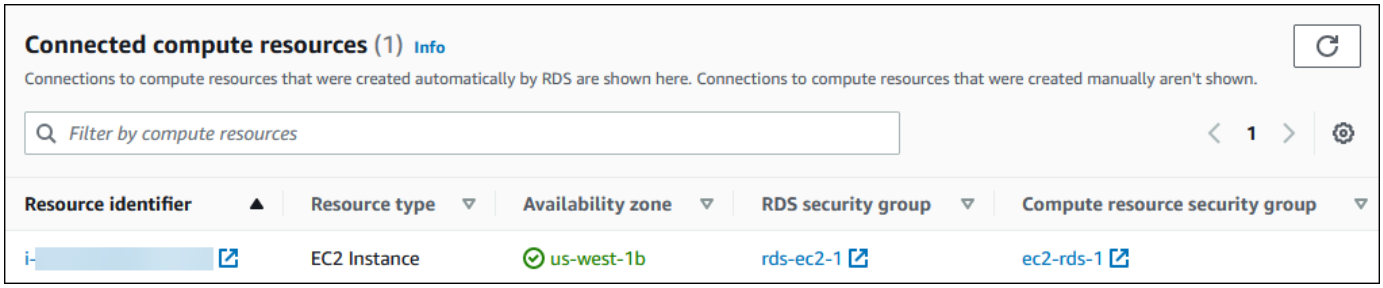
コンピューティングリソースリストには、手動でデータベースに接続されたものは含まれていません。例えば、データベースに関連付けられた VPC セキュリティグループにルールを追加することで、コンピューティングリソースがデータベースに手動でアクセスできるようになります。

コンピューティングリソースをリスト化するには、次の条件を満たしている必要があります。

- コンピューティングリソースに関連付けられているセキュリティグループの名前がパターン `ec2-rds-`*n* (*n* は数字) と一致する。
- コンピューティングリソースに関連付けられたセキュリティグループには、ポート範囲が DB クラスターが使用するポートに設定されたアウトバウンドルールがあります。
- コンピューティングリソースに関連付けられたセキュリティグループには、ソースが DB クラスターに関連付けられたセキュリティグループに設定されたアウトバウンドルールがある。
- DB クラスターに関連付けられたセキュリティグループの名前が、パターン `rds-ec2-`*n* (*n* は数字) に一致する。
- DB クラスターに関連付けられたセキュリティグループには、ポート範囲が DB クラスターが使用するポートに設定されたインバウンドルールがあります。
- DB クラスターに関連付けられたセキュリティグループには、ソースがコンピューティングリソースに関連付けられたセキュリティグループに設定されたインバウンドルールがある。

Aurora DB クラスターに接続されているコンピューティングリソースを表示するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Databases] (データベース) を選択し、 DB クラスターの名前を選択します。
3. [Connectivity & security] (接続とセキュリティ) タブの [Connected compute resources] (接続されたコンピューティングリソース) にコンピューティングリソースが表示されます。



特定の DB エンジンを実行している DB インスタンスに接続する

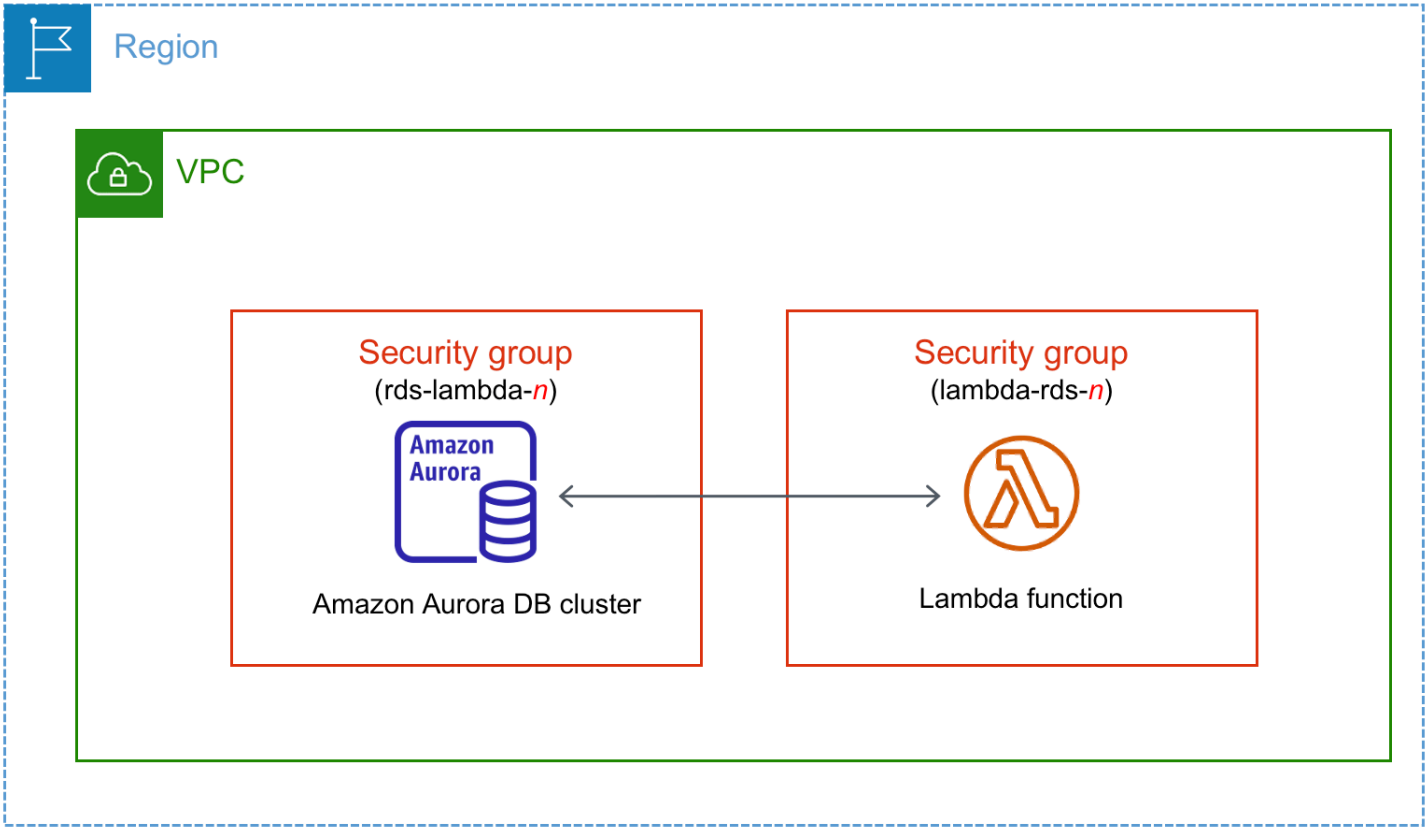
特定の DB エンジンを実行している DB インスタンスへの接続については、DB エンジンの指示に従ってください。

- Amazon Aurora MySQL DB クラスターへの接続
- Amazon Aurora PostgreSQL DB クラスターへの接続

Lambda 関数と Aurora DB クラスターを自動的に接続する

Amazon RDS コンソールを使用すると、Lambda 関数と Aurora DB クラスターとの接続を簡単に設定できます。多くの場合、DB クラスターは VPC 内のプライベートサブネットにあります。アプリケーションで Lambda 関数を使用すると、プライベート DB クラスターにアクセスできます。

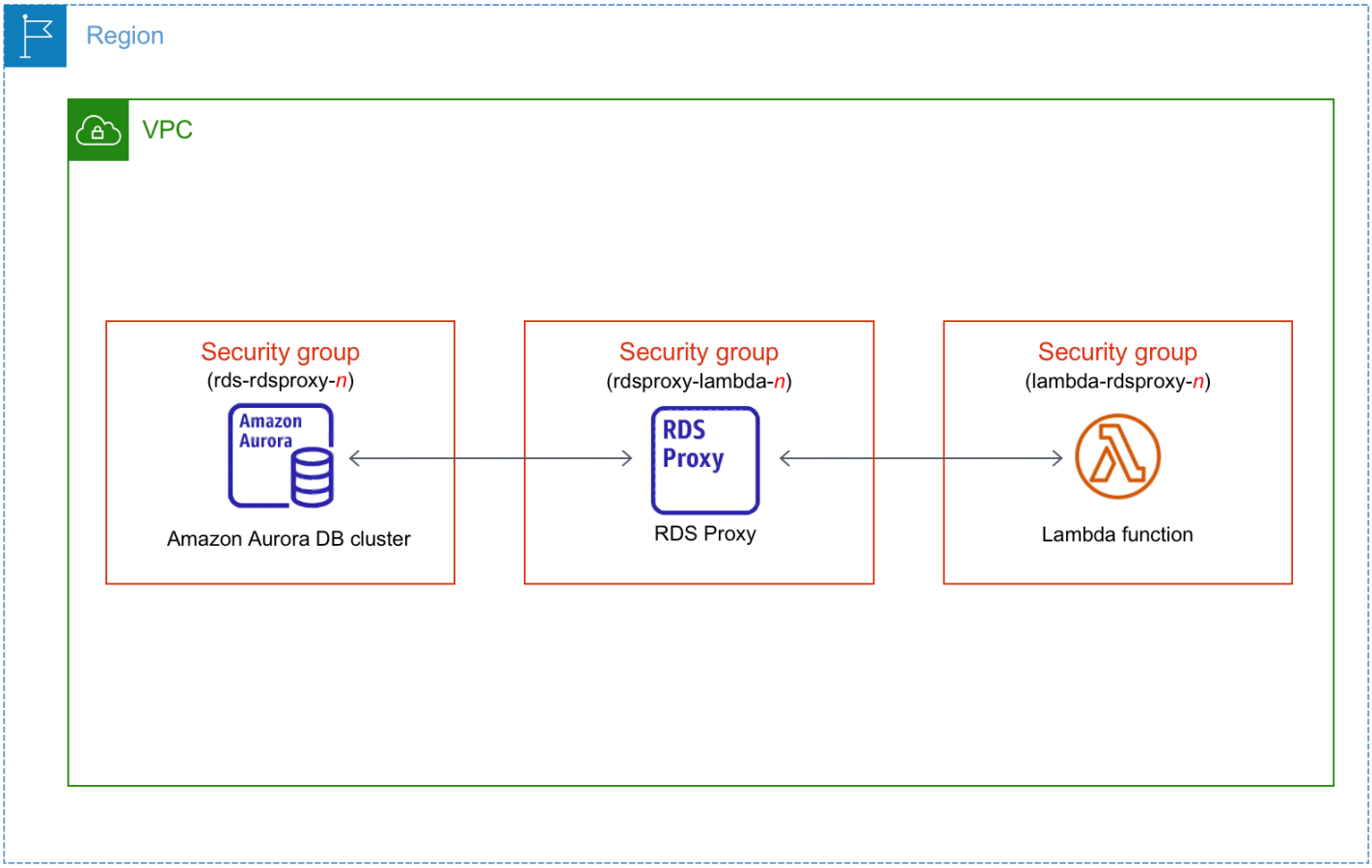
次の画像は、DB クラスターと Lambda 関数の間の直接接続を示しています。



Lambda 関数と DB クラスター間の RDS プロキシ経由の接続を設定して、データベースのパフォーマンスと耐障害性を改善できます。多くの場合、Lambda 関数は短いデータベース接続を頻繁に行い、RDS プロキシが提供する接続プールを使用することで利点を得られます。データベース認証情報を Lambda アプリケーションコードで管理する代わりに、Lambda 関数に設定済みの AWS Identity and Access Management IAM 認証を利用できます。詳細については、「Amazon RDS Proxy for Aurora の使用」を参照してください。

コンソールを使用して既存のプロキシに接続すると、Amazon RDS は、DB クラスターと Lambda 関数からの接続を許可するように、プロキシセキュリティグループを更新します。

同じコンソールページから新しいプロキシを作成することもできます。コンソールでプロキシを作成するときに、DB クラスターにアクセスするには、データベースの認証情報を入力するか、AWS Secrets Manager シークレットを選択する必要があります。



トピック

- Lambda 関数との自動接続の概要
- Lambda 関数と Aurora DB クラスター を自動的に接続する
- 接続中のコンピューティングリソースを表示する

Lambda 関数との自動接続の概要

Lambda 関数と Aurora DB クラスター を接続するための要件は次のとおりです。

- Lambda 関数は、DB クラスターと同じ VPC に存在する必要があります。
- 現在、DB クラスターを Aurora Serverless DB クラスターまたは Aurora グローバルデータベースの一部にすることはできません。
- 接続を設定するユーザーには、以下の Amazon RDS、Amazon EC2、Lambda、Secrets Manager、および IAM 操作を実行するアクセス許可が必要です。
  - Amazon RDS

    - rds:CreateDBProxies
    - rds:DescribeDBClusters
    - rds:DescribeDBProxies
    - rds:ModifyDBCluster
    - rds:ModifyDBProxy
    - rds:RegisterProxyTargets
- Amazon EC2
    - ec2:AuthorizeSecurityGroupEgress
    - ec2:AuthorizeSecurityGroupIngress
    - ec2:CreateSecurityGroup
    - ec2:DeleteSecurityGroup
    - ec2:DescribeSecurityGroups
    - ec2:RevokeSecurityGroupEgress
    - ec2:RevokeSecurityGroupIngress
- Lambda
    - lambda:CreateFunctions
    - lambda:ListFunctions
    - lambda:UpdateFunctionConfiguration
- Secrets Manager
    - secretsmanager:CreateSecret
    - secretsmanager:DescribeSecret
- IAM
    - iam:AttachPolicy
    - iam:CreateRole
    - iam:CreatePolicy
- AWS KMS
    - kms:describeKey

> **Note**
>
> DB クラスターと Lambda 関数が異なるアベイラビリティーゾーンにある場合、アベイラビリティーゾーン間のコストが発生する可能性があります。

Lambda 関数 Aurora DB クラスター間の接続を設定すると、Amazon RDS は、関数と DB クラスター の VPC セキュリティグループを設定します。RDS プロキシを使用する場合、Amazon RDS はプロキシの VPC セキュリティグループも設定します。Amazon RDS は、次の表で説明されているように、DB クラスター、Lambda 関数、およびプロキシに関連付けられたセキュリティグループの現在の設定に従って動作します。

現在の RDS セキュリティグループの設定	現在の Lambda セキュリティグループ設定	現在のプロキシセキュリティグループ設定	RDS アクション
DB クラスターには 1 つ以上のセキュリティグループが関連付けられ、その名前は、パターン `rds-lambda-` *n* に一致します。または、プロキシが既に DB クラスターに接続している場合、RDS は関連するプロキシの `TargetHealth` が `AVAILABLE` であるかどうかを確認します。 パターンに一致するセキュリティグループは変更されていません。このセキュリティグループには、	Lambda 関数に関連付けられたセキュリティグループが 1 つ以上あり、その名前はパターン `lambda-rds-` *n* または `lambda-rdsproxy-` *n* (*n* は数字) に一致します。 パターンに一致するセキュリティグループは変更されていません。このセキュリティグループには、DB クラスターまたはプロキシのいずれかの VPC セキュリティグループをデスティネーションとするアウトバウンドルール	プロキシに関連付けられたセキュリティグループが 1 つ以上あり、その名前はパターン `rdsproxy-lambda-` *n* (*n* は数字) に一致します。 パターンに一致するセキュリティグループは変更されていません。このセキュリティグループには、Lambda 関数と DB クラスター の VPC セキュリティグループに関するインバウンドルールとアウトバウンドルールがあります。	Amazon RDS は何もしません。 Lambda 関数、プロキシ (オプション)、および DB クラスター間の接続は既に自動的に設定されています。関数、プロキシ、およびデータベースの間には既に接続が存在するため、セキュリティグループは変更されません。

現在の RDS セキュリティグループの設定	現在の Lambda セキュリティグループ設定	現在のプロキシセキュリティグループ設定	RDS アクション
Lambda 関数またはプロキシの VPC セキュリティグループをソースとするインバウンドルールが 1 つのみ存在します。	が 1 つだけあります。		

現在の RDS セキュリティグループの設定	現在の Lambda セキュリティグループ設定	現在のプロキシセキュリティグループ設定	RDS アクション
次の条件のいずれかが適用されます。 • DB クラスターに関連付けられ、パターン `rds-lambda-` *n* に一致する名前を持つセキュリティグループはありません (または関連するプロキシの `TargetHealth` が `AVAILABLE` の場合)。 • DB クラスターに関連付けられ、パターン `rds-lambda-` *n* に一致する名前を持つ 1 つ以上のセキュリティグループがあります (または関連するプロキシの `TargetHealth` が `AVAILABLE` の場合)。ただし、これらのセキュリティグループは、いずれも Lambda 関数との接続には使用できません。	次の条件のいずれかが適用されます。 • Lambda 関数に関連付けられ、パターン `lambda-rds-` *n* または `lambda-rdsproxy-` *n* に一致する名前のセキュリティグループはありません。 • Lambda 関数に関連付けられたセキュリティグループが 1 つ以上あり、その名前はパターン `lambda-rds-` *n* または `lambda-rdsproxy-` *n* に一致します。ただし、Amazon RDS は、これらのセキュリティグループのいずれも、DB クラスターとの接続には使用できません。 Amazon RDS は、DB クラスターまたはプ	次の条件のいずれかが適用されます。 • プロキシに関連付けられ、パターン `rdsproxy-lambda-` *n* に一致する名前のセキュリティグループはありません。 • プロキシに関連付けられ、パターン `rdsproxy-lambda-` *n* に一致する名前のセキュリティグループが 1 つ以上あります。ただし、Amazon RDS は、これらのセキュリティグループのいずれも、DB クラスターまたは Lambda 関数との接続には使用できません。 Amazon RDS は、DB クラスターおよび Lambda 関数の VPC セキュリティグループに関するインバウンドルールとアウト	RDS action: create new security groups

現在の RDS セキュリティグループの設定	現在の Lambda セキュリティグループ設定	現在のプロキシセキュリティグループ設定	RDS アクション
Amazon RDS は、Lambda 関数またはプロキシの VPC セキュリティグループをソースとするインバウンドルールが 1 つも存在しないセキュリティグループを使用できません。また、Amazon RDS は、変更されたセキュリティグループを使用できません。変更の例としては、ルールの追加や、既存ルールのポート変更などがあります。	ロキシの VPC セキュリティグループをデスティネーションとするアウトバウンドルールが 1 つもないセキュリティグループを使用できません。また、Amazon RDS は、変更されたセキュリティグループを使用できません。	バウンドルールがないセキュリティグループを使用できません。また、Amazon RDS は、変更されたセキュリティグループを使用できません。	

現在の RDS セキュリティグループの設定	現在の Lambda セキュリティグループ設定	現在のプロキシセキュリティグループ設定	RDS アクション
DB クラスターに関連付けられ、パターン `rds-lambda-` *n* に一致する名前を持つ 1 つ以上のセキュリティグループがあります (または関連するプロキシの `TargetHealth` が `AVAILABLE` の場合)。 パターンに一致するセキュリティグループは変更されていません。このセキュリティグループには、Lambda 関数またはプロキシの VPC セキュリティグループをソースとするインバウンドルールが 1 つのみ存在します。	Lambda 関数に関連付けられたセキュリティグループが 1 つ以上あり、その名前はパターン `lambda-rds-` *n* または `lambda-rdsproxy-` *n* に一致します。 ただし、Amazon RDS は、これらのセキュリティグループのいずれも、DB クラスターとの接続には使用できません。Amazon RDS は、DB クラスターまたはプロキシの VPC セキュリティグループをデスティネーションとするアウトバウンドルールが 1 つもないセキュリティグループを使用できません。また、Amazon RDS は、変更されたセキュリティグループを使用できません。	プロキシに関連付けられ、パターン `rdsproxy-lambda-` *n* に一致する名前のセキュリティグループが 1 つ以上あります。 ただし、Amazon RDS は、これらのセキュリティグループのいずれも、DB クラスターまたは Lambda 関数との接続には使用できません。Amazon RDS は、DB クラスターおよび Lambda 関数の VPC セキュリティグループに関するインバウンドルールとアウトバウンドルールがないセキュリティグループを使用できません。また、Amazon RDS は、変更されたセキュリティグループを使用できません。	RDS action: create new security groups

現在の RDS セキュリティグループの設定	現在の Lambda セキュリティグループ設定	現在のプロキシセキュリティグループ設定	RDS アクション
DB クラスターに関連付けられ、パターン `rds-lambda-` *n* に一致する名前を持つ 1 つ以上のセキュリティグループがあります (または関連するプロキシの `TargetHealth` が `AVAILABLE` の場合)。 パターンに一致するセキュリティグループは変更されていません。このセキュリティグループには、Lambda 関数またはプロキシの VPC セキュリティグループをソースとするインバウンドルールが 1 つのみ存在します。	接続用の有効な Lambda セキュリティグループが存在しますが、Lambda 関数に関連付けられていません。このセキュリティグループには、パターン `lambda-rds-` *n* または `lambda-rdsproxy-` *n* に一致する名前が付いています。これは変更されていません。DB クラスターまたはプロキシの VPC セキュリティグループをデスティネーションとするアウトバウンドルールが 1 つだけあります。	接続に有効なプロキシセキュリティグループは存在しますが、プロキシに関連付けられていません。このセキュリティグループには、パターン `rdsproxy-lambda-` *n* に一致する名前が付いています。これは変更されていません。DB クラスターおよび Lambda 関数の VPC セキュリティグループに関するインバウンドルールとアウトバウンドルールがあります。	RDS action: associate Lambda security group

現在の RDS セキュリティグループの設定	現在の Lambda セキュリティグループ設定	現在のプロキシセキュリティグループ設定	RDS アクション
次の条件のいずれかが適用されます。 • DB クラスターに関連付けられ、パターン `rds-lambda-` *n* に一致する名前を持つセキュリティグループはありません (または関連するプロキシの `TargetHealth` が `AVAILABLE` の場合)。 • DB クラスターに関連付けられ、パターン `rds-lambda-` *n* に一致する名前を持つ 1 つ以上のセキュリティグループがあります (または関連するプロキシの `TargetHealth` が `AVAILABLE` の場合)。ただし、Amazon RDS は、これらのセキュリティグループを Lambda 関数またはプロキシと	Lambda 関数に関連付けられたセキュリティグループが 1 つ以上あり、その名前はパターン `lambda-rds-` *n* または `lambda-rdsproxy-` *n* に一致します。 パターンに一致するセキュリティグループは変更されていません。このセキュリティグループには、DB インスタンスまたはプロキシの VPC セキュリティグループをデスティネーションとするアウトバウンドルールが 1 つだけあります。	プロキシに関連付けられ、パターン `rdsproxy-lambda-` *n* に一致する名前のセキュリティグループが 1 つ以上あります。 パターンに一致するセキュリティグループは変更されていません。このセキュリティグループには、DB クラスターおよび Lambda 関数の VPC セキュリティグループに関するインバウンドルールとアウトバウンドルールがあります。	RDS action: create new security groups

現在の RDS セキュリティグループの設定	現在の Lambda セキュリティグループ設定	現在のプロキシセキュリティグループ設定	RDS アクション
の接続に使用できません。 Amazon RDS は、Lambda 関数またはプロキシの VPC セキュリティグループをソースとするインバウンドルールが 1 つも存在しないセキュリティグループを使用できません。また、Amazon RDS は、変更されたセキュリティグループを使用できません。			

現在の RDS セキュリティグループの設定	現在の Lambda セキュリティグループ設定	現在のプロキシセキュリティグループ設定	RDS アクション
次の条件のいずれかが適用されます。 • DB クラスターに関連付けられ、パターン `rds-lambda-` *n* に一致する名前を持つセキュリティグループはありません (または関連するプロキシの `TargetHealth` が `AVAILABLE` の場合)。 • DB クラスターに関連付けられ、パターン `rds-lambda-` *n* に一致する名前を持つ 1 つ以上のセキュリティグループがあります (または関連するプロキシの `TargetHealth` が `AVAILABLE` の場合)。ただし、Amazon RDS は、これらのセキュリティグループを Lambda 関数またはプロキシと	次の条件のいずれかが適用されます。 • Lambda 関数に関連付けられ、パターン `lambda-rds-` *n* または `lambda-rdsproxy-` *n* に一致する名前のセキュリティグループはありません。 • Lambda 関数に関連付けられたセキュリティグループが 1 つ以上あり、その名前はパターン `lambda-rds-` *n* または `lambda-rdsproxy-` *n* に一致します。ただし、Amazon RDS は、これらのセキュリティグループを DB DB クラスターとの接続に使用できません。 Amazon RDS は、DB クラスターまたはプロキシの VPC セキュ	次の条件のいずれかが適用されます。 • プロキシに関連付けられ、パターン `rdsproxy-lambda-` *n* に一致する名前のセキュリティグループはありません。 • プロキシに関連付けられ、パターン `rdsproxy-lambda-` *n* に一致する名前のセキュリティグループが 1 つ以上あります。ただし、Amazon RDS は、これらのセキュリティグループを DB クラスターまたは Lambda 関数との接続に使用することはできません。 Amazon RDS は、DB クラスターおよび Lambda 関数の VPC セキュリティグループに関するインバウンドルールとアウト	RDS action: create new security groups

現在の RDS セキュリティグループの設定	現在の Lambda セキュリティグループ設定	現在のプロキシセキュリティグループ設定	RDS アクション
の接続に使用できません。 Amazon RDS は、Lambda 関数またはプロキシの VPC セキュリティグループをソースとするインバウンドルールが 1 つも存在しないセキュリティグループを使用できません。また、Amazon RDS は、変更されたセキュリティグループを使用できません。	リティグループをソースとするアウトバウンドルールが 1 つもないセキュリティグループを使用できません。また、Amazon RDS は、変更されたセキュリティグループを使用できません。	バウンドルールがないセキュリティグループを使用できません。また、Amazon RDS は、変更されたセキュリティグループを使用できません。	

RDS アクション: 新しいセキュリティグループを作成する

Amazon RDS は以下のアクションを実行します。

- パターン `rds-lambda-`*n* または `rds-rdsproxy-`*n* (RDS プロキシを使用する場合) に一致する新しいセキュリティグループを作成します。このセキュリティグループには、Lambda 関数またはプロキシの VPC セキュリティグループをソースとするインバウンドルールがあります。このセキュリティグループは、DB クラスターに関連付けられ、関数またはプロキシが DB クラスターにアクセスすることを許可します。
- パターン `lambda-rds-`*n* または `lambda-rdsproxy-`*n* に一致する新しいセキュリティグループを作成します。このセキュリティグループには、DB クラスターまたはプロキシの VPC セキュリティグループをデスティネーションとするアウトバウンドルールがあります。このセキュリティグループは Lambda 関数に関連付けられ、関数が DB クラスター にトラフィックを送信するか、プロキシ経由でトラフィックを送信することを許可します。

- パターン `rdsproxy-lambda-`*n* に一致する新しいセキュリティグループを作成します。このセキュリティグループには、DB クラスターおよび Lambda 関数の VPC セキュリティグループに関するインバウンドルールとアウトバウンドルールがあります。

RDS アクション: Lambda セキュリティグループを関連付ける

Amazon RDS は、有効な既存の Lambda セキュリティグループを Lambda 関数に関連付けます。このセキュリティグループは、関数が DB クラスター にトラフィックを送信するか、プロキシ経由でトラフィックを送信することを許可します。

Lambda 関数と Aurora DB クラスター を自動的に接続する

Amazon RDS コンソールを使用して、Lambda 関数を DB クラスターに自動的に接続することができます。これにより、これらのリソース間の接続を設定するプロセスが簡単になります。

RDS プロキシを使用して、接続にプロキシを含めることもできます。Lambda 関数は短いデータベース接続を頻繁に行うため、RDS プロキシが提供する接続プールを使用することで利点を得られます。Lambda アプリケーションコードでデータベース認証情報を管理する代わりに、Lambda 関数用に設定済みの IAM 認証を使用することもできます。

[Lambda 接続の設定] ページを使用して、既存の DB クラスターを新規および既存の Lambda 関数に接続できます。セットアッププロセスでは、必要なセキュリティグループが自動的にセットアップされます。

Lambda 関数と DB クラスターの間の接続を設定する前に、次のことを確認してください。

- Lambda 関数と DB クラスターが同じ VPC にあります。
- ユーザーアカウントに適切なアクセス許可があります。要件の詳細については、「Lambda 関数との自動接続の概要」を参照してください。

接続の設定後にセキュリティグループを変更すると、Lambda 関数と DB クラスターとの接続に影響する可能性があります。

> **Note**
>
> AWS Management Console でのみ、DB クラスターと Lambda 関数の間の接続を自動的に設定できます。Lambda 関数を接続するには、DB クラスターのすべてのインスタンスが使用可能状態である必要があります。

Lambda 関数と Aurora DB クラスターを自動的に接続するには

<result>

設定を確認すると、Amazon RDS は、Lambda 関数、RDS プロキシ (プロキシを使用した場合)、および DB クラスターの接続プロセスを開始します。コンソールに [接続の詳細] ダイアログボックスが表示され、リソース間の接続を許可するセキュリティグループの変更が一覧表示されます。
</result>

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択し、Lambda 関数に接続するDB クラスターを選択します。
3. [アクション] として、[Lambda 接続の設定] を選択します。
4. [Lambda 接続の設定] ページの [Lambda 関数の選択] で、次のいずれかを実行します。
   - DB クラスターと同じ VPC に既存の Lambda 関数がある場合は、[既存の関数を選択] を選択し、関数を選択します。
   - 同じ VPC に Lambda 関数がない場合は、[新しい関数を作成] を選択し、[関数名] を入力します。デフォルトのランタイムは Nodejs.18 に設定されています。接続設定が完了した後、Lambda コンソールで新しい Lambda 関数の設定を変更できます。
5. (オプション) [RDS プロキシ] で、[RDS プロキシを使用して接続] を選択し、次のいずれかを実行します。
   - 使用する既存のプロキシがある場合は、[既存のプロキシを選択] を選択し、プロキシを選択します。
   - プロキシがなく、Amazon RDS にプロキシを自動的に作成させる場合は、[新しいプロキシの作成] を選択します。次に、[データベース認証情報] として、次のいずれかを実行します。
     a. [データベースのユーザー名とパスワード] を選択し、DB クラスターの [ユーザー名] と [パスワード] を入力します。
     b. [Secrets Manager シークレット] を選択します。次に、[シークレットを選択] で、AWS Secrets Manager シークレットを選択します。Secrets Manager シークレットがない場合は、[新しい Secrets Manager シークレットを作成] を選択して、新しいシークレットを作成します。シークレットを作成した後、[シークレットを選択] で、新しいシークレットを選択します。

新しいプロキシを作成した後、[既存のプロキシを選択] を選択し、プロキシを選択します。プロキシが接続可能になるまでに時間がかかる場合があることに注意してください。

6. (オプション) [接続の概要] を展開し、強調表示されているリソースの最新情報を確認します。
7. [Set up (セットアップ)] を選択します。

接続中のコンピューティングリソースを表示する

AWS Management Console を使用して、DB クラスターに接続されている Lambda 関数を確認できます。表示されるリソースには、Amazon RDS が自動的に設定したコンピューティングリソース接続が含まれます。

一覧表示されるコンピューティングリソースには、DB クラスターに手動で接続されたリソースは含まれていません。例えば、データベースに関連付けられた VPC セキュリティグループにルールを追加することで、コンピューティングリソースが DB クラスター に手動でアクセスするのを許可できます。

コンソールに Lambda 関数を一覧表示するには、以下の条件が適用される必要があります。

- コンピューティングリソースに関連付けられているセキュリティグループの名前がパターン `lambda-rds-`*n* または `lambda-rdsproxy-`*n* (*n* は数字) と一致します。
- コンピューティングリソースに関連付けられているセキュリティグループに、DB クラスターまたは該当するプロキシが使用するポートにポート範囲が設定されたアウトバウンドルールがあります。アウトバウンドルールのデスティネーションは、DB クラスターまたは関連するプロキシに関連付けられているセキュリティグループに設定されている必要があります。
- 構成にプロキシが含まれる場合、データベースに関連付けられたプロキシにアタッチされたセキュリティグループの名前は、パターン `rdsproxy-lambda-`*n* (*n* は数字) と一致します。
- 関数に関連付けられているセキュリティグループに、DB クラスターまたは該当するプロキシが使用するポートにポート範囲が設定されたアウトバウンドルールがあります。デスティネーションは、DB クラスターまたは関連するプロキシに関連付けられているセキュリティグループに設定されている必要があります。

DB クラスターに自動的に接続されたコンピューティングリソースを表示するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。

2. ナビゲーションペインで、[データベース] を選択し、DB インスタンスを選択します。
3. [接続とセキュリティ] タブの [接続されたコンピューティングリソース] にコンピューティングリソースが表示されます。

Amazon Aurora DB クラスターの変更

DB クラスターの設定を変更して、バックアップ保持期間やデータベースポートの変更などのタスクを実行できます。また、DB クラスターの DB インスタンスを変更して、DB インスタンスクラスの変更やそれに伴う Performance Insights の有効化などのタスクを実行することもできます。このトピックでは、Aurora DB クラスターやその DB インスタンスを変更する手順と、各設定について説明します。

本稼働の DB クラスターまたはインスタンスの変更前に、各変更の影響を完全に把握できるように、テストの DB クラスターまたは DB インスタンスで変更をテストすることをお勧めします。このテストは特に、データベースのバージョンをアップグレードするときに重要です。

トピック

コンソール、CLI、API を使用した DB クラスターの変更

DB クラスターは、AWS Management Console、AWS CLI、または RDS API を使用して変更できます。

> **Note**
>
> ほとんどの変更は、すぐに適用するか、スケジュールされた次のメンテナンス期間に適用できます。削除保護の有効化など、一部の変更は、選択した適用時期に関係なく、すぐに適用されます。
>
> AWS Management Console でのマスターパスワードの変更は、常に直ちに適用されます。ただし、AWS CLI または RDS API を使用するときには、この変更をすぐに適用するか、次回の定期メンテナンス期間中に適用するかを選択できます。
>
> SSL エンドポイントを使用しており、DB クラスター識別子を変更する場合は、DB クラスターを停止して再起動し、SSL エンドポイントを更新します。詳細については、「Amazon Aurora DB クラスターの停止と開始」を参照してください。

コンソール

DB クラスターを変更するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[データベース] を選択し、変更する DB クラスターを選択します。
3. [Modify] を選択します。[DB クラスターの変更] ページが表示されます。
4. 必要に応じて任意の設定を変更してください。各設定の詳細については、「Amazon Aurora の設定」を参照してください。

   > **Note**
   >
   > AWS Management Console で、インスタンスレベルの変更は、現在の DB インスタンスにのみ適用される場合や、DB クラスター全体に適用される場合があります。設定が DB インスタンスまたは DB クラスターに適用されるかどうかについての詳細は、Amazon Aurora の設定 に設定されているスコープを参照してください。インスタンスレベルで DB クラスター全体を変更する設定を AWS Management Console で変更するには、「DB クラスター内の DB インスタンスの変更」の手順に従います。

5. すべての変更が正しいことを確認したら、[Continue] を選択して変更の概要を確認します。
6. 変更をすぐに適用するには、[すぐに適用] を選択します。
7. 確認ページで、変更内容を確認します。正しい場合は、[クラスターの変更] を選択して変更を保存します。

   または、[Back] を選択して変更を編集するか、[Cancel] を選択して変更をキャンセルします。

AWS CLI

AWS CLI を使用して DB クラスターを変更するには、modify-db-cluster コマンドを呼び出します。DB クラスター識別子と、変更する設定の値を指定します。各設定の詳細については、「Amazon Aurora の設定」を参照してください。

> **Note**
>
> 一部の設定は、DB インスタンスにのみ適用されます。これらの設定を変更するには、「DB クラスター内の DB インスタンスの変更」の手順に従います。

Example

次のコマンドでは、バックアップ保存期間を 1 週間 (7 日間) に設定して、mydbcluster を変更します。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --backup-retention-period 7
```

Windows の場合:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --backup-retention-period 7
```

RDS API

Amazon RDS API を使用して DB クラスターを変更するには、ModifyDBCluster オペレーションを呼び出します。DB クラスター識別子と、変更する設定の値を指定します。各パラメータの詳細については、「Amazon Aurora の設定」を参照してください。

> **Note**
>
> 一部の設定は、DB インスタンスにのみ適用されます。これらの設定を変更するには、「DB クラスター内の DB インスタンスの変更」の手順に従います。

DB クラスター内の DB インスタンスの変更

DB クラスター内の DB インスタンスは、AWS Management Console、AWS CLI、または RDS API を使用して変更できます。

DB インスタンスを変更する際に、変更内容を即時に適用することができます。変更内容を即時適用するには、AWS Management Console で [Apply Immediately] (すぐに適用) オプションを選択するか、AWS CLI を呼び出して `--apply-immediately` パラメータを使用するか、Amazon RDS API を使用して `ApplyImmediately` パラメータを `true` に設定するかします。

変更をすぐに適用しないと、変更は次のメンテナンス期間まで延期されます。延期された変更は、次のメンテナンス期間中に適用されます。変更をすぐに適用すると、新しい変更と延期された変更が適用されます。

次のメンテナンスウィンドウに向けて保留中の変更を確認するには、describe-db-clusters AWS CLI コマンドを使用して `PendingModifiedValues` フィールドを確認します。

> **Important**
>
> ダウンタイムを要する延期中の変更がある場合、[すぐに適用] を選択すると、DB インスタンスで予定外のダウンタイムが発生することがあります。DB クラスターの他の DB インスタンスではダウンタイムはありません。
> 延期した変更は、`describe-pending-maintenance-actions` CLI コマンドの出力には表示されません。メンテナンスアクションに含まれるのは、次のメンテナンス期間にスケジュールされているシステムアップグレードのみです。

コンソール

DB クラスター内の DB インスタンスを変更するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[データベース] を選択し、変更する DB インスタンスを選択します。
3. [アクション]、[変更] の順に選択します。Modify DB instance ページが表示されます。
4. 必要に応じて任意の設定を変更してください。各設定の詳細については、「Amazon Aurora の設定」を参照してください。

> **Note**
>
> 一部の設定は DB クラスター全体に適用されるため、クラスターレベルで変更する必要があります。これらの設定を変更するには、「コンソール、CLI、API を使用した DB クラスターの変更」の手順に従います。
> AWS Management Console で、インスタンスレベルの変更は、現在の DB インスタンスにのみ適用される場合や、DB クラスター全体に適用される場合があります。設定が DB インスタンスまたは DB クラスターに適用されるかどうかについての詳細は、Amazon Aurora の設定 に設定されているスコープを参照してください。

5. すべての変更が正しいことを確認したら、[Continue] を選択して変更の概要を確認します。
6. 変更をすぐに適用するには、[すぐに適用] を選択します。
7. 確認ページで、変更内容を確認します。正しい場合は、[Modify DB Instance (DB インスタンスを変更)] を選択して変更を保存します。

   または、[Back] を選択して変更を編集するか、[Cancel] を選択して変更をキャンセルします。

AWS CLI

AWS CLI を使用して DB クラスターで DB インスタンスを変更するには、modify-db-instance コマンドを呼び出します。DB インスタンス識別子と、変更する設定の値を指定します。各パラメータの詳細については、「Amazon Aurora の設定」を参照してください。

> **Note**
>
> 一部の設定は、DB クラスター全体に適用されます。これらの設定を変更するには、「コンソール、CLI、API を使用した DB クラスターの変更」の手順に従います。

Example

次のコードは、DB インスタンスクラスを mydbinstance に設定して、db.r4.xlarge を変更します。変更は、--no-apply-immediately を使用して次のメンテナンスウィンドウ中に適用されます。今すぐ変更を適用するには、--apply-immediately を使用します。

Linux、macOS、Unix の場合:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --db-instance-class db.r4.xlarge \
    --no-apply-immediately
```

Windows の場合:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --db-instance-class db.r4.xlarge ^
    --no-apply-immediately
```

RDS API

Amazon RDS API を使用して DB インスタンスを変更するには、ModifyDBInstance オペレーションを呼び出します。DB インスタンス識別子と、変更する設定の値を指定します。各パラメータの詳細については、「Amazon Aurora の設定」を参照してください。

> **Note**
>
> 一部の設定は、DB クラスター全体に適用されます。これらの設定を変更するには、「コンソール、CLI、API を使用した DB クラスターの変更」の手順に従います。

データベースマスターユーザーのパスワードの変更

AWS Management Console または AWS CLI を使用して、マスターユーザーのパスワードを変更することができます。

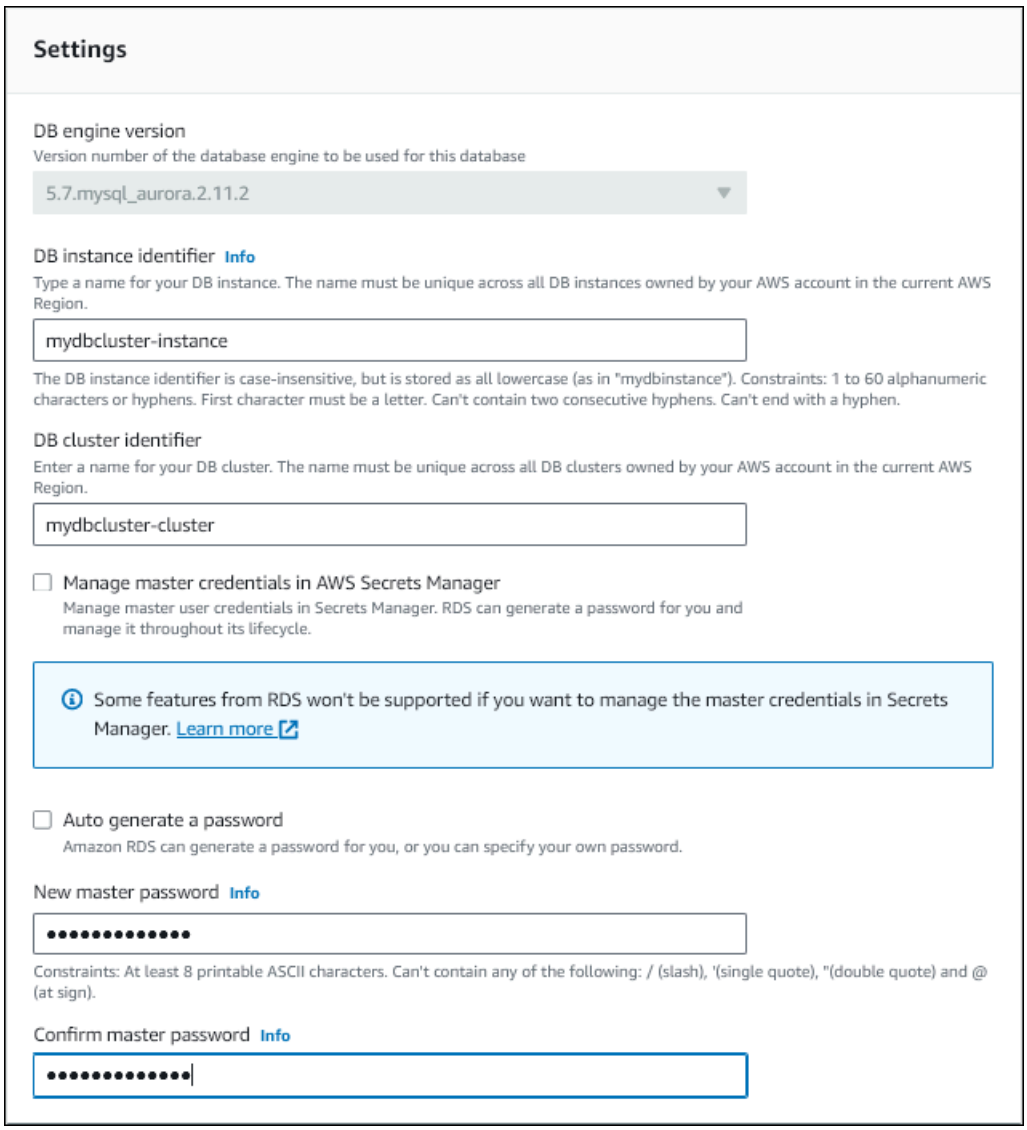
コンソール

AWS Management Console を使用して、ライター DB インスタンスを修正し、マスターユーザーパスワードを変更します。

マスターユーザーのパスワードを変更するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[データベース] を選択し、変更する DB インスタンスを選択します。
3. [アクション]、[変更] の順に選択します。

   Modify DB instance ページが表示されます。
4. [新しいマスターパスワード] を入力します。
5. [マスターパスワードの確認] に同じ新しいパスワードを入力します。



6. [続行] を選択して、変更の概要を確認します。

> **Note**
>
> パスワードの変更は常に即時適用されます。

7. 確認ページで、[DB インスタンスの変更] を選択します。

CLI

AWS CLI を使用してマスターユーザーパスワードを変更するには、modify-db-cluster コマンドを呼び出します。次の例に示すように、DB クラスター識別子と新しいパスワードを指定します。

パスワードの変更は常に即時に適用されるため、`--apply-immediately|--no-apply-immediately` を指定する必要はありません。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --master-user-password mynewpassword
```

Windows の場合:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --master-user-password mynewpassword
```

Amazon Aurora の設定

次の表には、変更ができる設定、設定を変更する方法、設定のスコープに関する詳細が含まれます。スコープは、設定が DB クラスター全体に適用されるか、または特定の DB インスタンスに対してのみ設定できるのかどうかを決定します。

> **Note**
>
> Aurora Serverless v1 または Aurora Serverless v2 DB クラスターを変更する場合は、追加の設定ができます。これらの設定の詳細については、「Aurora Serverless v1 DB クラスターの変更」および「Aurora Serverless v2 DB クラスターの管理」を参照してください。
> 制限により、Aurora Serverless v1 または Aurora Serverless v2 では一部の設定を使用できません。詳細については、Aurora Serverless v1 の制約事項およびAurora Serverless v2 の要件と制限を参照してください。

設定と説明	方法	スコープ	ダウンタイムに関する注意
マイナーバージョン自動アップグレード	**Note** この設定はデフォルトで有	DB クラスター全体	この変更時に機能停止は発生しません。Aurora で自動アップグレードを適用する

設定と説明	方法	スコープ	ダウンタイムに関する注意
指定したマイナーエンジンバージョンアップグレードをリリース時に自動的に DB インスタンスに適用するかどうか。アップグレードはスケジュールされたメンテナンスウィンドウ中にのみインストールされます。 エンジンの更新に関する詳細は、「Amazon Aurora PostgreSQL の更新」と「Amazon Aurora MySQL のデータベースエンジンの更新」を参照してください。Aurora MySQL の [マイナーバージョン自動アップグレード] 設定の詳細については、「Aurora MySQL マイナーバージョン間の自動アップグレードの有効化」を参照してください。	効になっています。新しいクラスターごとに、その重要性、予想される有効期間、各アップグレード後に行う検証テストの量に基づいて、この設定の適切な値を選択します。 この設定を変更する場合は、Aurora クラスター内のすべての DB インスタンスに対して、この変更を実行します。この設定がオフになっている DB インスタンスがクラスター内にある場合、クラスターは自動アップグレードされません。 AWS Management Console の使用、DB クラスター内の DB インスタンスの変更。		と、将来のメンテナンス期間中に停止が発生します。

設定と説明	方法	スコープ	ダウンタイムに関する注意
	AWS CLI を使用して、modify-db-instance を実行し、--auto-minor-version-upgrade|--no-auto-minor-version-upgrade オプションを設定します。 RDS API を使用して、ModifyDBInstance を呼び出し、AutoMinorVersionUpgrade パラメータを設定します。		

設定と説明	方法	スコープ	ダウンタイムに関する注意
バックアップの保存期間 自動バックアップが保持される日数。最小値は 1 です。 詳細については、「バックアップ」を参照してください。	AWS Management Console の使用、コンソール、CLI、API を使用した DB クラスターの変更。 AWS CLI を使用して、modify-db-cluster を実行し、--backup-retention-period オプションを設定します。 RDS API を使用して、ModifyDBCluster を呼び出し、BackupRetentionPeriod パラメータを設定します。	DB クラスター全体	この変更時に機能停止は発生しません。

設定と説明	方法	スコープ	ダウンタイムに関する注意
バックアップウィンドウ (開始時刻) データベースの自動バックアップが実行される時間範囲。バックアップウィンドウは、協定世界時 (UTC) の開始時間で、時間単位での実行期間です。 Aurora バックアップは継続的かつ漸増的ですが、バックアップウィンドウは、バックアップ保持期間内に保持される毎日のシステムバックアップを作成するために使用されます。保持期間外に保持するには、これをコピーします。 DB クラスターのメンテナンスウィンドウとバックアップウィンドウは、重複させることはできません。 詳細については、「バックアップウィン	AWS Management Console の使用、コンソール、CLI、API を使用した DB クラスターの変更。 AWS CLI を使用して、`modify-db-cluster` を実行し、`--preferred-backup-window` オプションを設定します。 RDS API を使用して、`ModifyDBCluster` を呼び出し、`PreferredBackupWindow` パラメータを設定します。	DB クラスター全体。	この変更時に機能停止は発生しません。

設定と説明	方法	スコープ	ダウンタイムに関する注意
ドウ」を参照してください。			
容量設定 Aurora Serverless v1 DB クラスターのスケーリングプロパティ。serverless DB エンジンモードで DB クラスターのスケーリングプロパティのみ変更できます。 Aurora Serverless v1 の詳細については、「Amazon Aurora Serverless v1 の使用」を参照してください。	AWS Management Console の使用、コンソール、CLI、API を使用した DB クラスターの変更。 AWS CLI を使用して、modify-db-cluster を実行し、--scaling-configuration オプションを設定します。 RDS API を使用して、ModifyDBCluster を呼び出し、ScalingConfiguration パラメータを設定します。	DB クラスター全体	この変更時に機能停止は発生しません。 変更はただちに発生します。この設定は、[Apply immediately (すぐに適用)] 設定を無視します。

設定と説明	方法	スコープ	ダウンタイムに関する注意
認証局 DB インスタンスによって使用されるサーバー証明書の認定機関 (CA)。	AWS Management Console の使用、DB クラスター内の DB インスタンスの変更。 AWS CLI を使用して、`modify-db-instance` を実行し、`--ca-certificate-identifier` オプションを設定します。 RDS API を使用して、`ModifyDBInstance` を呼び出し、`CACertificateIdentifier` パラメータを設定します。	指定された DB インスタンスのみ	機能停止は、DB エンジンが再起動なしのローテーションをサポートしていない場合にのみ発生します。describe-db-engine-versions AWS CLI コマンドを使用すると、DB エンジンが再起動なしのローテーションをサポートしているかどうかを判断できます。

設定と説明	方法	スコープ	ダウンタイムに関する注意
クラスターのストレージ設定 DB クラスターのストレージタイプ: Aurora I/O-Optimized または Aurora Standard。 詳細については、「Amazon Aurora DB クラスターのストレージ設定」を参照してください。	AWS Management Console の使用、コンソール、CLI、API を使用した DB クラスターの変更。 AWS CLI を使用して、`modify-db-cluster` を実行し、`--storage-type` オプションを設定します。 RDS API を使用して、`ModifyDBCluster` を呼び出し、`StorageType` パラメータを設定します。	DB クラスター全体	この変更時に機能停止は発生しません。

設定と説明	方法	スコープ	ダウンタイムに関する注意
Copy tags to snapshots 選択すると、この DB クラスターに対して定義されたタグが、この DB クラスターから作成された DB スナップショットにコピーされます。詳細については、「Amazon RDS リソースのタグ付け」を参照してください。	AWS Management Console の使用、コンソール、CLI、API を使用した DB クラスターの変更。 AWS CLI を使用して、`modify-db-cluster` を実行し、`--copy-tags-to-snapshot` オプションまたは `--no-copy-tags-to-snapshot` オプションを設定します。 RDS API を使用して、`ModifyDBCluster` を呼び出し、`CopyTagsToSnapshot` パラメータを設定します。	DB クラスター全体	この変更時に機能停止は発生しません。

設定と説明	方法	スコープ	ダウンタイムに関する注意
Data API Aurora Serverless v1 には、ウェブサービスベースのアプリケーション (AWS Lambda、AWS AppSync など) を使用してアクセスできます。 この設定は、Aurora Serverless v1 DB クラスターにのみ適用されます。 詳細については、「RDS Data API の使用」を参照してください。	AWS Management Console の使用、コンソール、CLI、API を使用した DB クラスターの変更。 AWS CLI を使用して、`modify-db-cluster` を実行し、`--enable-http-endpoint` オプションを設定します。 RDS API を使用して、`ModifyDBCluster` を呼び出し、`EnableHttpEndpoint` パラメータを設定します。	DB クラスター全体	この変更時に機能停止は発生しません。

設定と説明	方法	スコープ	ダウンタイムに関する注意
データベース認証 使用するデータベース認証。 MySQL の場合: • データベースパスワードのみを使用してデータベースのユーザーを認証するには、[パスワード認証] を選択します。 • IAM ユーザーおよびルールでデータベースパスワードとユーザー認証情報を使用してデータベースユーザーを認証するには、[パスワードと IAM データベース認証] を選択します。詳細については、「の IAM データベース認証」を参照してください。 PostgreSQL の場合: • ユーザーおよびロールでデータベースパスワー	AWS Management Console の使用、コンソール、CLI、API を使用した DB クラスターの変更。 AWS CLI を使用して、modify-db-cluster を実行し、次のオプションを設定します。 • IAM 認証の場合は、`--enable-iam-database-authentication|--no-enable-iam-database-authentication` オプションを設定します。 • Kerberos 認証の場合は、`--domain` と `--domain-iam-role-name` オプションを設定します。 RDS API を使用して、ModifyDBCluster を呼び出して、次の	DB クラスター全体	この変更時に機能停止は発生しません。

設定と説明	方法	スコープ	ダウンタイムに関する注意
ドとユーザー認証情報を使用してデータベースユーザーを認証するには、[IAM database authentication] (IAM データベース認証) を選択します。詳細については、「の IAM データベース認証」を参照してください。 • Kerberos 認証を使用してデータベースパスワードとユーザー認証情報を認証するには、[Kerberos 認証] を選択します。詳細については、「Aurora PostgreSQL で Kerberos 認証を使用する」を参照してください。	パラメータを設定します。 • IAM 認証の場合は、`EnableIAMDatabaseAuthentication` パラメータを設定します。 • Kerberos 認証の場合は、`Domain` と `DomainIAMRoleName` パラメータを設定します。		

設定と説明	方法	スコープ	ダウンタイムに関する注意
データベースポート DB クラスターへのアクセスに使用するポート。	AWS Management Console の使用、コンソール、CLI、API を使用した DB クラスターの変更。 AWS CLI を使用して、`modify-db-cluster` を実行し、`--port` オプションを設定します。 RDS API を使用して、`ModifyDBCluster` を呼び出し、`Port` パラメータを設定します。	DB クラスター全体	この変更中に、機能停止が発生します。DB クラスター内のすべての DB インスタンスは、すぐに再起動されます。

設定と説明	方法	スコープ	ダウンタイムに関する注意
DB クラスター識別子 DB クラスター識別子。この値は小文字で保存されます。 DB クラスター識別子を変更すると、DB クラスターエンドポイントが変更されます。DB クラスター内の DB インスタンスのエンドポイントは変更されません。	AWS Management Console の使用、コンソール、CLI、API を使用した DB クラスターの変更。 AWS CLI を使用して、`modify-db-cluster` を実行し、`--new-db-cluster-identifier` オプションを設定します。 RDS API を使用して、`ModifyDBCluster` を呼び出し、`NewDBClusterIdentifier` パラメータを設定します。	DB クラスター全体	この変更時に機能停止は発生しません。

設定と説明	方法	スコープ	ダウンタイムに関する注意
DB クラスターのパラメータグループ DB クラスターに関連付ける DB クラスターのパラメータグループ。 詳細については、「「パラメータグループを使用する」」を参照してください。	AWS Management Console の使用、コンソール、CLI、API を使用した DB クラスターの変更。 AWS CLI を使用して、modify-db-cluster を実行し、--db-cluster-parameter-group-name オプションを設定します。 RDS API を使用して、ModifyDBCluster を呼び出し、DBClusterParameterGroupName パラメータを設定します。	DB クラスター全体	この変更時に機能停止は発生しません。パラメータグループを変更すると、一部のパラメータの変更は、再起動なしで DB クラスター内の DB インスタンスに即時に適用されます。他のパラメータの変更は、DB クラスター内の DB インスタンスが再起動された後でのみ適用されます。

設定と説明	方法	スコープ	ダウンタイムに関する注意
DB インスタンスクラス 使用する DB インスタンスクラス。 詳細については、「Aurora DB インスタンスクラス」を参照してください。	AWS Management Console の使用、DB クラスター内の DB インスタンスの変更。 AWS CLI を使用して、`modify-db-instance` を実行し、`--db-instance-class` オプションを設定します。 RDS API を使用して、`ModifyDBInstance` を呼び出し、`DBInstanceClass` パラメータを設定します。	指定された DB インスタンスのみ	この変更中に、機能停止が発生します。

設定と説明	方法	スコープ	ダウンタイムに関する注意
DB インスタンス識別子 DB インスタンス識別子。この値は小文字で保存されます。	AWS Management Console の使用、DB クラスター内の DB インスタンスの変更。 AWS CLI を使用して、modify-db-instance を実行し、--new-db-instance-identifier オプションを設定します。 RDS API を使用して、ModifyDBInstance を呼び出し、NewDBInstanceIdentifier パラメータを設定します。	指定された DB インスタンスのみ	DB エンジンのバージョンが動的 SSL アップロードをサポートしていない限り、この変更中にダウンタイムが発生します。バージョンが再起動を必要とするかどうかを判断するには、次の AWS CLI コマンドを実行します。 `aws rds describe-db-engine-versions \ --default-only \ --engine your-db-engine \ --query 'DBEngineVersions[*].SupportsCertificateRotationWithoutRestart'` ただし、DB インスタンスを再起動して以下を更新する必要があります。 • Aurora MySQL – information_schema.replica_

設定と説明	方法	スコープ	ダウンタイムに関する注意
			`host_status` テーブルの `SERVER_ID` 列 • Aurora PostgreSQL – `aurora_replica_status()` 関数の `server_id` 列

設定と説明	方法	スコープ	ダウンタイムに関する注意
DB パラメータグループ DB インスタンスに関連付ける DB パラメータグループ。 詳細については、「「パラメータグループを使用する」」を参照してください。	AWS Management Console の使用、DB クラスター内の DB インスタンスの変更。 AWS CLI を使用して、`modify-db-instance` を実行し、`--db-parameter-group-name` オプションを設定します。 RDS API を使用して、`ModifyDBInstance` を呼び出し、`DBParameterGroupName` パラメータを設定します。	指定された DB インスタンスのみ	この変更時に機能停止は発生しません。 新しい DB パラメータグループを DB インスタンスに関連付ける場合、変更された静的パラメータと動的パラメータは、DB インスタンスが再起動された後にのみ適用されます。ただし、DB インスタンスに関連付けた後に DB パラメータグループの動的パラメータを変更すると、これらの変更は再起動せずに直ちに適用されます。 詳細については、「パラメータグループを使用する」および Amazon Aurora DB クラスターまたは Amazon Aurora DB インスタンスの再起動を参照してください。

設定と説明	方法	スコープ	ダウンタイムに関する注意
削除保護 削除保護を有効にして、DB クラスターが削除されないようにします。詳細については、「Aurora クラスターの削除保護」を参照してください。	AWS Management Console の使用、コンソール、CLI、API を使用した DB クラスターの変更。 AWS CLI を使用して、modify-db-cluster を実行し、--deletion-protection|--no-deletion-protection オプションを設定します。 RDS API を使用して、ModifyDBCluster を呼び出し、DeletionProtection パラメータを設定します。	DB クラスター全体	この変更時に機能停止は発生しません。

設定と説明	方法	スコープ	ダウンタイムに関する注意
エンジンバージョン 使用する DB エンジンのバージョン。本稼働 DB クラスターをアップグレードする前に、テスト DB クラスターでアップグレードプロセスをテストして、所要時間の確認とアプリケーションの検証をすることをお勧めします。	AWS Management Console の使用、コンソール、CLI、API を使用した DB クラスターの変更。 AWS CLI を使用して、`modify-db-cluster` を実行し、`--engine-version` オプションを設定します。 RDS API を使用して、`ModifyDBCluster` を呼び出し、`EngineVersion` パラメータを設定します。	DB クラスター全体	この変更中に、機能停止が発生します。

設定と説明	方法	スコープ	ダウンタイムに関する注意
拡張モニタリング [拡張モニタリングを有効にする] を選択すると、DB インスタンスが実行されているオペレーティングシステムに対してリアルタイムでのメトリクスの収集が有効になります。 詳細については、「拡張モニタリングを使用した OS メトリクスのモニタリング」を参照してください。	AWS Management Console の使用、DB クラスター内の DB インスタンスの変更。 AWS CLI を使用して、modify-db-instance を実行し、--monitoring-role-arn および --monitoring-interval オプションを設定します。 RDS API を使用して、ModifyDBInstance を呼び出し、MonitoringRoleArn および MonitoringInterval パラメータを設定します。	指定された DB インスタンスのみ	この変更時に機能停止は発生しません。

設定と説明	方法	スコープ	ダウンタイムに関する注意
ログのエクスポート Amazon CloudWatch Logs に発行するログタイプを選択します。 詳細については、「Aurora MySQL データベースのログファイル」を参照してください。	AWS Management Console の使用、コンソール、CLI、API を使用した DB クラスターの変更。 AWS CLI を使用して、modify-db-cluster を実行し、--cloudwatch-logs-export-configuration オプションを設定します。 RDS API を使用して、ModifyDBCluster を呼び出し、CloudwatchLogsExportConfiguration パラメータを設定します。	DB クラスター全体	この変更時に機能停止は発生しません。

設定と説明	方法	スコープ	ダウンタイムに関する注意
メンテナンスウィンドウ システムメンテナンスを実行する時間帯。該当する場合は、システムメンテナンスにはアップグレードが含まれます。メンテナンス時間は、協定世界時 (UTC) の開始時間で、時間単位での実行期間です。 そのウィンドウを現在の時刻に設定した場合、保留中の変更が確実に適用されるように、現在の時刻からウィンドウの終わりまで 30 分以上必要です。 DB クラスターおよび DB クラスターの各 DB インスタンスのそれぞれに対してメンテナンス時間を設定できます。変更のスコープが DB クラスター全体である場合、変更は DB クラスターのメンテナンス	AWS Management Console を使用して、DB クラスターのメンテナンス時間を変更するには、コンソール、CLI、API を使用した DB クラスターの変更。 AWS Management Console を使用して、DB インスタンスのメンテナンス時間を変更するには、DB クラスター内の DB インスタンスの変更。 AWS CLI を使用して DB クラスターのメンテナンス時間を変更するには、`modify-db-cluster` を実行して、`--preferred-maintenance-window` オプションを設定します。 AWS CLI を使用して DB インスタンスのメンテナンス時間を変更するには、`modify-db-instance` を実行し	DB クラスター全体または単一の DB インスタンス	機能停止を引き起こす保留中のアクションが 1 つ以上あり、現在の時刻を含むようにメンテナンス時間を変更した場合、それらの保留中のアクションはすぐに適用され、機能停止は発生します。

設定と説明	方法	スコープ	ダウンタイムに関する注意
時間中に実行されます。変更のスコープが DB インスタンスである場合、変更は DB インスタンスのメンテナンス時間中に実行されます。 DB クラスターのメンテナンスウィンドウとバックアップウィンドウは、重複させることはできません。 詳細については、「Amazon RDS メンテナンスウィンドウ」を参照してください。	て、--preferred-maintenance-window オプションを設定します。 RDS API を使用して DB クラスターのメンテナンス時間を変更するには、ModifyDBCluster を呼び出して、PreferredMaintenanceWindow パラメータを設定します。 RDS API を使用して DB インスタンスのメンテナンス時間を変更するには、ModifyDBInstance を呼び出して、PreferredMaintenanceWindow パラメータを設定します。		

設定と説明	方法	スコープ	ダウンタイムに関する注意
AWS Secrets Manager でマスター認証情報を管理する [Manage master credentials in AWS Secrets Manager] (でマスター認証情報を管理する) を選択して、Secrets Manager でユーザーのパスワードをシークレットに管理します。 オプションで、シークレットを保護するために使用する KMS キーを選択します。お客様のアカウントの KMS キーから選択するか、別のアカウントからキーを入力します。 詳細については、「Amazon Aurora および AWS Secrets Manager によるパスワード管理」を参照してください。 Aurora によって既に DB クラスターのマスターユーザーのパスワードを管理して	AWS Management Console の使用、DB クラスター内の DB インスタンスの変更。 AWS CLI を使用して、modify-db-cluster を実行し、--manage-master-user-password | --no-manage-master-user-password および --master-user-secret-kms-key-id オプションを設定します。マスターユーザーのパスワードをすぐにローテーションするには、--rotate-master-user-password オプションを設定します。 RDS API を使用して、ModifyDBCluster を呼び出し、ManageMasterUserPassword および MasterUserSecretKm	DB クラスター全体	この変更時に機能停止は発生しません。

設定と説明	方法	スコープ	ダウンタイムに関する注意
いる場合は、[Rotate secret immediately] (すぐにシークレットをローテーションする) を選択してマスターユーザーパスワードをすぐにローテーションできます。 詳細については、「Amazon Aurora および AWS Secrets Manager によるパスワード管理」を参照してください。	sKeyId パラメータを設定します。マスターユーザーのパスワードをすぐにローテーションするには、RotateMasterUserPassword パラメータを true に設定します。		

設定と説明	方法	スコープ	ダウンタイムに関する注意
ネットワークの種類 DB クラスターでサポートされている IP アドレス設定プロトコル。 リソースが、IPv4 アドレス設定プロトコル経由でのみ DB クラスターと通信できるように指定する IPv4。 リソースが、IPv4、IPv6、またはその両方で DB クラスター通信できるように指定するデュアルスタックモード。IPv6 アドレス設定プロトコルで DB クラスターと通信する必要があるリソースがある場合は、デュアルスタックモードを使用します。デュアルスタックモードを使用するには、IPv4 と IPv6 の両方のネットワークプロトコルをサポートする 2 つのアベイラビリティーゾーンにまたがるサ	AWS Management Console の使用、コンソール、CLI、API を使用した DB クラスターの変更。 AWS CLI を使用して、`modify-db-cluster` を実行し、`--network-type` オプションを設定します。 RDS API を使用して、`ModifyDBCluster` を呼び出し、`NetworkType` パラメータを設定します。	DB クラスター全体	この変更時に機能停止は発生しません。

設定と説明	方法	スコープ	ダウンタイムに関する注意
ブネットが最低 2 つ必要です。また、IPv6 CIDR ブロックを、指定した DB サブネットグループのサブネットに関連付けてください。 詳細については、「Amazon Aurora IP アドレス指定」を参照してください。			
新しいマスターパスワード マスターユーザーのパスワード。 • Aurora MySQL の場合、パスワードには 8 ~ 41 個の印刷可能な ASCII 文字を使用する必要があります。 • Aurora PostgreSQL の場合は、8 ~ 99 個の印刷可能な ASCII 文字を使用する必要があります。 • /、"、@、またはスペースは使用できません。	AWS Management Console の使用、DB クラスター内の DB インスタンスの変更。 AWS CLI を使用して、modify-db-cluster を実行し、--master-user-password オプションを設定します。 RDS API を使用して、ModifyDBCluster を呼び出し、MasterUserPassword パラメータを設定します。	DB クラスター全体	この変更時に機能停止は発生しません。

設定と説明	方法	スコープ	ダウンタイムに関する注意
Performance Insights DB インスタンスの負荷をモニタリングするツールである、Performance Insights を有効にするかどうかによって、データベースパフォーマンスの分析およびトラブルシューティングができます。 詳細については、「Amazon Aurora での Performance Insights を使用したDB 負荷のモニタリング」を参照してください。	AWS Management Console の使用、DB クラスター内の DB インスタンスの変更。 AWS CLI を使用して、modify-db-instance を実行し、--enable-performance-insights|--no-enable-performance-insights オプションを設定します。 RDS API を使用して、ModifyDBInstance を呼び出し、EnablePerformanceInsights パラメータを設定します。	指定された DB インスタンスのみ	この変更時に機能停止は発生しません。

設定と説明	方法	スコープ	ダウンタイムに関する注意
Performance Insights AWS KMS key Performance Insights データを暗号化するための AWS KMS key キー識別子。KMS キー識別子は、KMS キーの Amazon リソースネーム (ARN)、キー識別子、またはキーエイリアスです。 詳細については、「Performance Insights の有効化と無効化」を参照してください。	AWS Management Console の使用、DB クラスター内の DB インスタンスの変更。 AWS CLI を使用して、`modify-db-instance` を実行し、`--performance-insights-kms-key-id` オプションを設定します。 RDS API を使用して、`ModifyDBInstance` を呼び出し、`PerformanceInsightsKMSKeyId` パラメータを設定します。	指定された DB インスタンスのみ	この変更時に機能停止は発生しません。

設定と説明	方法	スコープ	ダウンタイムに関する注意
Performance Insights の保持期間 Performance Insights データを保持する期間 (日数)。無料利用枠の保持設定は「デフォルト (7 日)」です。パフォーマンスデータをさらに長期間保持するには、1 ~ 24 か月を指定します。保持期間の詳細については、「Performance Insights の料金とデータ保持」を参照してください。 詳細については、「Performance Insights の有効化と無効化」を参照してください。	AWS Management Console の使用、DB クラスター内の DB インスタンスの変更。 AWS CLI を使用して、`modify-db-instance` を実行し、`--performance-insights-retention-period` オプションを設定します。 RDS API を使用して、`ModifyDBInstance` を呼び出し、`PerformanceInsightsRetentionPeriod` パラメータを設定します。	指定された DB インスタンスのみ	この変更時に機能停止は発生しません。

設定と説明	方法	スコープ	ダウンタイムに関する注意
昇格階層 既存のプライマリインスタンスの障害後に、Aurora レプリカが DB クラスター内のプライマリインスタンスに昇格される順序を指定する値。 詳細については、「Aurora DB クラスターの耐障害性」を参照してください。	AWS Management Console の使用、DB クラスター内の DB インスタンスの変更。 AWS CLI を使用して、`modify-db-instance` を実行し、`--promotion-tier` オプションを設定します。 RDS API を使用して、`ModifyDBInstance` を呼び出し、`PromotionTier` パラメータを設定します。	指定された DB インスタンスのみ	この変更時に機能停止は発生しません。

設定と説明	方法	スコープ	ダウンタイムに関する注意
パブリックアクセス パブリック IP アドレスを DB インスタンスに付与する場合は [パブリックアクセス可能] を選択します。これにより、DB インスタンスは VPC 外からアクセス可能になります。パブリックにアクセス可能となるよう、DB インスタンスは、VPC のパブリックサブネット内にある必要があります。 VPC 内からのみ DB インスタンスにアクセス可能にするには、[パブリックアクセス不可] を選択します。 詳細については、「VPC 内の DB クラスターをインターネットから隠す」を参照してください。 Amazon VPC の外部から DB インスタンスに接続するには、DB インスタンスがパブ	AWS Management Console の使用、DB クラスター内の DB インスタンスの変更。 AWS CLI を使用して、`modify-db-instance` を実行し、`--publicly-accessible|--no-publicly-accessible` オプションを設定します。 RDS API を使用して、`ModifyDBInstance` を呼び出し、`PubliclyAccessible` パラメータを設定します。	指定された DB インスタンスのみ	この変更時に機能停止は発生しません。

設定と説明	方法	スコープ	ダウンタイムに関する注意
リックにアクセス可能であり、アクセスが DB インスタンスのセキュリティグループのインバウンドルールで許可されているなど、いくつかの要件を満たす必要があります。詳細については、「Amazon RDS DB インスタンスに接続できない」を参照してください。 DB インスタンスがパブリックアクセス可能でない場合は、AWS Site-to-Site VPN 接続または AWS Direct Connect 接続を使用してプライベートネットワークからアクセスすることもできます。詳細については、「インターネットトラフィックのプライバシー」を参照してください。			

設定と説明	方法	スコープ	ダウンタイムに関する注意
Serverless v2 容量設定 Aurora キャパシティユニット (ACU) で測定される Aurora Serverless v2 DB クラスターのデータベース容量。 詳細については、「Aurora Serverless v2 クラスターの容量設定」を参照してください。	AWS Management Console の使用、コンソール、CLI、API を使用した DB クラスターの変更。 AWS CLI を使用して、`modify-db-cluster` を実行し、`--serverless-v2-scaling-configuration` オプションを設定します。 RDS API を使用して、`ModifyDBCluster` を呼び出し、`ServerlessV2ScalingConfiguration` パラメータを設定します。	DB クラスター全体	この変更時に機能停止は発生しません。 変更はただちに発生します。この設定は、[Apply immediately (すぐに適用)] 設定を無視します。

設定と説明	方法	スコープ	ダウンタイムに関する注意
セキュリティグループ DB クラスターに関連付けるセキュリティグループ。 詳細については、「セキュリティグループによるアクセス制御」を参照してください。	AWS Management Console の使用、コンソール、CLI、API を使用した DB クラスターの変更。 AWS CLI を使用して、`modify-db-cluster` を実行し、`--vpc-security-group-ids` オプションを設定します。 RDS API を使用して、`ModifyDBCluster` を呼び出し、`VpcSecurityGroupIds` パラメータを設定します。	DB クラスター全体	この変更時に機能停止は発生しません。

設定と説明	方法	スコープ	ダウンタイムに関する注意
ターゲットバックトラックウィンドウ DB クラスターをバックトラックできる時間 (秒)。この設定は、Aurora MySQL にのみ、およびバックトラックを有効にして DB クラスターが作成された場合にのみ使用できます。	AWS Management Console の使用、コンソール、CLI、API を使用した DB クラスターの変更。 AWS CLI を使用して、`modify-db-cluster` を実行し、`--backtrack-window` オプションを設定します。 RDS API を使用して、`ModifyDBCluster` を呼び出し、`BacktrackWindow` パラメータを設定します。	DB クラスター全体	この変更時に機能停止は発生しません。

Amazon Aurora DB クラスターには適用されない設定

AWS CLI コマンドの `modify-db-cluster`、および RDS API オペレーションの `ModifyDBCluster` に関する以下の設定は、Amazon Aurora DB クラスターには適用されません。

> **Note**
>
> AWS Management Console を使用して Aurora DB クラスターのこれらの設定を変更することはできません。

AWS CLI の設定	RDS API の設定
`--allocated-storage`	`AllocatedStorage`
`--auto-minor-version-upgrade | --no-auto-minor-version-upgrade`	`AutoMinorVersionUpgrade`
`--db-cluster-instance-class`	`DBClusterInstanceClass`
`--enable-performance-insights | --no-enable-performance-insights`	`EnablePerformanceInsights`
`--iops`	`Iops`
`--monitoring-interval`	`MonitoringInterval`
`--monitoring-role-arn`	`MonitoringRoleArn`
`--option-group-name`	`OptionGroupName`
`--performance-insights-kms-key-id`	`PerformanceInsightsKMSKeyId`
`--performance-insights-retention-period`	`PerformanceInsightsRetentionPeriod`

Amazon Aurora DB インスタンスには適用されない設定

AWS CLI コマンドの `modify-db-instance`、および RDS API オペレーションの `ModifyDBInstance` に関する以下の設定は、Amazon Aurora DB インスタンスには適用されません。

> **Note**
>
> AWS Management Console を使用して Aurora DB インスタンスのこれらの設定を変更することはできません。

AWS CLI の設定	RDS API の設定
--allocated-storage	AllocatedStorage
--allow-major-version-upgrade|--no-allow-major-version-upgrade	AllowMajorVersionUpgrade
--copy-tags-to-snapshot|--no-copy-tags-to-snapshot	CopyTagsToSnapshot
--domain	Domain
--db-security-groups	DBSecurityGroups
--db-subnet-group-name	DBSubnetGroupName
--domain-iam-role-name	DomainIAMRoleName
--multi-az|--no-multi-az	MultiAZ
--iops	Iops
--license-model	LicenseModel
--network-type	NetworkType
--option-group-name	OptionGroupName
--processor-features	ProcessorFeatures
--storage-type	StorageType
--tde-credential-arn	TdeCredentialArn
--tde-credential-password	TdeCredentialPassword
--use-default-processor-features|--no-use-default-processor-features	UseDefaultProcessorFeatures

DB クラスターに Aurora レプリカを追加する

レプリケーションがある Aurora DB クラスターには、1 つのプライマリ DB インスタンスと最大 15 の Aurora レプリカがあります。プライマリ DB インスタンスは読み書きオペレーションをサポートし、クラスターボリュームに対するすべてのデータ変更を実行します。Aurora レプリカは、プライマリ DB インスタンスと同じストレージボリュームに接続しますが、読み取りオペレーションのみサポートしています。Aurora レプリカを使用して、プライマリ DB インスタンスから読み取りワークロードをオフロードします。詳細については、「Aurora レプリカ」を参照してください。

Amazon Aurora レプリカには以下の制限事項があります。

- Aurora Serverless v1 DB クラスターの Aurora レプリカを作成することはできません。Aurora Serverless v1 には、データベースのすべての読み取りおよび書き込みオペレーションをサポートするために、自動的にスケールアップおよびスケールダウンする単一の DB インスタンスがあります。

  ただし、リーダーインスタンスを Aurora Serverless v2 DB クラスターに追加することはできます。詳細については、「Aurora Serverless v2 リーダーの追加」を参照してください。

Aurora DB クラスターのプライマリインスタンスと Aurora レプリカを複数のアベイラビリティーゾーンに分散させて、DB クラスターの可用性を改善することをお勧めします。詳細については、「利用可能なリージョン」を参照してください。

Aurora レプリカを Aurora DB クラスターから削除するには、「Aurora DB クラスターからの DB インスタンスの削除」の手順に従って、Aurora レプリカ DB インスタンスを削除します。

> **Note**
>
> Amazon Aurora では、RDS DB インスタンスなどの外部データベースとのレプリケーションもサポートします。RDS DB インスタンスは、Amazon Aurora と同じ AWS リージョンにある必要があります。詳細については、「Amazon Aurora でのレプリケーション」を参照してください。

Aurora レプリカを DB クラスターに追加するには AWS Management Console、AWS CLI、または RDS API を使用します。

コンソール

Aurora レプリカを DB クラスターに追加するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択し、新しい DB インスタンスを追加する DB クラスターを選択します。
3. クラスターとプライマリインスタンスの両方が [使用可能] 状態であることを確認します。DB クラスターまたはプライマリインスタンスが [作成中] などの移行状態にある場合、レプリカを追加することはできません。

   クラスターにプライマリインスタンスがない場合は、create-db-instance AWS CLI コマンドを使用してプライマリインスタンスを作成します。この状況は、CLI を使用して DB クラスタースナップショットを復元し、AWS Management Console でクラスターを表示した場合に発生する可能性があります。
4. [アクション] で、[リーダーの追加] を選択します。

   [リーダーの追加] ページが表示されます。
5. [リーダーの追加] ページで、Aurora レプリカのオプションを指定します。次の表は、Aurora レプリカの設定を示しています。

使用するオプション	操作
アベイラビリティーゾーン	特定のアベイラビリティーゾーンを指定するかどうかを指定します。リストには、DB クラスター作成時にセキュリティした DB サブネットグループにマッピングされたアベイラビリティーゾーンのみが含まれます。利用可能ゾーンについての詳細は、「リージョンとアベイラビリティーゾーン」を参照してください。
パブリックアクセス可能	Aurora レプリカにパブリック IP アドレスを割り当てるには [Yes] を選択します。それ以外の場合は [No] を選択します。Aurora レプリカをパブリックアクセスから隠す方法については、「VPC 内の DB クラスターをインターネットから隠す」を参照してください。

使用するオプション	操作
暗号化	この Aurora レプリカを保管時に暗号化できるように、[Enable encryption] を選択します。詳細については、「Amazon Aurora リソースの暗号化」を参照してください。
DB インスタンスクラス	Aurora レプリカの処理要件やメモリ要件を定義する DB インスタンスクラスを選択します。DB インスタンスクラスのオプションについては、「Aurora DB インスタンスクラス」を参照してください。
Aurora レプリカの出典	Aurora レプリカを作成するプライマリインスタンスの識別子を選択します。
DB インスタンス識別子	選択した AWS リージョン内で、アカウントにとって一意となるインスタンス名を入力します。名前には、選択した AWS リージョンと DB エンジンなどを含めると理解しやすくなります (**aurora-read-instance1** など)。
優先度	インスタンスのフェイルオーバー優先度を選択します。値を選択しない場合、デフォルト値は tier-1 になります。この優先度により、プライマリインスタンスの障害からの復旧時に、Aurora レプリカを昇格する順序が決まります。詳細については、「Aurora DB クラスターの耐障害性」を参照してください。
データベースポート	Aurora レプリカのポートは、DB クラスターのポートと同じです。
DB パラメータグループ	パラメータグループを選択します。Aurora にはデフォルトのパラメータグループが用意されています。また、独自のパラメータグループを作成することもできます。パラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。

使用するオプション	操作
Performance Insights	[Performance Insights を有効にする] チェックボックスはデフォルトで選択されています。値はライターインスタンスから継承されることはありません。詳細については、「Amazon Aurora での Performance Insights を使用したDB 負荷のモニタリング」を参照してください。
拡張モニタリング	DB クラスターが実行されているオペレーティングシステムに対してリアルタイムでのメトリクスの収集を有効にするには、[拡張モニタリングを有効にする] を選択します。詳細については、「拡張モニタリングを使用した OS メトリクスのモニタリング」を参照してください。
モニタリングロール	[拡張モニタリング] が [拡張モニタリングの有効化] に設定されている場合にのみ使用できます。Amazon CloudWatch Logs との通信を Amazon RDS に許可するために作成した IAM ロールを選択するか、[デフォルト] を選択して、RDS によって `rds-monitoring-role` という名前のロールが作成されるようにします。詳細については、「拡張モニタリングを使用した OS メトリクスのモニタリング」を参照してください。
詳細度	[拡張モニタリング] が [拡張モニタリングの有効化] に設定されている場合にのみ使用できます。DB クラスターのメトリクスを収集する間隔を秒単位で設定します。

使用するオプション	操作
マイナーバージョン自動アップグレード	Aurora DB クラスターに DB エンジンのマイナーバージョンアップグレードがリリースと同時に自動的に適用されるようにする場合は、[Enable auto minor version upgrade (マイナーバージョン自動アップグレードの有効化)] を選択します。 [マイナーバージョン自動アップグレード] の設定は Aurora PostgreSQL および Aurora MySQL DB クラスターの両方に適用されます。Aurora MySQL 2.x クラスターの場合、この設定により、クラスターは最大でバージョン 2.07.2 にアップグレードされます。 Aurora PostgreSQL のエンジンに関する更新の詳細については、「Amazon Aurora PostgreSQL の更新」を参照してください。 Aurora MySQL のエンジンに関する更新の詳細については、「Amazon Aurora MySQL のデータベースエンジンの更新」を参照してください。

6. [リーダーの追加] を選択して、Aurora レプリカを作成します。

AWS CLI

DB クラスターに Aurora レプリカを作成するには、AWS CLI コマンドの create-db-instance を実行します。`--db-cluster-identifier` オプションとして DB クラスターの名前を含めます。次の例に示すように、オプションで `--availability-zone` パラメータを使用して Aurora レプリカのアベイラビリティーゾーンを指定できます。

例えば、次のコマンドは `sample-instance-us-west-2a` という名前の新しい MySQL 5.7 互換 Aurora レプリカを作成します。

Linux、macOS、Unix の場合:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
 db.r5.large \
```

```
    --availability-zone us-west-2a
```

Windows の場合:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
 db.r5.large ^
    --availability-zone us-west-2a
```

次のコマンドは、sample-instance-us-west-2a という名前の新しい MySQL 5.7 互換 Aurora レプリカを作成します。

Linux、macOS、Unix の場合:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
 db.r5.large \
    --availability-zone us-west-2a
```

Windows の場合:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora --db-instance-class
 db.r5.large ^
    --availability-zone us-west-2a
```

次のコマンドは sample-instance-us-west-2a という名前の新しい PostgreSQL 互換 Aurora レプリカを作成します。

Linux、macOS、Unix の場合:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-instance-
class db.r5.large \
    --availability-zone us-west-2a
```

Windows の場合:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-instance-
class db.r5.large ^
```

```
    --availability-zone us-west-2a
```

RDS API

DB クラスターに Aurora レプリカを作成するには、CreateDBInstance オペレーションを呼び出します。DBClusterIdentifier パラメータとして DB クラスターの名前を含めます。オプションで、AvailabilityZone パラメータを使用して Aurora レプリカのアベイラビリティーゾーンを指定できます。

Aurora DB クラスターのパフォーマンスとスケーリングの管理

次のオプションを使用して、Aurora DB クラスターおよび DB インスタンスのパフォーマンスおよびスケーリングを管理できます。

トピック

ストレージのスケーリング

Aurora ストレージは、クラスターボリューム内のデータに合わせて自動的にスケーリングします。データが増大するに従って、クラスターボリュームストレージは最大 128 tebibytes (TiB) または 64 TiB まで拡張されます。最大サイズは、DB エンジンのバージョンによって異なります。クラスターボリュームに含まれるデータの種類については、「Amazon Aurora ストレージと信頼性」を参照してください。特定のバージョンの最大サイズは、「Amazon Aurora サイズ制限」でご確認ください。

クラスターボリュームのサイズは 1 時間ごとに評価され、ストレージコストが決定されます。料金情報については、Aurora の料金表ページを参照してください。

Aurora クラスターボリュームのサイズは多数のテビバイトまでスケールアップできますが、課金対象となるのはそのボリュームの使用した領域分のみです。請求対象のストレージ領域を決定する方法は、Aurora クラスターのバージョンによって異なります。

- クラスターボリュームから Aurora データを削除すると、相応して請求対象の領域が全体的に減少します。この動的なサイズ変更動作は、基礎となるテーブルスペースの削除や再編成に伴って、必要な領域が減った場合に発生します。したがって、不要になったテーブルとデータベースを削除することで、ストレージ料金を削減できます。動的サイズ変更は、特定の Aurora バージョンに適用されます。データを削除するとクラスターボリュームが動的にサイズ変更される Aurora バージョンは次のとおりです。

Aurora MySQL	• バージョン 3 (MySQL 8.0 互換): すべてのサポート対象バージョン • バージョン 2 (MySQL 5.7 互換): 2.11 以上
Aurora PostgreSQL	サポートされている全バージョン
Aurora Serverless v2	サポートされている全バージョン
Aurora Serverless v1	サポートされている全バージョン

- 上記のバージョンよりも低い Aurora バージョンでは、データを削除したときに解放されたスペースをクラスターボリュームで再利用できますが、ボリューム自体のサイズが小さくなることはありません。
- この機能は、Aurora が利用可能な AWS リージョンに段階的にデプロイされています。クラスターを使用するリージョンによっては、この機能はまだ利用できない場合があります。

動的サイズ変更は、クラスターボリューム内のテーブルスペースを物理的に削除またはサイズ変更するオペレーションに適用されます。つまり、`DROP TABLE`、`DROP DATABASE`、`TRUNCATE TABLE`、および `ALTER TABLE ... DROP PARTITION` などの SQL ステートメントに対し適用されます。`DELETE` ステートメントを使用した行の削除には適用されません。テーブルから多数の行を削除する場合は、Aurora MySQL `OPTIMIZE TABLE` ステートメントを実行するか、後で Aurora PostgreSQL `pg_repack` エクステンションを使用して、テーブルを再編成し、クラスターボリュームのサイズを動的に変更できます。

> **Note**
>
> Aurora MySQL の場合、`innodb_file_per_table` パラメータがテーブルストレージの編成方法に影響します。テーブルがシステムテーブルスペースの一部である場合、テーブルを削除してもシステムテーブルスペースのサイズは縮小されません。したがって、動的サイズ変更を最大限に活用するには、Aurora MySQL クラスターで必ず `innodb_file_per_table` を 1 に設定してください。
> Aurora MySQL バージョン 2.11 以降では、InnoDB 一時テーブルスペースは再起動時に削除され、再作成されます。これにより、一時テーブルスペースが占めていたスペースがシステムに解放され、クラスターボリュームのサイズが変更されます。動的サイズ変更機能を最大限に活用するには、DB クラスターを Aurora MySQL バージョン 2.11 以降にアップグレードすることをお勧めします。

動的サイズ変更機能は、テーブルスペース内のテーブルが削除されてもすぐにスペースを再利用しませんが、1 日あたり約 10 TB のレートで徐々に増加します。システムテーブルスペースは削除されないため、システムテーブルスペース内のスペースは再利用されません。テーブルスペース内の再利用されていないスペースは、操作によってそのテーブルスペースにスペースが必要になったときに再利用されます。動的サイズ変更機能では、クラスターが使用可能な状態にある場合にのみ、ストレージスペースを再利用できます。

クラスターで使用されているストレージ容量を確認するには、CloudWatch の VolumeBytesUsed メトリクスをモニタリングします。ストレージ料金の詳細については、「Aurora データストレージに対する請求方法」を参照してください。

- AWS Management Console の場合、クラスターの詳細ページで Monitoring タブを表示することで、この容量をグラフで確認できます。
- AWS CLI の場合、次の Linux の例のようなコマンドを実行できます。スタート時刻、終了時刻、クラスター名は、独自の値に置き換えます。

```
aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '6 hours ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" \
  --statistics Average Maximum Minimum \
  --dimensions Name=DBClusterIdentifier,Value=my_cluster_identifier
```

このコマンドでは、次のような出力が生成されます。

```
{
    "Label": "VolumeBytesUsed",
    "Datapoints": [
        {
            "Timestamp": "2020-08-04T21:25:00+00:00",
            "Average": 182871982080.0,
            "Minimum": 182871982080.0,
            "Maximum": 182871982080.0,
            "Unit": "Bytes"
        }
    ]
}
```

次の例では、Linux システムで AWS CLI コマンドを使用することにより、Aurora クラスターのストレージ使用状況を経時的に追跡する方法を示しています。--start-time パラメータと --end-time パラメータは、全体の時間間隔を 1 日として定義します。--period パラメータは、1 時間間隔で測定値をリクエストします。メトリクスは継続的にではなく、間隔を置いて収集されるため、--period として小さい値を選択しても意味がありません。また、Aurora ストレージオペレーションは、関連する SQL ステートメントが終了した後で、バックグラウンドでしばらく続行することがあります。

初期の例では、デフォルトの JSON 形式で出力が返されます。データポイントは、タイムスタンプでソートされず、任意の順で返されます。この JSON データをチャートツールにインポートして、並べ替えや視覚化を行うことができます。

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '1 day ago')" --end-time "$(date -d 'now')" --period 3600
  --namespace "AWS/RDS" --statistics Maximum --dimensions
 Name=DBClusterIdentifier,Value=my_cluster_id
{
    "Label": "VolumeBytesUsed",
    "Datapoints": [
        {
            "Timestamp": "2020-08-04T19:40:00+00:00",
            "Maximum": 182872522752.0,
            "Unit": "Bytes"
        },
        {
            "Timestamp": "2020-08-05T00:40:00+00:00",
            "Maximum": 198573719552.0,
            "Unit": "Bytes"
        },
        {
            "Timestamp": "2020-08-05T05:40:00+00:00",
            "Maximum": 206827454464.0,
            "Unit": "Bytes"
        },
        {
            "Timestamp": "2020-08-04T17:40:00+00:00",
            "Maximum": 182872522752.0,
            "Unit": "Bytes"
        },
... output omitted ...
```

次の例は、前と同じデータを返します。--output パラメータは、コンパクトなプレーンテキスト形式でデータを表します。aws cloudwatch コマンドは、その出力を sort コマンドにパイプします。-k コマンドの sort パラメータは、3 番目のフィールドで出力をソートします。これは、UTC (世界協定時刻) 形式のタイムスタンプです。

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '1 day ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Maximum --dimensions
 Name=DBClusterIdentifier,Value=my_cluster_id \
  --output text | sort -k 3
VolumeBytesUsed
DATAPOINTS  182872522752.0  2020-08-04T17:41:00+00:00 Bytes
DATAPOINTS  182872522752.0  2020-08-04T18:41:00+00:00 Bytes
DATAPOINTS  182872522752.0  2020-08-04T19:41:00+00:00 Bytes
DATAPOINTS  182872522752.0  2020-08-04T20:41:00+00:00 Bytes
DATAPOINTS  187667791872.0  2020-08-04T21:41:00+00:00 Bytes
DATAPOINTS  190981029888.0  2020-08-04T22:41:00+00:00 Bytes
DATAPOINTS  195587244032.0  2020-08-04T23:41:00+00:00 Bytes
DATAPOINTS  201048915968.0  2020-08-05T00:41:00+00:00 Bytes
DATAPOINTS  205368492032.0  2020-08-05T01:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T02:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T03:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T04:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T05:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T06:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T07:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T08:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T09:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T10:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T11:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T12:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T13:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T14:41:00+00:00 Bytes
DATAPOINTS  206833664000.0  2020-08-05T15:41:00+00:00 Bytes
DATAPOINTS  206833664000.0  2020-08-05T16:41:00+00:00 Bytes
```

ソートされた出力は、モニタリング期間のスタート時と終了時のストレージの使用量を示します。また、該当期間中に Aurora でより多くのストレージがクラスターに割り当てられたポイントも確認できます。次の例では、Linux コマンドを使用して、スタート時と終了時の VolumeBytesUsed 値をギガバイト (GB) およびギビバイト (GiB) として再フォーマットします。ギガバイトは 10 の累乗で測定された単位を表し、回転式ハードドライブのストレージに関する説明でよく使用されます。ギビ

バイトは、2 の累乗で測定された単位を表します。Aurora ストレージでは通常、測定値と制限を、ギビバイトやテビバイトなどの 2 の累乗単位を使用して表記します。

```
$ GiB=$((1024*1024*1024))
$ GB=$((1000*1000*1000))
$ echo "Start: $((182872522752/$GiB)) GiB, End: $((206833664000/$GiB)) GiB"
Start: 170 GiB, End: 192 GiB
$ echo "Start: $((182872522752/$GB)) GB, End: $((206833664000/$GB)) GB"
Start: 182 GB, End: 206 GB
```

VolumeBytesUsed メトリクスは、料金を発生させているクラスター内のストレージの量を示します。したがって、この数値はできるだけ最小限に抑えるようにします。ただし、このメトリクスには、Aurora がクラスターで内部的に使用している、課金対象外のストレージは含まれません。クラスターがストレージ制限に近づき、容量が不足する可能性がある場合は、AuroraVolumeBytesLeftTotal メトリクスをモニタリングし、その数値を最大限に増やすようにします。次の例では、前と同様の計算を実行しますが、AuroraVolumeBytesLeftTotal ではなく、VolumeBytesUsed が対象です。

```
$ aws cloudwatch get-metric-statistics --metric-name "AuroraVolumeBytesLeftTotal" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Maximum --dimensions
 Name=DBClusterIdentifier,Value=my_old_cluster_id \
  --output text | sort -k 3
AuroraVolumeBytesLeftTotal
DATAPOINTS      140530528288768.0       2023-02-23T19:25:00+00:00       Count
$ TiB=$((1024*1024*1024*1024))
$ TB=$((1000*1000*1000*1000))
$ echo "$((69797067915264 / $TB)) TB remaining for this cluster"
69 TB remaining for this cluster
$ echo "$((69797067915264 / $TiB)) TiB remaining for this cluster"
63 TiB remaining for this cluster
```

Aurora MySQL バージョン 2.09 以降または Aurora PostgreSQL を実行しているクラスターの場合、VolumeBytesUsed によって報告される空きサイズは、データが追加されると増加し、データが削除されると減少します。以下の例のように指定します。このレポートには、テンポラリデータを含むテーブルの作成や削除に伴う、クラスターの最大と最小のストレージサイズが 15 分間隔で表示されます。レポートには、最小値の前に最大値が表示されます。したがって、15 分間隔内のストレージ使用量の変動を確認するには、数値を右から左へと解釈します。

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
```

```
  --start-time "$(date -d '4 hours ago')" --end-time "$(date -d 'now')" --period 1800 \
  --namespace "AWS/RDS" --statistics Maximum Minimum --dimensions
 Name=DBClusterIdentifier,Value=my_new_cluster_id
  --output text | sort -k 4
VolumeBytesUsed
DATAPOINTS 14545305600.0 14545305600.0 2020-08-05T20:49:00+00:00 Bytes
DATAPOINTS 14545305600.0 14545305600.0 2020-08-05T21:19:00+00:00 Bytes
DATAPOINTS 22022176768.0 14545305600.0 2020-08-05T21:49:00+00:00 Bytes
DATAPOINTS 22022176768.0 22022176768.0 2020-08-05T22:19:00+00:00 Bytes
DATAPOINTS 22022176768.0 22022176768.0 2020-08-05T22:49:00+00:00 Bytes
DATAPOINTS 22022176768.0 15614263296.0 2020-08-05T23:19:00+00:00 Bytes
DATAPOINTS 15614263296.0 15614263296.0 2020-08-05T23:49:00+00:00 Bytes
DATAPOINTS 15614263296.0 15614263296.0 2020-08-06T00:19:00+00:00 Bytes
```

次の例は、Aurora MySQL バージョン 2.09 以降または Aurora PostgreSQL を実行しているクラスターで、AuroraVolumeBytesLeftTotal によって報告される空きサイズに 128 TiB のサイズ制限がどのように反映されているかを示しています。

```
$ aws cloudwatch get-metric-statistics --region us-east-1 --metric-name
 "AuroraVolumeBytesLeftTotal" \
  --start-time "$(date -d '4 hours ago')" --end-time "$(date -d 'now')" --period 1800 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBClusterIdentifier,Value=pq-57 \
  --output text | sort -k 3
AuroraVolumeBytesLeftTotal
DATAPOINTS 140515818864640.0 2020-08-05T20:56:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-05T21:26:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-05T21:56:00+00:00 Count
DATAPOINTS 140514866757632.0 2020-08-05T22:26:00+00:00 Count
DATAPOINTS 140511020580864.0 2020-08-05T22:56:00+00:00 Count
DATAPOINTS 140503168843776.0 2020-08-05T23:26:00+00:00 Count
DATAPOINTS 140503168843776.0 2020-08-05T23:56:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-06T00:26:00+00:00 Count
$ TiB=$((1024*1024*1024*1024))
$ TB=$((1000*1000*1000*1000))
$ echo "$((140515818864640 / $TB)) TB remaining for this cluster"
140 TB remaining for this cluster
$ echo "$((140515818864640 / $TiB)) TiB remaining for this cluster"
127 TiB remaining for this cluster
```

インスタンスのスケーリング

DB クラスター内の各 DB インスタンスの DB インスタンスクラスを変更することで、必要に応じて Aurora DB クラスターをスケーリングできます。Aurora は、データベースエンジンの互換性に応じて Aurora 用に最適化された、複数の DB インスタンスクラスをサポートしています。

データベースエンジン	インスタンスのスケーリング
Amazon Aurora MySQL	「Aurora MySQL DB インスタンスのスケーリング」を参照してください。
Amazon Aurora PostgreSQL	「Aurora PostgreSQL DB インスタンスのスケーリング」を参照してください。

読み取りのスケーリング

Aurora DB クラスターの読み取りのスケーリングは、DB クラスターに最大 15 個の Aurora レプリカを作成することで実現できます。各 Aurora レプリカは、最小限のレプリカラグでクラスターボリュームから同じデータを返します。通常、このラグはプライマリインスタンスが更新を書き込んだ後、100 ミリ秒を大幅に下回ります。読み取りトラフィックが増えたら、追加の Aurora レプリカを作成し、それらに直接接続することで DB クラスターの読み取りワークロードを分散できます。Aurora レプリカの DB インスタンスクラスは、プライマリイスタンスと同じものである必要はありません。

DB クラスターに Aurora レプリカを追加する方法については、「DB クラスターに Aurora レプリカを追加する」を参照してください。

接続の管理

Aurora DB インスタンスへの許可されている接続の最大数は、DB インスタンスのインスタンスレベルパラメータグループの `max_connections` パラメータによって決まります。そのパラメータのデフォルト値は、DB インスタンスおよびデータベースエンジンの互換性に使用される DB インスタンスクラスによって異なります。

データベースエンジン	max_connections のデフォルト値
Amazon Aurora MySQL	「Aurora MySQL DB インスタンスへの最大接続数」を参照してください。
Amazon Aurora PostgreSQL	「Aurora PostgreSQL DB インスタンスへの最大接続数」を参照してください。

> **Tip**
>
> アプリケーションが頻繁に接続を開いたり閉じたりする場合や、長時間の接続を多数開いたままにする場合は、Amazon RDS Proxy の使用を推奨します。RDS Proxy は、接続プーリングを使用してデータベース接続を安全かつ効率的に共有する、フルマネージドの高可用性データベースプロキシです。RDS Proxy の詳細については、Amazon RDS Proxy for Aurora の使用 を参照してください。

クエリ実行計画の管理

Aurora PostgreSQL のクエリプラン管理を使用すると、オプティマイザーが実行する計画を制御できます。詳細については、「Aurora PostgreSQL のクエリ実行計画の管理」を参照してください。

Amazon Aurora DB クラスターのボリュームのクローン作成

Aurora クローン作成を使用すると、元のクラスターと同じデータページを共有するものの、個別の独立したボリュームを持つ新しいクラスターを作成できます。このプロセスは、高速で、費用効果が高いように設計されています。関連付けられたデータボリュームを持つ新しいクラスターは、クローンと呼ばれます。クローンの作成は、スナップショットの復元など、他の手法を使用してデータを物理的にコピーするよりも、高速かつスペース効率に優れています。

トピック

Aurora クローン作成の概要

Aurora では、クローン作成に、コピーオンライトプロトコルが使用されます。このメカニズムでは、初期クローンを作成するために使用する追加領域は最小限です。クローンが初期に作成されると、Aurora は、ソース Aurora DB クラスターと新しい (クローンの) Aurora DB クラスターで使用されるデータのコピーを 1 つだけ保持します。追加のストレージは、ソース Aurora DB クラスターまたは Aurora DB クラスターのクローンが (Aurora ストレージボリューム上の) データに変更を加えた場合にのみ割り当てられます。コピーオンライトプロトコルの詳細については、「Aurora クローン作成の仕組み」を参照してください。

Aurora のクローン作成は、データを破損の危険にさらすことなく、本番データを使用してテスト環境を迅速にセットアップする場合に特に役立ちます。クローンは、次のようなさまざまなタイプのアプリケーションに使用できます。

- 潜在的な変更 (スキーマの変更やパラメータグループの変更など) を試して、すべての影響を評価する。
- データのエクスポートや分析クエリの実行など、大量のワークロードを扱うオペレーションをクローン上で実行する。
- 開発、テスト、またはその他の目的のために、本番 DB クラスターのコピーを作成する。

同じ Aurora DB クラスターから複数のクローンを作成できます。また、別のクローンから複数のクローンを作成することもできます。

Aurora クローンを作成したら、その Aurora DB インスタンスを、ソース Aurora DB クラスターとは異なる設定にできます。例えば、開発用途のクローンは、本番環境のソース Aurora DB クラスターと同じ高可用性要件を満たす必要がない場合があります。この場合、Aurora DB クラスターで使用される複数の DB インスタンスではなく、単一の Aurora DB インスタンスを使用するようにクローンを設定できます。

作成元とは異なるデプロイ設定を使用してクローンを作成すると、作成元の Aurora DB エンジンの最新のマイナーバージョンを使用してクローンが作成されます。

Aurora DB クラスターからクローンを作成すると、クローンは作成者の AWS アカウント (ソース Aurora DB クラスターを所有しているアカウントと同じアカウント) に作成されます。ただし、Aurora Serverless v2 とプロビジョニングされた Aurora DB クラスターおよびクローンを他の AWS アカウントと共有することもできます。詳細については、「AWS RAM および Amazon Aurora を使用したクロスアカウントのクローン作成」を参照してください。

テスト、開発などの用途へのクローンの使用が終了したら、クローンを削除できます。

Aurora クローン作成の制限

Aurora のクローン作成には、現在、次の制約事項があります。

- AWS リージョン で許可される DB クラスターの最大数まで、必要な数のクローンを作成できます。

  コピーオンライトプロトコルまたはフルコピープロトコルを使用してクローンを作成できます。フルコピープロトコルは、ポイントインタイムリカバリのように機能します。
- ソース Aurora DB クラスターとは異なる AWS リージョンにはクローンを作成できません。
- パラレルクエリ機能なしの Aurora DB クラスターから、パラレルクエリを使用するクラスターにクローンを作成することはできません。並行クエリを使用するクラスターにデータを格納するには、元のクラスターのスナップショットを作成して、並行クエリ機能を使用するクラスターにそれを復元します。
- DB インスタンスを持たない Aurora DB クラスターからクローンを作成することはできません。少なくとも 1 つの DB インスタンスを持つ Aurora DB クラスターのクローン作成のみが可能です。
- クローンは、Aurora DB クラスターとは異なる仮想プライベートクラウド (VPC) で作成できます。その場合、VPCのサブネットは同じアベイラビリティーゾーンにマッピングする必要があります。

- プロビジョニングされた Aurora DB クラスターから、Aurora プロビジョニングされたクローンを作成できます。
- Aurora Serverless v2 インスタンスを持つクラスターは、プロビジョンドクラスターと同じルールに従います。
- Aurora Serverless v1 の場合:
  - Aurora Serverless v1 DB クラスターからプロビジョニングされたクローンを作成できます。
  - Aurora Serverless v1 または プロビジョニングされた DB クラスターから Aurora Serverless v1 クローンを作成できます。
  - 暗号化されていないプロビジョニングされた Aurora DB クラスターからは Aurora Serverless v1 クローンを作成できません。
  - 現在、Aurora Serverless v1 DB クラスターのクローン作成では、他のアカウントへのクローン作成はサポートされていません。詳細については、「クロスアカウントのクローン作成の制約事項」を参照してください。
  - クローンの Aurora Serverless v1 DB クラスターの動作と制限は、Aurora Serverless v1 DB クラスターと同じです。詳細については、「Amazon Aurora Serverless v1 の使用」を参照してください。
  - Aurora Serverless v1 DB クラスターは常に暗号化されます。クローンを作成するとき Aurora Serverless v1 DB クラスターをプロビジョニングされた Aurora DB クラスターに変換すると、プロビジョニングされた Aurora DB クラスターは暗号化されます。暗号化キーは選択できますが、暗号化を無効にすることはできません。プロビジョニングされた Aurora DB クラスターから Aurora Serverless v1 にクローンを作成するには、まず暗号化プロビジョニングされた Aurora DB クラスターから開始する必要があります。

Aurora クローン作成の仕組み

Aurora クローン作成は Aurora DB クラスターのストレージレイヤーで動作します。コピーオンライトプロトコルが使用されます。これは、Aurora ストレージボリュームをサポートする基盤となる耐久性の高いメディアという点で、高速かつスペース効率に優れています。Aurora クラスターボリュームの詳細については、「Amazon Aurora ストレージの概要」を参照してください。

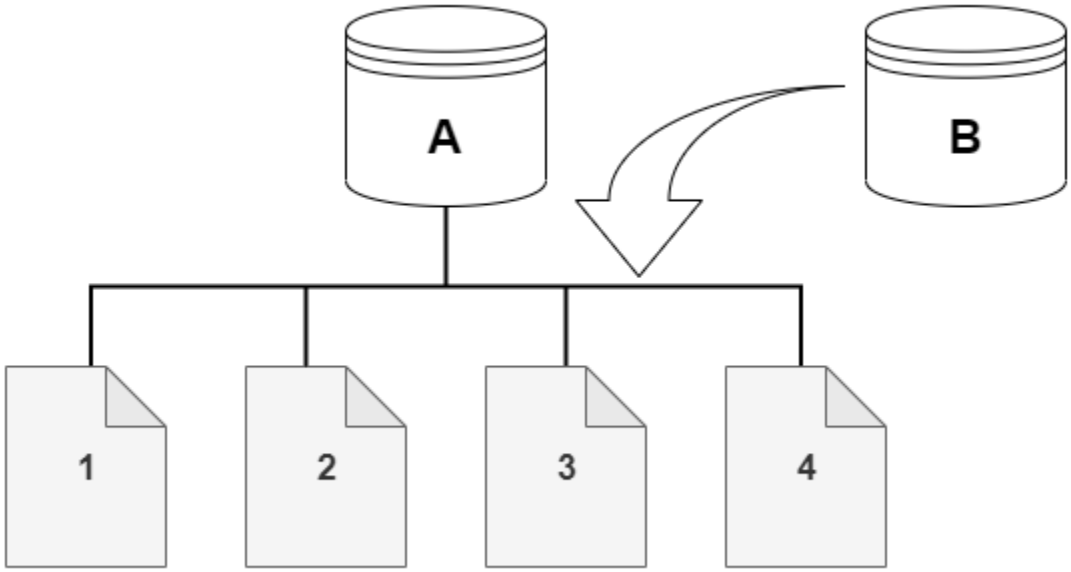
トピック

- コピーオンライトプロトコルの理解
- ソースクラスターボリュームの削除

コピーオンライトプロトコルの理解

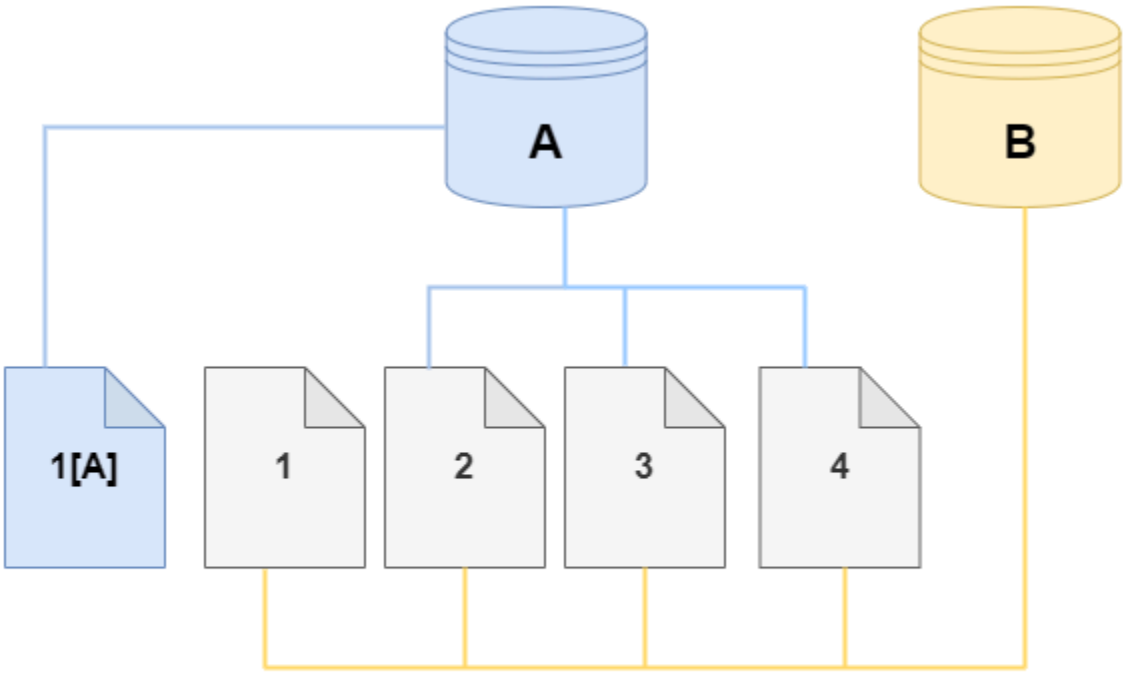
Aurora DB クラスターでは、基になる Aurora ストレージボリュームのページにデータが格納されます。

例えば、次の図には、4 つのデータページ 1、2、3、4 を持つ Aurora DB クラスター (A) があります。クローン B が Aurora DB クラスターから作成されたとします。クローンが作成されても、データはコピーされません。クローンは、ソース Aurora DB クラスターと同じページのセットを参照しています。

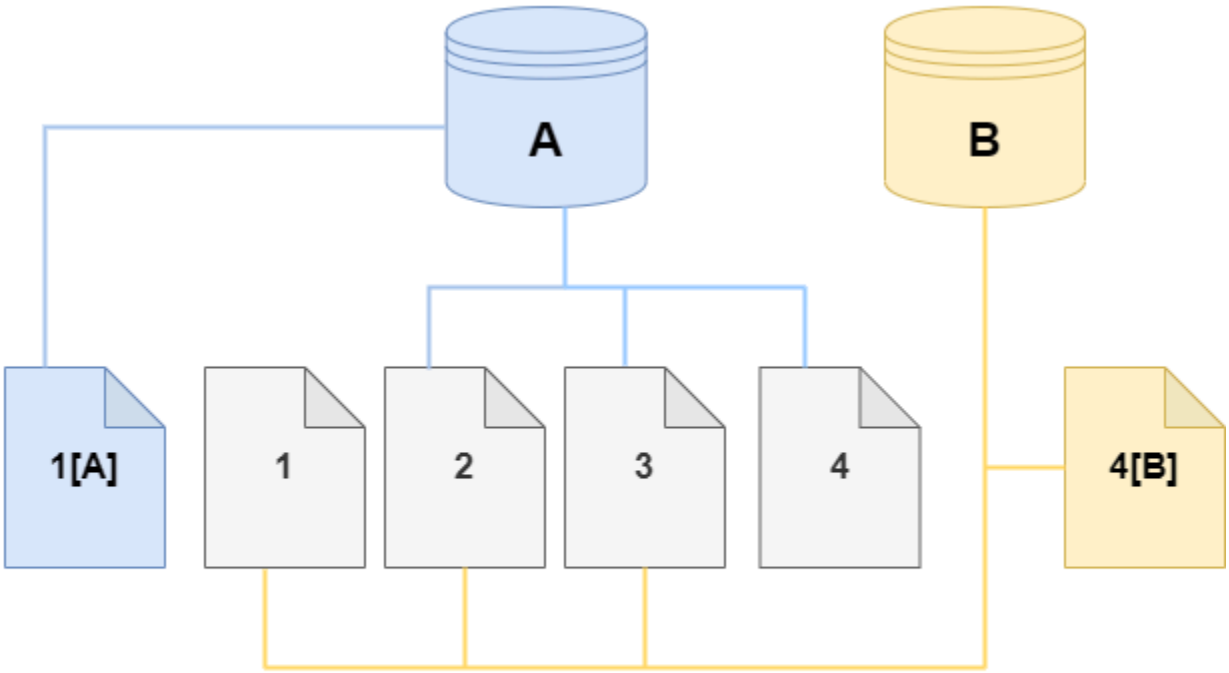


クローンが作成されたとき、通常は追加のストレージは必要ありません。コピーオンライトプロトコルでは、ソースセグメントと同じ物理ストレージメディア上のセグメントを使用します。追加のストレージが必要になるのは、ソースセグメントの容量がクローンセグメント全体に対して十分でない場合のみです。この場合、ソースセグメントは別の物理デバイスにコピーされます。

次の図に、前述と同様にクラスター A とそのクローン B を使用して動作中のコピーオンライトプロトコルの例を示します。Aurora DB クラスター (A) に変更を加えて、ページ 1 に保持されているデータが変更されたとします。元のページ 1 に書き込む代わりに、Aurora は新しいページ 1[A] を作成します。クラスター (A) の Aurora DB クラスターボリュームは、1[A]、2、3、4 ページを参照していますが、クローン (B) は引き続き元のページを参照しています。



クローンでは、ストレージボリュームのページ 4 に変更が加えられています。元のページ 4 に書き込む代わりに、Aurora は新しいページ 4[B] を作成します。クローンはページ 1、2、3、およびページ 4[B] を参照し、クラスター (A) は引き続き 1[A]、2、3、4 を参照しています。



時間が経過してソース Aurora DB クラスターボリュームとクローンの両方で追加の変更があると、その変更をキャプチャして保存するためにさらにストレージが必要になります。

ソースクラスターボリュームの削除

当初、クローンボリュームは、クローンの作成元のボリュームと同じデータページを共有します。元のボリュームが存在する限り、クローンボリュームはクローンが作成または変更したページの所有者と見なされるのみです。したがって、クローンボリュームの `VolumeBytesUsed` メトリクスは最初は小さく、元のクラスターとクローンの間でデータが分岐するに従って増えていきます。ソースボリュームとクローン間でページが同じである場合、ストレージ料金は元のクラスターにのみ適用されます。`VolumeBytesUsed` メトリクスの詳細については、「Amazon Aurora のクラスターレベルのメトリクス」を参照してください。

1 つ以上のクローンが関連付けられているソースクラスターボリュームを削除しても、クローンのクラスターボリューム内のデータは変更されません。Aurora は、ソースクラスターボリュームが以前に所有していたページを保持します。Aurora は、削除したクラスターが所有していたページのストレージ料金を再分配します。例えば、元のクラスターに 2 つのクローンがあり、後で元のクラス

ターを削除したとします。元のクラスターが所有していたデータページの半分は、1 つのクローンが所有することになります。残りの半分のページは、もう 1 つのクローンが所有します。

元のクラスターを削除してクローンを作成または削除すると、Aurora は同じページを共有するすべてのクローン間で、データページの所有権を引き続き再配分します。したがって、クローンのクラスターボリュームに応じてメトリクスの値が変わることがわかります。より多くのクローンを作成して、ページの所有権がより多くのクラスターに分散されると、メトリクスの値は減少する可能性があります。また、クローンを削除して、ページの所有権を割り当てるクラスターの数が少なくなると、メトリクスの値は増える可能性があります。書き込みオペレーションがクローンボリュームのデータページにどのように影響するかについては、「コピーオンライトプロトコルの理解」を参照してください。

元のクラスターとクローンを同じ AWS アカウントが所有している場合、これらのクラスターのすべてのストレージ料金は同じ AWS アカウントに適用されます。一部のクラスターがクロスアカウントクローンである場合、元のクラスターを削除すると、クロスアカウントクローンを所有する AWS アカウントに追加のストレージ料金がかかる可能性があります。

例えば、クローンを作成する前に、クラスターボリュームに 1,000 件の使用済みデータページがあるとします。このクラスターのクローンを作成すると、当初のクローンボリュームの使用済みページはゼロです。クローンが 100 件のデータページに変更を加えると、その 100 ページだけがクローンボリュームに保存され、使用済みとしてマークされます。親ボリュームから変更されていない残りの 900 ページは、両方のクラスターで共有されます。この例で、親クラスターの場合は 1,000 ページ、クローンボリュームの場合は 100 ページに対してストレージ料金が発生します。

ソースボリュームを削除した場合、クローンのストレージ料金は、変更した 100 ページと元のボリュームの 900 の共有ページを含めて、合計 1,000 ページ分となります。

Amazon Aurora クローンの作成

ソース Aurora DB クラスターと同じ AWS アカウントにクローンを作成できます。そうするには、AWS Management Console または AWS CLI を使用して、その後の手順を行ってください。

別の AWS アカウントにクローン作成を許可したり、別の AWS アカウントとクローンを共有したりするには、AWS RAM および Amazon Aurora を使用したクロスアカウントのクローン作成 の手順を使用します。

コンソール

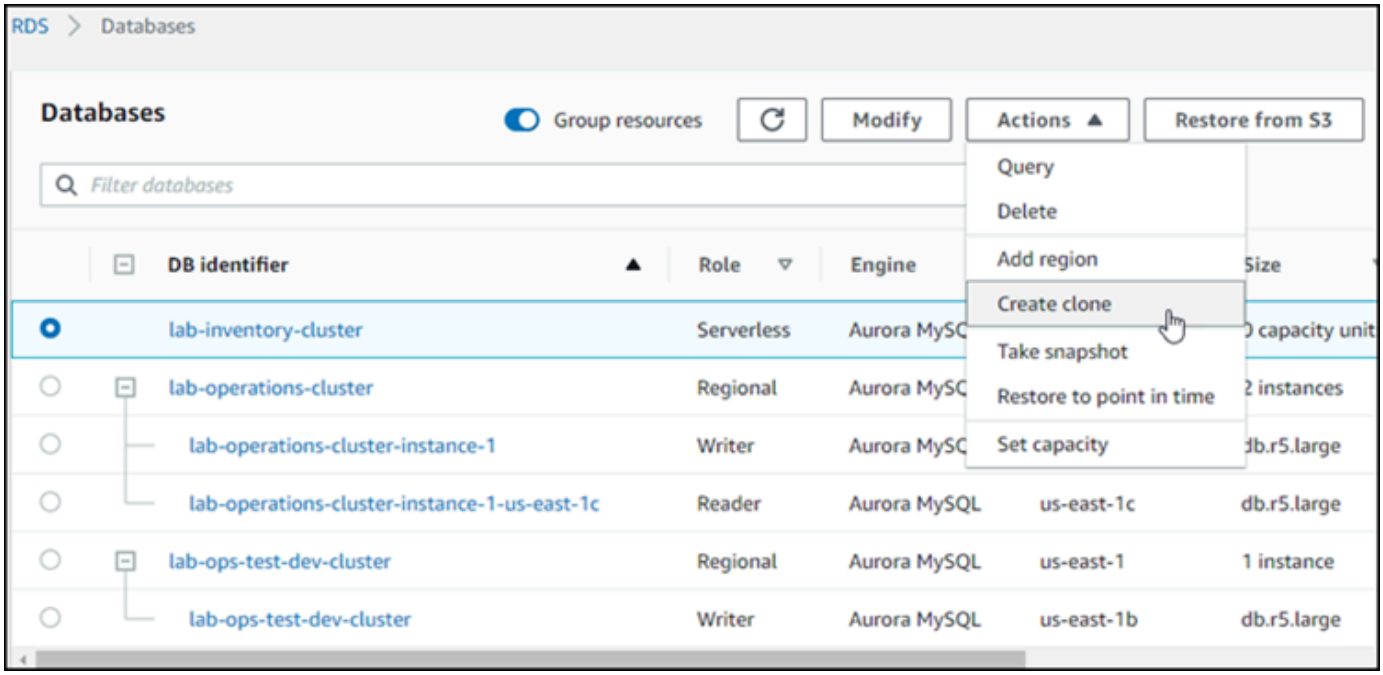
AWS Management Console を使用して Aurora DB クラスターのクローンを作成する手順を以下に示します。

AWS Management Console を使用してクローンを作成すると、1 つの Aurora DB インスタンスを持つ Aurora DB クラスターができます。

これらの手順は、クローンを作成しているのと同じ AWS アカウントが所有する DB クラスターに適用されます。別の AWS アカウントが所有する DB クラスターの場合は、AWS RAM および Amazon Aurora を使用したクロスアカウントのクローン作成 を参照してください。

AWS を使用して、お客様の AWS Management Console アカウントが所有している DB クラスターのクローンを作成するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. リストから Aurora DB クラスターを選択し、[アクション] で、[クローンの作成] を選択します。



[クローンの作成] ページが開きます。そこで、[設定]、[接続] などの Aurora DB クラスタークローンのオプションが設定できます。

4. [DB インスタンス識別子] に、作成する Aurora DB クラスターのクローンに付ける名前を入力します。
5. Aurora Serverless v1 DB クラスターの場合は、[キャパシティータイプ] に[プロビジョニング済み] または [サーバーレス] を選択します。

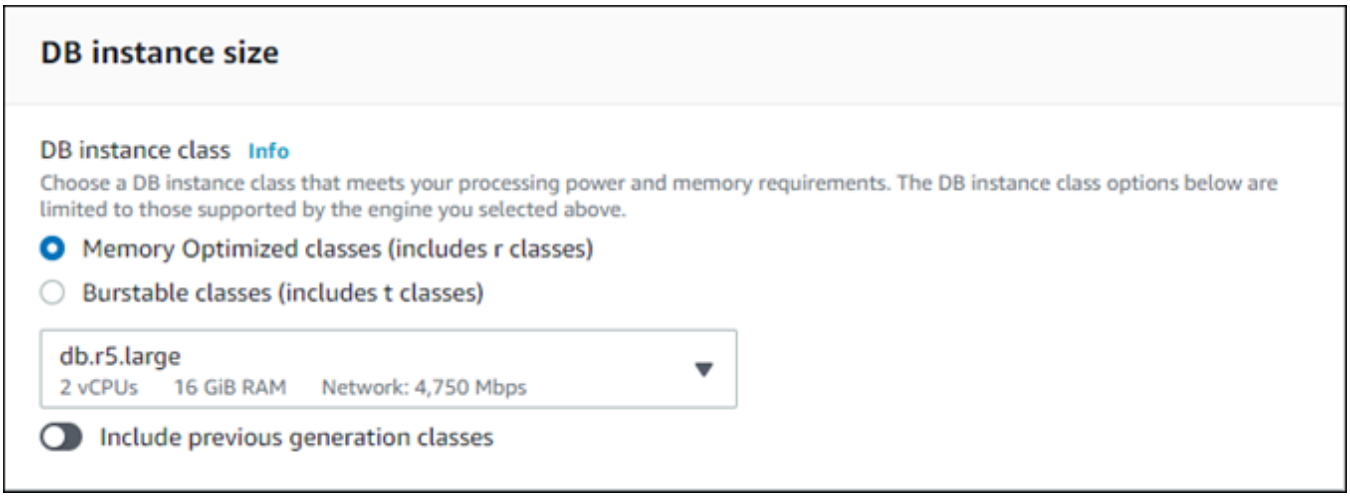
ソース Aurora DB クラスターが Aurora Serverless v1 DB クラスターまたは暗号化されているプロビジョニングされた Aurora DB クラスターであるときのみ、[サーバーレス] を選択することができます。

6. Aurora Serverless v2 またはプロビジョニングされた DB クラスターの場合は、[クラスターストレージ設定] に Aurora Standard または Aurora I/O-Optimized を選択します。

   詳細については、「Amazon Aurora DB クラスターのストレージ設定」を参照してください。

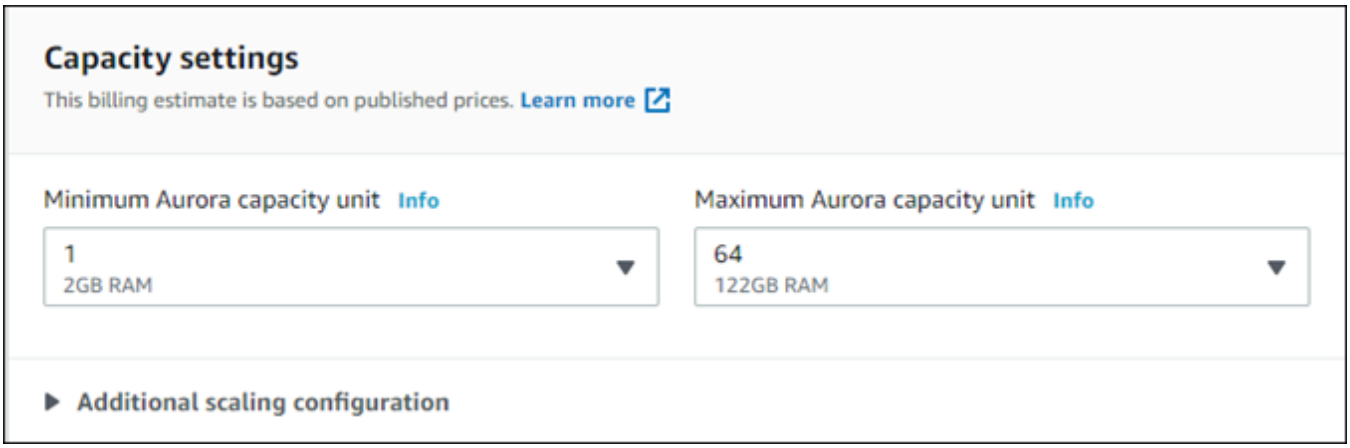
7. DB インスタンスのサイズまたは DB クラスターの容量を選択してください。

   - プロビジョニングされたクローンの場合は、DB インスタンスクラスを選択します。



   用意されている設定のままにすることも、クローンに別の DB インスタンスクラスを使用することもできます。

   - Aurora Serverless v1 または Aurora Serverless v2 クローンの場合は、[容量設定] を選択します。



用意されている設定のままにすることも、クローンに合わせて変更することもできます。

8. クローンに必要な他の設定を選択します。Aurora DB クラスターとインスタンスの設定の詳細については、「Amazon Aurora DB クラスターの作成」を参照してください。
9. [クローンの作成] を選択します。

クローンが作成されると、コンソールの [データベース] セクションに他の Aurora DB クラスターとともに一覧表示され、現在の状態が表示されます。状態が [使用可能] の場合は、クローンはすぐに使用できます。

AWS CLI

AWS CLI を使用して Aurora DB クラスターのクローンを作成するには、いくつかのステップが必要です。

`restore-db-cluster-to-point-in-time` AWS CLI コマンドを使用すると、Aurora DB インスタンスの個数が 0 の空の Aurora DB クラスターができます。つまり、このコマンドは Aurora DB クラスターのみを復元し、クラスターの DB インスタンスは復元しません。これは、クローンが使用可能になった後に別途行います。プロセスは 2 ステップで、次のとおりです。

1. restore-db-cluster-to-point-in-time CLI コマンドを使用して、クローンを作成します。このコマンドで使用するパラメータで、作成する空の Aurora DB クラスター (クローン) の容量タイプなどの詳細が制御されます。
2. create-db-instance CLI コマンドを使用して、クローン用の Aurora DB インスタンスを作成し、復元された Aurora DB クラスターに Aurora DB インスタンスを再作成します。

トピック

- クローンの作成
- ステータスの確認とクローンの詳細の取得
- クローン用の Aurora DB インスタンスの作成
- クローン作成に使用するパラメータ

クローンの作成

`restore-db-cluster-to-point-in-time` CLI コマンドに渡す特定のパラメータは、さまざまです。渡す内容は、ソース DB クラスターのエンジンモードのタイプ (Serverless またはプロビジョニング) および作成するクローンのタイプによって異なります。

ソース Aurora DB クラスターと同じエンジンモードのクローンを作成するには

- restore-db-cluster-to-point-in-time CLI コマンドを使用して、次のパラメータに値を指定します。
  - `--db-cluster-identifier` - クローン用の意味のある名前を選択します。restore-db-cluster-to-point-in-time CLI コマンド使用時に、クローンに名前を付けます。次に、create-db-instance CLI コマンドで、クローンの名前を渡します。
  - `--restore-type` - ソース DB クラスターのクローンの作成に `copy-on-write` を使用します。このパラメータを指定しない場合、`restore-db-cluster-to-point-in-time` は、クローンを作成するのではなく、Aurora DB クラスターを復元します。
  - `--source-db-cluster-identifier` - クローンを作成するソース Aurora DB クラスターの名前を使用します。
  - `--use-latest-restorable-time` - この値は、ソース DB クラスターの最新の復元可能なボリュームデータを指します。これを使用してクローンを作成します。

次の例では、`my-source-cluster` という名前のクラスターから `my-clone` という名前のクローンを作成します。

Linux、macOS、Unix の場合:

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-source-cluster \
    --db-cluster-identifier my-clone \
    --restore-type copy-on-write \
    --use-latest-restorable-time
```

Windows の場合:

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-identifier my-source-cluster ^
    --db-cluster-identifier my-clone ^
    --restore-type copy-on-write ^
    --use-latest-restorable-time
```

このコマンドは、クローンの詳細を含む JSON オブジェクトを返します。クローンの DB インスタンスを作成する前に、作成した DB クラスターのクローンが使用可能であることを確認します。詳細については、「ステータスの確認とクローンの詳細の取得」を参照してください。

ソース Aurora DB クラスターとは異なるエンジンモードでクローンを作成するには

- `restore-db-cluster-to-point-in-time` CLI コマンドを使用して、次のパラメータに値を指定します。
  - `--db-cluster-identifier` - クローン用の意味のある名前を選択します。restore-db-cluster-to-point-in-time CLI コマンド使用時に、クローンに名前を付けます。次に、create-db-instance CLI コマンドで、クローンの名前を渡します。
  - `--source-db-cluster-identifier` - クローンを作成するソース Aurora DB クラスターの名前を使用します。
  - `--restore-type` - ソース DB クラスターのクローンの作成に `copy-on-write` を使用します。このパラメータを指定しない場合、`restore-db-cluster-to-point-in-time` は、クローンを作成するのではなく、Aurora DB クラスターを復元します。
  - `--use-latest-restorable-time` - この値は、ソース DB クラスターの最新の復元可能なボリュームデータを指します。これを使用してクローンを作成します。
  - `--engine-mode` - (オプション) このパラメータは、ソース Aurora DB クラスターとは異なるタイプのクローンを作成する場合にのみ使用します。次のように `--engine-mode` として渡す値を選択します。
    - Aurora Serverless DB クラスターからプロビジョニングされた Aurora DB クラスターのクローンを作成するには、`provisioned` を使用します。
    - プロビジョニングされた Aurora DB クラスターから Aurora Serverless v1 DB クラスターのクローンを作成するには、`serverless` を使用します。`serverless` エンジンモードを指定すると、`--scaling-configuration` を選択することもできます。
  - `--scaling-configuration` - (オプション) `--engine-mode serverless` の場合に使用して、Aurora Serverless v1 クローンの最小と最大の容量を設定します。このパラメータを使用しない場合、Aurora は DB エンジンのデフォルトの容量値を使用してクローンを作成します。
  - `--serverless-v2-scaling-configuration` - (オプション) このパラメータを使用して、Aurora Serverless v2 クローンの最小と最大の容量を設定します。このパラメータを使用しない場合、Aurora は DB エンジンのデフォルトの容量値を使用してクローンを作成します。

次の例では、`my-source-cluster` という名前のプロビジョニングされた Aurora DB クラスターから `my-clone` という名前の Aurora Serverless v1 クローンを作成します。プロビジョニングされた Aurora DB クラスターは暗号化されています。

Linux、macOS、Unix の場合:

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-source-cluster \
    --db-cluster-identifier my-clone \
    --engine-mode serverless \
    --scaling-configuration MinCapacity=8,MaxCapacity=64 \
    --restore-type copy-on-write \
    --use-latest-restorable-time
```

Windows の場合:

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-identifier my-source-cluster ^
    --db-cluster-identifier my-clone ^
    --engine-mode serverless ^
    --scaling-configuration MinCapacity=8,MaxCapacity=64 ^
    --restore-type copy-on-write ^
    --use-latest-restorable-time
```

これらのコマンドは、DB インスタンスの作成に必要なクローンの詳細を含む JSON オブジェクトを返します。クローン (空の Aurora DB クラスター) のステータスが [使用可能] になるまでこれを実行できません。

> **Note**
>
> restore-db-cluster-to-point-in-time AWS CLI コマンドは、その DB クラスターの DB インスタンスではなく、DB クラスターのみを復元します。復元された DB クラスターの DB インスタンスを作成するには、create-db-instance コマンドを呼び出すときに、復元する DB クラスターの識別子を `--db-cluster-identifier` に指定する必要があります。`restore-db-cluster-to-point-in-time` コマンドが完了し、DB クラスターが使用可能になった後でのみ、DB インスタンスを作成できます。

例えば、これからクローンを作成しようとしている `tpch100g` という名前のクラスターがあるとします。次の Linux の例では、`tpch100g-clone` という名前のクローンクラスターと、新しいクラスター用の `tpch100g-clone-instance` という名前のプライマリインスタンスを作成します。`--master-username` や `--master-user-password` など、いくつかのパラメータは指定する必要がありません。Aurora では、元のクラスターからそれらが自動的に決定されます。使用する DB エ

ンジンについては、指定が必要です。この例では、新しいクラスターをテストして、--engine パラメータに使用する適切な値を判断します。

```
$ aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier tpch100g \
  --db-cluster-identifier tpch100g-clone \
  --restore-type copy-on-write \
  --use-latest-restorable-time

$ aws rds describe-db-clusters \
  --db-cluster-identifier tpch100g-clone \
    --query '*[].[Engine]' \
    --output text
aurora-mysql

$ aws rds create-db-instance \
  --db-instance-identifier tpch100g-clone-instance \
  --db-cluster-identifier tpch100g-clone \
  --db-instance-class db.r5.4xlarge \
  --engine aurora-mysql
```

ステータスの確認とクローンの詳細の取得

次のコマンドを使用して、新しく作成した空の DB クラスターのステータスが確認できます。

```
$ aws rds describe-db-clusters --db-cluster-identifier my-clone --query '*[].[Status]'
 --output text
```

または、次の AWS CLI クエリを使用して、ステータスなどの必要な値を取得して、クローン用の DB インスタンスを作成できます。

Linux、macOS、Unix の場合:

```
aws rds describe-db-clusters --db-cluster-identifier my-clone \
  --query '*[].
{Status:Status,Engine:Engine,EngineVersion:EngineVersion,EngineMode:EngineMode}'
```

Windows の場合:

```
aws rds describe-db-clusters --db-cluster-identifier my-clone ^
```

```
  --query "*[].
{Status:Status,Engine:Engine,EngineVersion:EngineVersion,EngineMode:EngineMode}"
```

このクエリにより、以下のような出力が返されます。

```
[
  {
        "Status": "available",
        "Engine": "aurora-mysql",
        "EngineVersion": "8.0.mysql_aurora.3.04.1",
        "EngineMode": "provisioned"
    }
]
```

クローン用の Aurora DB インスタンスの作成

Aurora Serverless v2 またはプロビジョニングされたクローン用の DB インスタンスを作成するには、create-db-instance CLI コマンドを使用します。Aurora Serverless v1 クローン用の DB インスタンスは作成しません。

DB インスタンスは、`--master-username` および `--master-user-password` プロパティを、ソース DB クラスターから引き継ぎます。

次の例では、プロビジョニングされたクローンの DB インスタンスを作成します。

Linux、macOS、Unix の場合:

```
aws rds create-db-instance \
    --db-instance-identifier my-new-db \
    --db-cluster-identifier my-clone \
    --db-instance-class db.r5.4xlarge \
    --engine aurora-mysql
```

Windows の場合:

```
aws rds create-db-instance ^
    --db-instance-identifier my-new-db ^
    --db-cluster-identifier my-clone ^
    --db-instance-class db.r5.4xlarge ^
    --engine aurora-mysql
```

クローン作成に使用するパラメータ

次の表は `restore-db-cluster-to-point-in-time` を使用して Aurora DB クラスターのクローンを作成する際に使用されるさまざまなパラメータをまとめたものです。

Parameter	説明
`--source-db-cluster-identifier`	クローンを作成するソース Aurora DB クラスターの名前を使用します。
`--db-cluster-identifier`	`restore-db-cluster-to-point-in-time` コマンドを使用してクローンを作成するときに、わかりやすい名前を選択します。次に、この名前を `create-db-instance` コマンドに渡します。
`--restore-type`	`--restore-type` として `copy-on-write` を指定すると、ソース Aurora DB クラスターを復元するのではなく、ソース DB クラスターのクローンを作成します。
`--use-latest-restorable-time`	この値は、ソース DB クラスターの最新の復元可能なボリュームデータを指します。これを使用してクローンを作成します。
`--engine-mode`	このパラメータを使用して、以下のいずれかの値でソース Aurora DB クラスターとは異なるタイプのクローンを作成します。 • Aurora Serverless v1 DB クラスターからプロビジョニングされた、または Aurora Serverless v2 クローンを作成するには、`provisioned` を使用します。 • プロビジョニングされた、または Aurora Serverless v2 DB クラスターから Aurora Serverless v1 クローンを作成するには、`serverless` を使用します。 `serverless` エンジンモードを指定すると、`--scaling-configuration` を選択することもできます。
`--scaling-configuration`	このパラメータを使用して、Aurora Serverless v1 クローンの最小と最大の容量を設定します。このパラメータを指定しない場合、Aurora は DB エンジンのデフォルトの容量値を使用してクローンを作成します。

Parameter	説明
`--serverless-v2-scaling-configuration`	このパラメータを使用して、Aurora Serverless v2 クローンの最小と最大の容量を設定します。このパラメータを指定しない場合、Aurora は DB エンジンのデフォルトの容量値を使用してクローンを作成します。

AWS RAM および Amazon Aurora を使用したクロスアカウントのクローン作成

Amazon Aurora とともに AWS Resource Access Manager (AWS RAM) を使用して、自分の AWS アカウントにある Aurora DB クラスターとクローンを別の AWS アカウントまたは組織と共有できます。このようなクロスアカウントのクローン作成は、データベーススナップショットを作成および復元するよりもはるかに高速です。Aurora DB クラスターの 1 つのクローンを作成し、そのクローンを共有できます。また、Aurora DB クラスターを別の AWS アカウントと共有し、そのアカウント所有者にクローンを作成させることもできます。選択するアプローチは、ユースケースによって異なります。

例えば、財務データベースのクローンを組織の内部監査チームと定期的に共有する必要がある場合があります。この場合、監査チームには使用しているアプリケーション用の独自の AWS アカウントがあります。監査チームの AWS アカウントに、Aurora DB クラスターにアクセスし、必要に応じてクローンを作成するアクセス許可を付与できます。

一方、外部ベンダーが財務データを監査する場合は、自分でクローンを作成することをお勧めします。次に、外部ベンダーにクローンのみへのアクセス権を付与します。

また、クロスアカウントのクローン作成を使用して、開発やテストなど、同じ AWS アカウントでクローンを作成する多くの同じユースケースをサポートできます。例えば、組織では、本番、開発、テストなどに、別々の AWS アカウントを使用していることがあります。詳細については、「Aurora クローン作成の概要」を参照してください。

したがって、クローンを別の AWS アカウントと共有したり、別の AWS アカウントで Aurora DB クラスターのクローンを作成したりできるようにする必要があることがあります。どちらの場合も、まず AWS RAM を使用して、共有オブジェクトを作成します。AWS アカウント間での AWS リソースの共有の詳細については、「AWS RAM ユーザーガイド」を参照してください。

クロスアカウントのクローンを作成するには、元のクラスターを所有する AWS アカウントと、クローンを作成する AWS アカウントによるアクションが必要です。まず、元のクラスターの所有者が

他の 1 つ以上のアカウントによるクローンの作成を許可するようにクラスターを変更します。アカウントのいずれかが別の AWS 組織に属している場合は、AWS は、共有の招待を生成します。先に進む前に、もう一方のアカウントは招待を受け入れる必要があります。招待を受け入れた、承認済みのアカウントはそれぞれ、クラスターのクローンを作成することができます。このプロセスを通じて、クラスターは一意の Amazon リソースネーム (ARN) で識別されます。

同じAWSアカウント内のクローン作成と同様に、追加のストレージ領域が使用されるのは、ソースまたはクローンによってデータに変更が加えられた場合のみです。ストレージ料金は、その時点で適用されます。ソースクラスターが削除された場合、ストレージコストは残りのクローンクラスター間で均等に分散されます。

トピック

- クロスアカウントのクローン作成の制約事項
- 他の AWS アカウントによるクラスターのクローン作成を許可する
- 別の AWS アカウントが所有するクラスターのクローンを作成する

クロスアカウントのクローン作成の制約事項

Aurora クロスアカウントのクローン作成には、次の制約事項があります。

- Aurora Serverless v1 アカウント間で AWS クラスターを複製することはできません
- AWS Management Console で、共有リソースへの招待を表示したり受け入れたりすることはできません。AWS CLI、Amazon RDS API、または AWS RAM コンソールを使用して、共有リソースへの招待を表示し受け入れます。
- 自分の AWS アカウントと共有されているクローンから新しいクローンを作成することはできます。
- 自分の AWS アカウントと共有されているリソース (クローンまたは Aurora DB クラスター) を共有することはできません。
- 1 つの Aurora DB クラスターから最大 15 のクロスアカウントクローンを作成することができます。
- 15 のクローンは、それぞれ異なる AWS アカウントが所有する必要があります。つまり、クラスターのクロスアカウントクローンは AWS アカウントに 1 つしか作成できません。
- クラスターのクローンを作成した後、クロスアカウントクローンに制限を適用する目的で、元のクラスターとそのクローンは同じと見なされます。元のクラスターとクローンされたクラスターの両方のクロスアカウントクローンを同じ AWS アカウント内に作成することはできません。元のクラスターとそのクローンのクロスアカウントクローンの合計数は 15 を超えることはできません。

- Aurora DB クラスターは、そのクラスターが ACTIVE 状態でなければ、他の AWS アカウントと共有できません。
- 他の AWS アカウントと共有されている Aurora DB クラスターの名前を変更することはできません。
- デフォルトの RDS キーで暗号化されているクラスターのクロスアカウントクローンを作成することはできません。
- ある AWS アカウントと共有されている暗号化された Aurora DB クラスターから、暗号化されていないクローンを別の AWS アカウントに作成することはできません。クラスター所有者は、ソースクラスターの AWS KMS key にアクセスするアクセス許可を付与する必要があります。ただし、クローンを作成するときに別のキーを使用できます。

他の AWS アカウントによるクラスターのクローン作成を許可する

他の AWS アカウントで、所有しているクラスターのクローンを作成できるようにするには、AWS RAM を使用して共有のアクセス許可を設定します。そのようにすることで、異なる AWS 組織の他の各アカウントに招待が送信されます。

所有しているリソースを AWS RAM コンソールで共有する手順については、AWS RAM ユーザーガイドの「お客様が所有しているリソースの共有」を参照してください。

トピック

- クラスターのクローンを作成するためのアクセス許可を他の AWS アカウントに付与する
- 所有しているクラスターが他の AWS アカウントと共有されているかどうかを確認する

クラスターのクローンを作成するためのアクセス許可を他の AWS アカウントに付与する

共有しているクラスターが暗号化されている場合は、そのクラスターの AWS KMS key も共有します。ある AWS アカウントの AWS Identity and Access Management (IAM) ユーザーまたはロールに、別のアカウントの KMS キーの使用を許可できます。

これを行うには、まず AWS KMS を使用して、外部アカウント (ルートユーザー) を KMS キーのキーポリシーに追加します。個別のユーザーまたはロールをキーポリシーに追加するのではなく、それらを所有する外部アカウントのみを追加します。作成する KMS キーのみ共有することができます。デフォルトの RDS サービスキーを共有することはできません。KMS キーのアクセス制御については、「AWS KMS に対する認証とアクセス制御」を参照してください。

コンソール

クラスターのクローンを作成するためのアクセス許可を付与するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. [詳細] ページを表示するために共有する DB クラスターを選択後、[接続とセキュリティ] タブを選択します。
4. 「DB クラスターを他のAWSアカウントと共有する」セクションで、このクラスターの複製を許可するAWSアカウントの数値アカウントIDを入力します。同じ組織のアカウント ID の場合は、ボックスに入力をスタートし、メニューから選択することができます。

   > **Important**
   >
   > 場合によっては、アカウントと同じ AWS 組織にないアカウントで、クラスターのクローンを作成することがあります。このような場合は、セキュリティ上の理由により、コンソールで、アカウント ID を所有しているユーザーや、アカウントの有無はレポートされません。
   > AWS アカウントと同じ AWS 組織にないアカウントの数値は慎重に入力してください。目的のアカウントと共有されたことをすぐに確認します。

5. 確認ページで、指定したアカウント ID が正しいことを確認します。確認するには、確認ボックスに「`share`」と入力します。

   [詳細] ページの [この DB クラスターが共有されているアカウント] に、指定した AWS アカウント ID を示すエントリが表示されます。[ステータス] 列には初期、[Pending (保留中)] のステータスが表示されます。
6. 他の AWS アカウントの所有者に連絡するか、両方を所有している場合はそのアカウントにサインインしてください。以下の説明に従って、共有の招待を受け入れ、DB クラスターのクローンを作成するように他のアカウントの所有者に指示します。

AWS CLI

クラスターのクローンを作成するためのアクセス許可を付与するには

1. 必須パラメータに関する情報を収集します。クラスターの ARN と他の AWS アカウントの数値 ID が必要です。
2. AWS RAM CLI コマンド `create-resource-share` を実行します。

   Linux、macOS、Unix の場合:

   ```
   aws ram create-resource-share --name descriptive_name \
     --region region \
     --resource-arns cluster_arn \
     --principals other_account_ids
   ```

   Windows の場合:

   ```
   aws ram create-resource-share --name descriptive_name ^
     --region region ^
     --resource-arns cluster_arn ^
     --principals other_account_ids
   ```

   `--principals` パラメータに複数アカウント ID を含めるには、ID をスペースで区切ります。許可されたアカウント ID が、AWS 組織の外部かどうかを指定するには、`--allow-external-principals` に `--no-allow-external-principals` または `create-resource-share` パラメータを含めます。

AWS RAM API

クラスターのクローンを作成するためのアクセス許可を付与するには

1. 必須パラメータに関する情報を収集します。クラスターの ARN と他の AWS アカウントの数値 ID が必要です。
2. AWS RAM API オペレーション CreateResourceShare を呼び出し、次の値を指定します。

   - 1 つ以上の AWS アカウントのアカウント ID を `principals` パラメータとして指定します。
   - 1 つ以上の Aurora DB クラスターの ARN を `resourceArns` パラメータとして指定します。

- 許可されたアカウント ID が、AWS 組織の外部かどうかを指定するには、`allowExternalPrincipals` パラメータにブール値を含めます。

デフォルトの RDS キーを使用するクラスターを再作成する

共有する予定の暗号化クラスターでデフォルトの RDS キーを使用している場合は、クラスターを再作成します。これを行うには、DB クラスターの手動スナップショットを作成し、AWS KMS key を使用して、クラスターを新しいクラスターに復元します。その後、新しいクラスターを共有します。このプロセスを実行するには、次のステップを実行します。

デフォルトの RDS キーを使用する暗号化クラスターを再作成するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインから、[スナップショット] を選択します。
3. スナップショットを選択します。
4. [アクション] で、[スナップショットのコピー] を選択後、[暗号化の有効化] を選択します。
5. [AWS KMS key] では、使用する新しい暗号化キーを選択します。
6. コピーしたスナップショットを復元します。これを行うには、「DB クラスターのスナップショットからの復元」の手順に従います。新しい DB インスタンスでは、新しい暗号化キーが使用されます。
7. (オプション) 不要になった古い DB クラスターは削除します。これを行うには、「DB クラスターのスナップショットの削除」の手順に従います。削除する前に、新しいクラスターに必要なデータがすべて揃っていることと、アプリケーションでそれらに問題なくアクセスできることを確認します。

所有しているクラスターが他の AWS アカウントと共有されているかどうかを確認する

クラスターを共有するためのアクセス許可が他のユーザーに付与されているかどうかを確認することができます。確認することで、クラスターがクロスアカウントのクローンの最大数の制限に近づいているかどうかを把握することができます。

AWS RAM コンソールを使用してリソースを共有する手順については、AWS RAM ユーザーガイドの「お客様が所有しているリソースの共有」を参照してください。

AWS CLI

所有しているクラスターが他の AWS アカウントと共有されているかどうかを確認するには

- AWS RAM CLI コマンド `list-principals` を呼び出します。この際、アカウント ID をリソースの所有者として、クラスターの ARN をリソース ARN として使用します。共有はすべて、次のコマンドで確認することができます。クラスターのクローン作成を許可されている AWS アカウントが結果で示されます。

```
aws ram list-principals \
    --resource-arns your_cluster_arn \
    --principals your_aws_id
```

AWS RAM API

所有しているクラスターが他の AWS アカウントと共有されているかどうかを確認するには

- AWS RAM API オペレーション ListPrincipals を呼び出します。アカウント ID をリソースの所有者として、クラスターの ARN をリソース ARN として使用します。

別の AWS アカウントが所有するクラスターのクローンを作成する

別の AWS アカウントが所有するクラスターのクローンを作成するには、AWS RAM を使用して、クローンを作成するためのアクセス許可を取得します。必要なアクセス許可を取得したら、Aurora クラスターのクローンを作成するためのスタンダード的な手順を行います。

また、所有しているクラスターが、別の AWS アカウントが所有するクラスターのクローンであるかどうかを確認することもできます。

AWS RAM コンソールで、別のアカウントが所有するリソースを使用する手順については、AWS RAM ユーザーガイドの「お客様が共有先になっているリソースへのアクセス」を参照してください。

トピック

- 他の AWS アカウントが所有するクラスターのクローン作成の招待を表示する
- 他の AWS アカウントが所有するクラスターを共有する招待を承諾する
- 別の AWS アカウントが所有する Aurora クラスターのクローンを作成する
- DB クラスターがクロスアカウントのクローンであるかどうかを確認する

他の AWS アカウントが所有するクラスターのクローン作成の招待を表示する

他の AWS 組織の AWS アカウントが所有するクラスターのクローンを作成するための招待を表示するには、AWS CLI、AWS RAM コンソール、または AWS RAM API を使用します。現在、Amazon RDS コンソールを使用してこの手順を実行することはできません。

AWS RAM コンソールで招待を扱う手順については、AWS RAM ユーザーガイドの「お客様が共有先になっているリソースへのアクセス」を参照してください。

AWS CLI

他の AWS アカウントが所有するクラスターのクローンを作成する招待を表示するには

1. AWS RAM CLI コマンド `get-resource-share-invitations` を実行します。

   ```
   aws ram get-resource-share-invitations --region region_name
   ```

   上記のコマンドの結果には、クラスターのクローンを作成するためのすべての招待が表示されます。これには、既に承認または却下されたものも含まれます。

2. (オプション) 自分のアクションが必要な招待のみ表示されるようにリストをフィルタリングする そのためには、`--query 'resourceShareInvitations[?status==`PENDING`]'` パラメータを追加します。

AWS RAM API

他の AWS アカウントが所有するクラスターのクローンを作成する招待を表示するには

1. AWS RAM API オペレーション `GetResourceShareInvitations` を呼び出します。このオペレーションでは、既に承認または却下されたものを含め、すべての招待が返ります。
2. (オプション) 自分のアクションが必要な招待のみを検索するには、`resourceShareAssociations` 戻りフィールドで値が `status` の `PENDING` を確認します。

他の AWS アカウントが所有するクラスターを共有する招待を承諾する

別の AWS 組織の他の AWS アカウントが所有するクラスターを共有するための招待を承認することができます。これらの招待を扱うには、AWS CLI、AWS RAM と RDS API、または AWS RAM コンソールを使用します。現在、RDS コンソールを使用してこの手順を実行することはできません。

AWS RAM コンソールで招待を扱う手順については、AWS RAM ユーザーガイドの「お客様が共有先になっているリソースへのアクセス」を参照してください。

AWS CLI

別の AWS アカウントからクラスターを共有するための招待を承認するには

1. 招待 ARN を検索するには、前述されているように、AWS RAM CLI コマンド `get-resource-share-invitations` を実行します。
2. 招待を承認するには、前述されているように、AWS RAM CLI コマンド `accept-resource-share-invitation` を呼び出します。

   Linux、macOS、Unix の場合:

   ```
   aws ram accept-resource-share-invitation \
     --resource-share-invitation-arn invitation_arn \
     --region region
   ```

   Windows の場合:

   ```
   aws ram accept-resource-share-invitation ^
     --resource-share-invitation-arn invitation_arn ^
     --region region
   ```

AWS RAM と RDS API

別のアカウントのクラスターを共有するための招待を承認するには

1. 招待 ARN を検索するには、前述されているように、AWS RAM API オペレーション `GetResourceShareInvitations` を呼び出します。
2. `resourceShareInvitationArn` パラメータとして ARN を RDS API オペレーション AcceptResourceShareInvitation を渡します。

別の AWS アカウントが所有する Aurora クラスターのクローンを作成する

DB クラスターを所有する AWS アカウントからの招待を承認したら、前述のように、クラスターのクローンを作成することができます。

コンソール

別の AWS アカウントが所有する Aurora クラスターのクローンを作成するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択します。

   データベースリストの上部に、[ロール] 値が `Shared from account #account_id` の項目が 1 つ以上表示されます。セキュリティ上の理由により、元のクラスターに関する情報は一部のみ表示されます。表示されているプロパティ(データベースエンジン、バージョンなど) は、クローン作成されたクラスターのものと同じです 。
3. クローンを作成するクラスターを選択します。
4. [Actions] の [クローンの作成] を選択します。
5. コンソール の手順に従って、クローンを作成したクラスターの設定を終了します。
6. 必要に応じて、クローンを作成したクラスターの暗号化を有効にします。クローンを作成しているクラスターが暗号化されている場合は、クローンを作成したクラスターの暗号化を有効にする必要があります。また、お客様とクラスターを共有した AWS アカウントは、クラスターの暗号化に使用した KMS キーも共有する必要があります。同じ KMS キーを使用してクローンを暗号化したり、独自の KMS キーを暗号化したりすることもできます。デフォルトの KMS キーで暗号化されているクラスターのクロスアカウントクローンを作成することはできません。

   暗号化キーを所有するアカウントは、キーポリシーを使用して送信先アカウントにキーを使用するためのアクセス許可を付与する必要があります。このプロセスは、暗号化されたスナップショットが共有される方法と似ています。つまり、キーポリシーを使用して、キーを使用するためのアクセス許可を送信先アカウントに付与します。

AWS CLI

別の AWS アカウントが所有する Aurora クラスターのクローンを作成するには

1. 前述のように、DB クラスターを所有する AWS アカウントからの招待を承認します。
2. クラスターのクローンを作成するには、前述のように、RDS CLI コマンド `source-db-cluster-identifier` の `restore-db-cluster-to-point-in-time` パラメータで、ソースクラスターのフル ARN を指定します。

source-db-cluster-identifier として渡した ARN が共有されていない場合は、指定されたクラスターが存在しないかのように同じエラーが返ります。

Linux、macOS、Unix の場合:

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier=arn:aws:rds:arn_details \
  --db-cluster-identifier=new_cluster_id \
  --restore-type=copy-on-write \
  --use-latest-restorable-time
```

Windows の場合:

```
aws rds restore-db-cluster-to-point-in-time ^
  --source-db-cluster-identifier=arn:aws:rds:arn_details ^
  --db-cluster-identifier=new_cluster_id ^
  --restore-type=copy-on-write ^
  --use-latest-restorable-time
```

3. クローンを作成するクラスターが暗号化されている場合は、kms-key-id パラメータを含めて、クローンを作成したクラスターを暗号化します。この kms-key-id 値は、元の DB クラスターの暗号化に使用したのと同じ値か、独自の KMS キーになります。その暗号化キーを使用するためのアクセス許可が、お客様のアカウントに付与されている必要があります。

   Linux、macOS、Unix の場合:

   ```
   aws rds restore-db-cluster-to-point-in-time \
     --source-db-cluster-identifier=arn:aws:rds:arn_details \
     --db-cluster-identifier=new_cluster_id \
     --restore-type=copy-on-write \
     --use-latest-restorable-time \
     --kms-key-id=arn:aws:kms:arn_details
   ```

   Windows の場合:

   ```
   aws rds restore-db-cluster-to-point-in-time ^
     --source-db-cluster-identifier=arn:aws:rds:arn_details ^
     --db-cluster-identifier=new_cluster_id ^
     --restore-type=copy-on-write ^
     --use-latest-restorable-time ^
   ```

```
  --kms-key-id=arn:aws:kms:arn_details
```

暗号化キーを所有するアカウントは、キーポリシーを使用して送信先アカウントにキーを使用するためのアクセス許可を付与する必要があります。このプロセスは、暗号化されたスナップショットが共有される方法と似ています。つまり、キーポリシーを使用して、キーを使用するためのアクセス許可を送信先アカウントに付与します。キーポリシーの例を次に示します。

```
{
  "Id": "key-policy-1",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow use of the key",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Allow attachment of persistent resources",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": "*",
      "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
```

```
    }
  ]
}
```

> **Note**
>
> restore-db-cluster-to-point-in-time AWS CLI コマンドは、その DB クラスターの DB インスタンスではなく、DB クラスターのみを復元します。復元された DB クラスターの DB インスタンスを作成するには、create-db-instanceコマンドを起動します。復元された DB クラスターの ID を `--db-cluster-identifier` に指定します。`restore-db-cluster-to-point-in-time` コマンドが完了し、DB クラスターが使用可能になった後でのみ、DB インスタンスを作成できます。

RDS API

別の AWS アカウントが所有する Aurora クラスターのクローンを作成するには

1. 前述のように、DB クラスターを所有する AWS アカウントからの招待を承認します。
2. クラスターのクローンを作成するには、RDS API オペレーション `SourceDBClusterIdentifier` の `RestoreDBClusterToPointInTime` パラメータで、ソースクラスターのフル ARN を指定します。

   `SourceDBClusterIdentifier` として渡した ARN が共有されていない場合は、指定されたクラスターが存在しないかのように同じエラーが返ります。
3. クローンを作成しているクラスターが暗号化されている場合は、クローン作成されたクラスターを暗号化するように `KmsKeyId` パラメータを含めます。この `kms-key-id` 値は、元の DB クラスターの暗号化に使用したのと同じ値か、独自の KMS キーになります。その暗号化キーを使用するためのアクセス許可が、お客様のアカウントに付与されている必要があります。

   ボリュームのクローンを作成する場合は、ソースクラスターの暗号化に使用した暗号化キーを使用するためのアクセス許可が送信先アカウントに付与されている必要があります。Aurora は、`KmsKeyId` に指定した暗号化キーを使用して、新しいクローンクラスターを暗号化します。

   暗号化キーを所有するアカウントは、キーポリシーを使用して送信先アカウントにキーを使用するためのアクセス許可を付与する必要があります。このプロセスは、暗号化されたスナップ

ショットが共有される方法と似ています。つまり、キーポリシーを使用して、キーを使用するためのアクセス許可を送信先アカウントに付与します。キーポリシーの例を次に示します。

```
{
  "Id": "key-policy-1",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow use of the key",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Allow attachment of persistent resources",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": "*",
      "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
    }
  ]
}
```

> **Note**
>
> RestoreDBClusterToPointInTime RDS API オペレーションでは、DB クラスターのみが復元され、その DB クラスターの DB インスタンスは復元されません。復元された DB クラスターの DB インスタンスを作成するには、RDS API オペレーション CreateDBInstance を呼び出します。復元された DB クラスターの ID を DBClusterIdentifier に指定します。RestoreDBClusterToPointInTime オペレーションが完了し、DB クラスターのステータスが使用可能になった後でのみ、DB インスタンスを作成できます。

DB クラスターがクロスアカウントのクローンであるかどうかを確認する

DBClusters オブジェクトは、各クラスターがクロスアカウントのクローンであるかどうかを判別します。クローンを作成するアクセス許可が付与されているクラスターを確認するには、RDS CLI コマンド include-shared を実行するときに describe-db-clusters オプションを使用します。ただし、そのようなクラスターの設定詳細の大分部は確認できません。

AWS CLI

DB クラスターがクロスアカウントのクローンであるかどうかを確認するには

- RDS CLI コマンド describe-db-clusters を呼び出します。

  以下の例では、実際または可能性のあるクロスアカウントクローンの DB クラスターが describe-db-clusters 出力に表示される様子を示します。お客様の AWS アカウントが所有する既存のクラスターの場合は CrossAccountClone フィールドに、クラスターがお客様の AWS アカウントが所有する DB クラスターのクローンかどうかが示されます。

  場合によっては、エントリの AWS フィールドにお客様のアカウントとは異なる DBClusterArn アカウント番号が含まれていることがあります。この場合、そのエントリは別の AWS アカウントが所有し、クローン作成が可能なクラスターであることを表します。そのようなエントリには、DBClusterArn 以外のフィールドがいくつか含まれます。クローン作成されたクラスターを作成する際は、元のクラスターと同じ StorageEncrypted、Engine、および EngineVersion 値を指定します。

```
$aws rds describe-db-clusters --include-shared --region us-east-1
{
  "DBClusters": [
      {
          "EarliestRestorableTime": "2023-02-01T21:17:54.106Z",
```

```
            "Engine": "aurora-mysql",
            "EngineVersion": "8.0.mysql_aurora.3.02.0",
            "CrossAccountClone": false,
...
        },
        {
            "EarliestRestorableTime": "2023-02-09T16:01:07.398Z",
            "Engine": "aurora-mysql",
            "EngineVersion": "8.0.mysql_aurora.3.02.0",
            "CrossAccountClone": true,
...
        },
        {
            "StorageEncrypted": false,
            "DBClusterArn": "arn:aws:rds:us-east-1:12345678:cluster:cluster-
abcdefgh",
            "Engine": "aurora-mysql",
            "EngineVersion": "8.0.mysql_aurora.3.02.0
    ]
}
```

RDS API

DB クラスターがクロスアカウントのクローンであるかどうかを確認するには

- RDS API オペレーション DescribeDBClusters を呼び出します。

  お客様の AWS アカウントが所有する既存のクラスターの場合は `CrossAccountClone` フィールドに、クラスターが別の AWS アカウントが所有する DB クラスターのクローンかどうかが示されます。AWS フィールドに別の `DBClusterArn` アカウント番号を持つエントリは、クローン作成が可能で、他の AWS アカウントが所有するクラスターであることを表します。このようなエントリには、`DBClusterArn` 以外のフィールドがいくつか含まれます。クローン作成されたクラスターを作成する際は、元のクラスターと同じ `StorageEncrypted`、`Engine`、および `EngineVersion` 値を指定します。

  次の例は、実際のクローンクラスターと潜在的なクローンクラスターの両方を示す戻り値を示しています。

```
{
  "DBClusters": [
      {
```

```
            "EarliestRestorableTime": "2023-02-01T21:17:54.106Z",
            "Engine": "aurora-mysql",
            "EngineVersion": "8.0.mysql_aurora.3.02.0",
            "CrossAccountClone": false,
...
        },
        {
            "EarliestRestorableTime": "2023-02-09T16:01:07.398Z",
            "Engine": "aurora-mysql",
            "EngineVersion": "8.0.mysql_aurora.3.02.0",
            "CrossAccountClone": true,
...
        },
        {
            "StorageEncrypted": false,
            "DBClusterArn": "arn:aws:rds:us-east-1:12345678:cluster:cluster-
abcdefgh",
            "Engine": "aurora-mysql",
            "EngineVersion": "8.0.mysql_aurora.3.02.0"
        }
    ]
}
```

Aurora と他の AWS のサービスの統合

Amazon Aurora を他の AWS のサービスと統合することで、AWS クラウドの追加機能を使用できるように Aurora DB クラスターを拡張できます。

トピック

- AWS のサービスと Amazon Aurora MySQL の統合
- AWS のサービスと Amazon Aurora PostgreSQL の統合
- Aurora レプリカでの Amazon Aurora Auto Scaling の使用

AWS のサービスと Amazon Aurora MySQL の統合

Amazon Aurora MySQL を他の AWS のサービスと統合することで、Aurora MySQL DB クラスターを拡張して AWS クラウドの追加機能を使用できるようになります。Aurora MySQL DB クラスターでは、AWS のサービスを使用して以下のことができます。

- ネイティブ関数 AWS Lambda または `lambda_sync` を使用して、`lambda_async` 関数を同期または非同期に呼び出します。または、AWS Lambda プロシージャを使用して `mysql.lambda_async` 関数を非同期に呼び出します。
- `LOAD DATA FROM S3` コマンドまたは `LOAD XML FROM S3` コマンドを使用して、Amazon S3 バケットに格納されているテキストファイルや XML ファイルのデータを DB クラスター内にロードする。
- `SELECT INTO OUTFILE S3` コマンドを使用して DB クラスターから Amazon S3 バケット内のテキストファイルにデータを保存する。
- アプリケーションの Auto Scaling で Aurora レプリカを自動的に追加または削除します。詳細については、「Aurora レプリカでの Amazon Aurora Auto Scaling の使用」を参照してください。

Aurora MySQL と他の AWS のサービスとの統合の詳細については、「Amazon Aurora MySQL と他の AWS のサービスの統合」を参照してください。

AWS のサービスと Amazon Aurora PostgreSQL の統合

Amazon Aurora PostgreSQL を他の AWS のサービスと統合することで、AWS クラウドの追加機能を使用するために、Aurora PostgreSQL DB クラスターを拡張できます。Aurora PostgreSQL DB クラスターでは、AWS のサービスを使用して以下のことができます。

- Performance Insights を使用してリレーショナルデータベースのワークロードのパフォーマンスを迅速に収集、表示、および評価します。
- Aurora Auto Scaling で Aurora レプリカを自動的に追加または削除します。詳細については、「Aurora レプリカでの Amazon Aurora Auto Scaling の使用」を参照してください。

Aurora PostgreSQL と他の AWS のサービスとの統合の詳細については、「Amazon Aurora PostgreSQL を他の AWS のサービスと統合する」を参照してください。

Aurora レプリカでの Amazon Aurora Auto Scaling の使用

接続およびワークロード要件を満たすために、Aurora Auto Scaling は、Aurora DB クラスター用にプロビジョニングされる Aurora レプリカ (リーダー DB インスタンス) の数を動的に調整します。Aurora Auto Scaling は、Aurora MySQL と Aurora PostgreSQL の両方で使用できます。Aurora Auto Scaling により、お使いの Aurora DB クラスターは急激な接続やワークロードの増加を処理できます。接続やワークロードが減ると、Aurora Auto Scaling は未使用のプロビジョニングされた DB インスタンスに対する料金が発生しないように、不要な Aurora レプリカを削除します。

スケーリングポリシーを定義して Aurora DB クラスターに適用します。スケーリングポリシーは、Aurora Auto Scaling で管理できる Aurora レプリカの最小数と最大数を定義します。そのポリシーに基づいて、Aurora Auto Scaling は Amazon CloudWatch メトリクスとターゲット値を使用して決定される実際のワークロードに応じて、Aurora レプリカの数を上下に調整します。

AWS Management Console を使用し、事前定義されたメトリクスに基づいてスケーリングポリシーを適用できます。代わりに、AWS CLI または Aurora Auto Scaling API を使用し、事前定義済みまたはカスタムのメトリクスに基づいたスケーリングポリシーを適用することも可能です。

トピック

開始する前に

Aurora DB クラスターで Aurora Auto Scaling を使用する前に、まず、プライマリ (書き込み) DB インスタンスで Aurora DB クラスターを作成する必要があります。Aurora DB クラスター作成の詳細については、「Amazon Aurora DB クラスターの作成」を参照してください。

Aurora Auto Scaling は、DB クラスターが利用可能な状態にある場合のみ、DB クラスターをスケーリングします。

Aurora Auto Scaling が新しい Aurora レプリカを追加すると、その新しい Aurora レプリカはプライマリインスタンスが使用するのと同じ DB インスタンスクラスになります。DB インスタンスクラスの詳細については、「Aurora DB インスタンスクラス」を参照してください。また、新しい Aurora レプリカの昇格階層は、最も低い優先順位 (デフォルトでは 15) に設定されています。つまり、フェイルオーバー時には、手動で作成されたものなど、優先順位の高いレプリカが最初に昇格されます。詳細については、「Aurora DB クラスターの耐障害性」を参照してください。

Aurora Auto Scaling では、自身が作成した Aurora レプリカのみ削除されます。

Aurora Auto Scaling から益を得るには、お使いのアプリケーションが新しい Aurora レプリカへの接続をサポートしている必要があります。これを行うには、Aurora 読み込みエンドポイントを使用することをお勧めします。AWS JDBC ドライバーなどのドライバーを使用できます。詳細については、「Amazon Aurora DB クラスターへの接続」を参照してください。

> **Note**
>
> Aurora グローバルデータベースは、現在、セカンダリ DB クラスターの Aurora Auto Scaling をサポートしていません。

Aurora Auto Scaling ポリシー

Aurora Auto Scaling はスケーリングポリシーを使用して、Aurora DB クラスターの Aurora レプリカ数を調整します。Aurora Auto Scaling には以下のコンポーネントがあります。

- サービスにリンクされたロール
- ターゲットメトリクス
- 最小容量と最大容量
- クールダウン期間

トピック

- サービスにリンクされたロール
- ターゲットメトリクス
- 最小容量と最大容量
- クールダウン期間
- スケールインアクティビティの有効化または無効化

サービスにリンクされたロール

Aurora Auto Scaling は、サービスにリンクされたロール `AWSServiceRoleForApplicationAutoScaling_RDSCluster` を使用します。詳細については、Application Auto Scaling ユーザーガイドの「Application Auto Scaling のサービスにリンクされたロール」を参照してください。

ターゲットメトリクス

このタイプのポリシーでは、ターゲット追跡スケーリングポリシー設定で、事前定義されたメトリクスまたはカスタムメトリクスとそのメトリクスのターゲット値を指定します。Aurora Auto Scaling は、スケーリングポリシーをトリガーする CloudWatch アラームを作成および管理し、メトリクスとターゲット値に基づいてスケーリング調整値を計算します。スケーリングポリシーは、指定されたターゲット値、またはそれに近い値にメトリクスを維持するため、必要に応じて Aurora レプリカを追加または削除します。メトリクスをターゲット値に近い値に維持することに加えて、ターゲット追跡スケーリングポリシーは、変化するワークロードによるメトリクスの変動に適応します。そのようなポリシーは、DB クラスターに使用可能な Aurora レプリカ数の急速な変動の最小化もします。

例として、事前定義された平均 CPU 使用率メトリクスを使用するスケーリングポリシーを挙げましょう。そのようなポリシーは、40 パーセントなどの指定された使用率に、またはそれに近い割合に CPU 使用率を維持できます。

> **Note**
>
> 各 Aurora DB クラスターについては、各ターゲットメトリクスに対して 1 つの Auto Scaling ポリシーのみを作成できます。

最小容量と最大容量

アプリケーションの Auto Scaling が管理する Aurora レプリカの最大数を指定できます。この値は 0–15 に設定される必要があります。また、Aurora レプリカの最小数に指定された値以上である必要があります。

アプリケーションの Auto Scaling が管理する Aurora レプリカの最小数も指定できます。この値は 0–15 に設定される必要があります。また、Aurora レプリカの最大数に指定された値以下である必要があります。

> **Note**
>
> 最小容量と最大容量は、Aurora DB クラスターに対して設定されます。指定された値は、その Aurora DB クラスターに関連付けられたポリシーすべてに適用されます。

クールダウン期間

Aurora DB クラスターのスケールインやスケールアウトに影響するクールダウン期間を追加することで、ターゲット追跡スケーリングポリシーの応答性を調整できます。クールダウン期間を設定すると、その期間が過ぎるまでその後のスケールインやスケールアウトのリクエストがブロックされます。これらのブロックにより、スケールインリクエストのための Aurora DB クラスターの Aurora レプリカの削除、およびスケールアウトリクエストのための Aurora レプリカの作成を遅らせます。

以下のクールダウン期間を指定できます。

- スケールインアクティビティは、Aurora DB クラスターの Aurora レプリカ数を減らします。スケールインのクールダウン期間は、スケールインアクティビティが完了してから別のスケールインアクティビティが開始されるまでの時間 (秒) を指定します。
- スケールアウトアクティビティは、Aurora DB クラスターの Aurora レプリカ数を増やします。スケールアウトのクールダウン期間は、スケールアウトアクティビティが完了してから別のスケールアウトアクティビティが開始されるまでの時間 (秒) を指定します。

> **Note**
>
> 後続のスケールアウトのリクエストが最初のリクエストよりも多くの Aurora レプリカを対象とする場合、スケールアウトのクールダウン期間は無視されます。

スケールインやスケールアウトのクールダウン期間を設定しない場合、それぞれのデフォルト値は 300 秒です。

スケールインアクティビティの有効化または無効化

ポリシーに対してスケールインアクティビティを有効化または無効化できます。スケールインアクティビティを有効にすると、スケーリングポリシーは Aurora レプリカを削除できます。スケールインアクティビティが有効な場合、スケーリングポリシーのスケールインのクールダウン期間がスケールインアクティビティに適用されます。スケールインアクティビティを無効にすると、スケーリングポリシーは Aurora レプリカを削除できなくなります。

> **Note**
>
> スケールアウトアクティビティは、スケーリングポリシーが必要に応じて Aurora レプリカを作成できるように、常に有効にしておきます。

Aurora DB クラスターへのスケーリングポリシーの追加

AWS Management Console、AWS CLI、または アプリケーションの Auto Scaling API を使用してスケーリングポリシーを追加できます。

> **Note**
>
> AWS CloudFormation を使用してスケーリングポリシーを追加する例については、AWS CloudFormation ユーザーガイドの「Aurora DB クラスターのスケーリングポリシーの宣言」を参照してください。

コンソール

AWS Management Console を使用して、Aurora DB クラスターにスケーリングポリシーを追加できます。

Aurora DB クラスターに Auto Scaling ポリシーを追加するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[データベース] を選択します。

3. ポリシーを追加する Aurora DB クラスターを選択します。
4. [ログとイベント] タブを選択します。
5. [Auto Scaling policies (Auto Scaling ポリシー)] セクションで、[Add (追加)] を選択します。

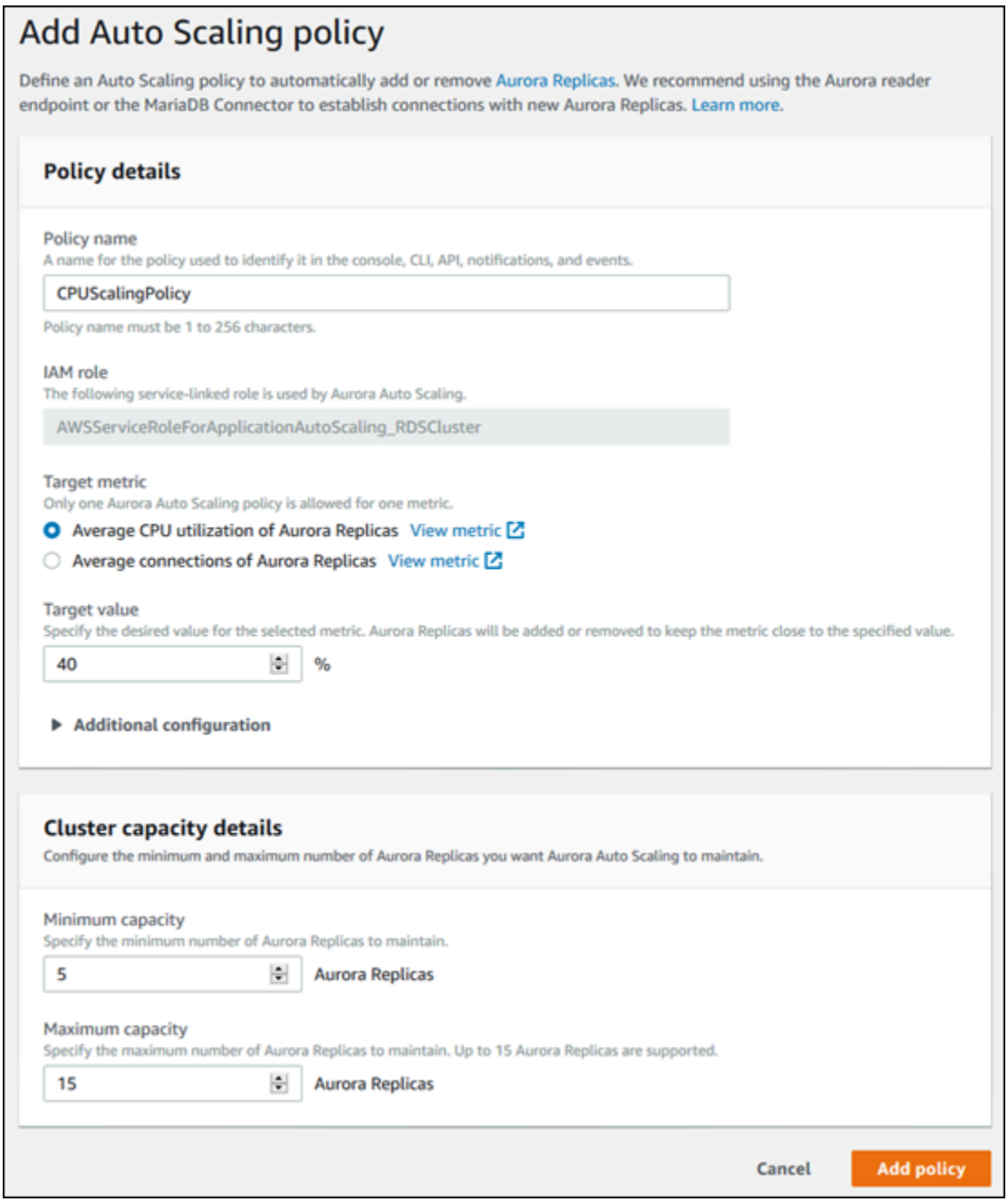
   [Add Auto Scaling policy] ダイアログボックスが表示されます。
6. [ポリシー名] に、ポリシー名を入力します。
7. ターゲットメトリクスには、以下のいずれかを選択します。

   - 平均 CPU 使用率に基づいてポリシーを作成するための [Aurora レプリカの平均 CPU 使用率]。
   - Aurora レプリカへの平均接続数に基づいてポリシーを作成するための [Aurora レプリカの平均接続数]。
8. ターゲット値には、以下のいずれかを入力します。

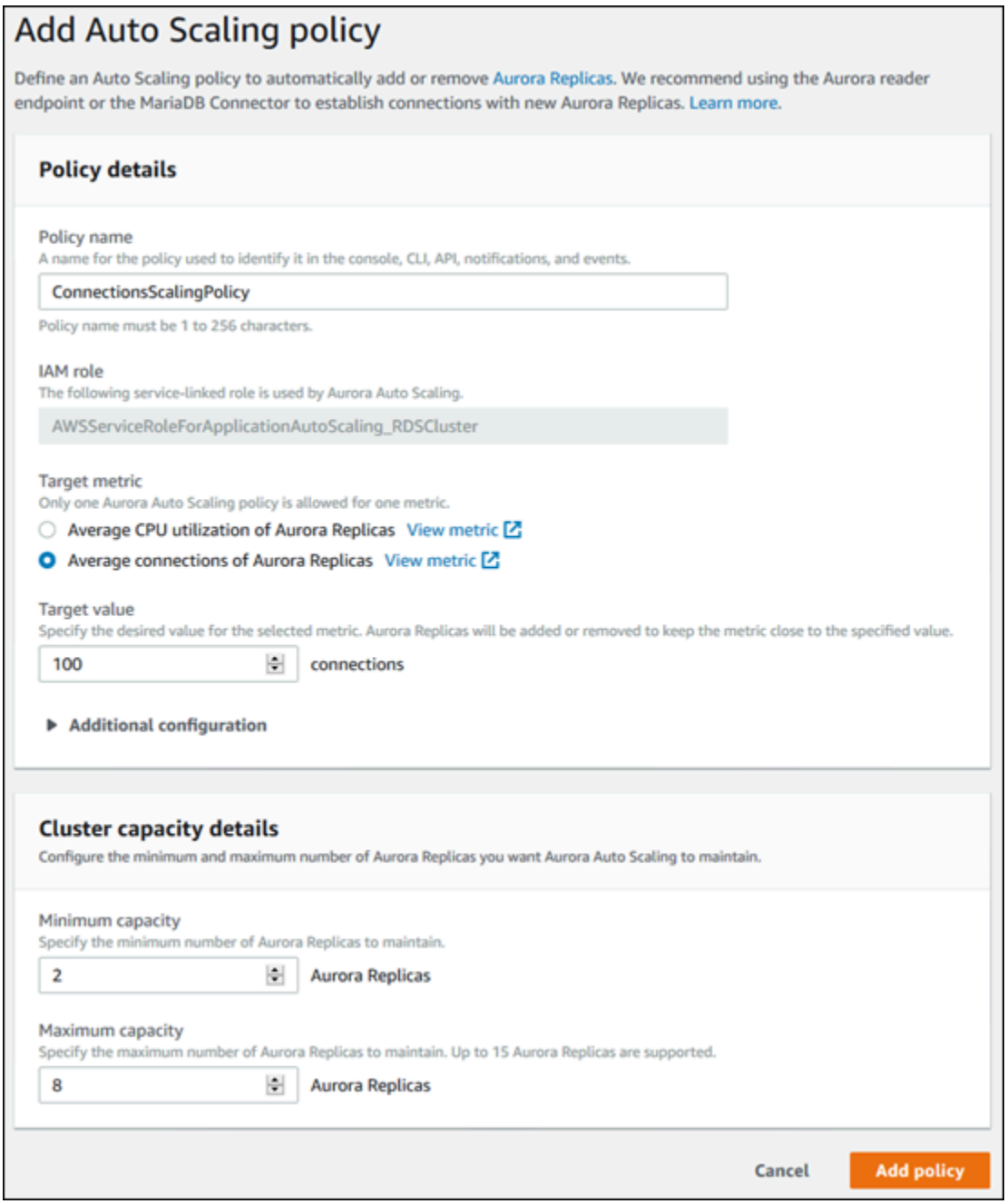
   - 前のステップで [Aurora レプリカの平均 CPU 使用率] を選択した場合、Aurora レプリカに維持する CPU 使用率を入力します。
   - 前のステップで [Aurora レプリカの平均接続数] を選択した場合は、維持する接続数を入力します。

   Aurora レプリカが追加または削除され、メトリクスが指定された値に近い値に維持されます。
9. (省略可能) [Additional Configuration] (追加設定) を展開し、スケールインまたはスケールアウトのクールダウン期間を作成します。
10. [Minimum capacity (最小キャパシティー)] には、Aurora Auto Scaling ポリシーが維持する必要がある Aurora レプリカの最小数を入力します。
11. [Maximum capacity (最大キャパシティー)] には、Aurora Auto Scaling ポリシーが維持する必要がある Aurora レプリカの最大数を入力します。
12. [Add policy] を選択します。

次のダイアログボックスでは、Auto Scaling ポリシーを 40 パーセントの平均 CPU 使用率に基づいて作成します。このポリシーでは、最小 5 個の Aurora レプリカと最大 15 個の Aurora レプリカを指定します。



次のダイアログボックスでは、平均接続数 100 に基づいて、Auto Scaling ポリシーを作成します。このポリシーでは、最小 2 個の Aurora レプリカと最大 8 個の Aurora レプリカを指定します。



AWS CLI または Application Auto Scaling API

事前定義されたメトリクスまたはカスタムメトリクスに基づいて、スケーリングポリシーを適用できます。そのためには、AWS CLI または アプリケーションの Auto Scaling API を使用します。最初のステップでは、アプリケーションの Auto Scaling で Aurora DB クラスターを登録します。

Aurora DB クラスターの登録

Aurora Auto Scaling を Aurora DB クラスターで使用可能にするには、アプリケーションの Auto Scaling で Aurora DB クラスターを登録します。これは、そのクラスターに適用するスケーリングのディメンションと制限を定義するためです。アプリケーションの Auto Scaling では、Aurora レプリカの数に対応したスケーラブルディメンション `rds:cluster:ReadReplicaCount` で、Aurora DB クラスターを動的にスケーリングします。

Aurora DB クラスターを登録するには、AWS CLI または アプリケーションの Auto Scaling API を使用します。

AWS CLI

Aurora DB クラスターを登録するには、以下のパラメータを指定しながら、AWS CLI コマンドの `register-scalable-target` を使用します。

- `--service-namespace` この値は に設定します。`rds`
- `--resource-id` – Aurora DB クラスターのリソース識別子です。このパラメータでは、リソースタイプは `cluster` で、一意の識別子は Aurora DB クラスターの名前、例えば `cluster:myscalablecluster` です。
- `--scalable-dimension` この値は に設定します。`rds:cluster:ReadReplicaCount`
- `--min-capacity` – アプリケーションの Auto Scaling で管理するリーダー DB インスタンスの最小数。`--min-capacity`、`--max-capacity`、およびクラスター内の DB インスタンスの数の関係については、「最小容量と最大容量」を参照してください。
- `--max-capacity` – アプリケーションの Auto Scaling で管理するリーダー DB インスタンスの最大数。`--min-capacity`、`--max-capacity`、およびクラスター内の DB インスタンスの数の関係については、「最小容量と最大容量」を参照してください。

Example

以下の例では、`myscalablecluster` という名前の Aurora DB クラスターを登録します。この登録は、DB クラスターが 1 から 8 個の Aurora レプリカを持つよう動的にスケールされることを示します。

Linux、macOS、Unix の場合:

```
aws application-autoscaling register-scalable-target \
    --service-namespace rds \
    --resource-id cluster:myscalablecluster \
```

```
    --scalable-dimension rds:cluster:ReadReplicaCount \
    --min-capacity 1 \
    --max-capacity 8 \
```

Windows の場合:

```
aws application-autoscaling register-scalable-target ^
    --service-namespace rds ^
    --resource-id cluster:myscalablecluster ^
    --scalable-dimension rds:cluster:ReadReplicaCount ^
    --min-capacity 1 ^
    --max-capacity 8 ^
```

Application Auto Scaling API

Aurora DB クラスターを アプリケーションの Auto Scaling に登録するには、以下のパラメータを指定して RegisterScalableTarget アプリケーションの Auto Scaling API オペレーションを使用します。

- ServiceNamespace この値は に設定します。rds
- ResourceID – Aurora DB クラスターのリソース識別子です。このパラメータでは、リソースタイプは cluster で、一意の識別子は Aurora DB クラスターの名前、例えば cluster:myscalablecluster です。
- ScalableDimension この値は に設定します。rds:cluster:ReadReplicaCount
- MinCapacity – アプリケーションの Auto Scaling で管理するリーダー DB インスタンスの最小数。MinCapacity、MaxCapacity、およびクラスター内の DB インスタンスの数の関係については、「最小容量と最大容量」を参照してください。
- MaxCapacity – アプリケーションの Auto Scaling で管理するリーダー DB インスタンスの最大数。MinCapacity、MaxCapacity、およびクラスター内の DB インスタンスの数の関係については、「最小容量と最大容量」を参照してください。

Example

以下の例では、myscalablecluster という名前の Aurora DB クラスターを アプリケーションの Auto Scaling API に登録します。この登録は、DB クラスターが 1 ~ 8 個の Aurora レプリカを持つよう動的にスケールされることを示します。

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.RegisterScalableTarget
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount",
    "MinCapacity": 1,
    "MaxCapacity": 8
}
```

Aurora DB クラスターのスケーリングポリシーの定義

ターゲット追跡スケーリングポリシー設定は、メトリクスとターゲット値が定義されている JSON ブロックで表されます。JSON ブロックとしてスケーリングポリシー設定をテキストファイルに保存できます。このテキストファイルは、AWS CLI または アプリケーションの Auto Scaling API を呼び出す際に使用します。ポリシー設定構文の詳細については、Application Auto Scaling API リファレンスの「TargetTrackingScalingPolicyConfiguration」を参照してください。

ターゲット追跡スケーリングポリシー設定を定義するには、次のオプションを使用できます。

トピック

- 事前定義メトリクスの使用
- カスタムメトリクスの使用
- クールダウン期間の使用
- スケールインアクティビティの無効化

事前定義メトリクスの使用

定義済みのメトリクスを使用することにより、Aurora Auto Scaling のターゲット追跡と動的スケーリングの両方でうまく動作する Aurora DB クラスターのターゲット追跡スケーリングポリシーを迅速に定義できます。

現在、Aurora は、Aurora Auto Scaling で次の定義済みメトリクスをサポートしています。

- RDSReaderAverageCPUUtilization – Aurora DB クラスター内のすべての Aurora レプリカでの CloudWatch の `CPUUtilization` メトリクスの平均値です。
- RDSReaderAverageDatabaseConnections – Aurora DB クラスター内のすべての Aurora レプリカでの CloudWatch の `DatabaseConnections` メトリクスの平均値です。

`CPUUtilization` と `DatabaseConnections` メトリクスの詳細については、「Amazon Aurora の Amazon CloudWatch メトリクス」を参照してください。

スケーリングポリシーで事前定義メトリクスを使用するには、スケーリングポリシーのターゲット追跡構成を作成します。この設定は、事前定義メトリクスの `PredefinedMetricSpecification` と、そのメトリクスのターゲット値の `TargetValue` が含まれている必要があります。

Example

次の例では、Aurora DB クラスターのターゲット追跡スケーリングの一般的なポリシー設定について説明します。この設定では、`RDSReaderAverageCPUUtilization` 事前定義メトリクスを使用して、すべての Aurora レプリカでの平均 CPU 使用率 40% に基づいて Aurora DB クラスターが調整されます。

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    }
}
```

カスタムメトリクスの使用

カスタムメトリクスを使用することで、カスタム要件を満たすターゲット追跡スケーリングポリシーを定義できます。スケーリングに比例して変化する Aurora メトリクスに基づいて、カスタムメトリクスを定義することができます。

Aurora のすべてのメトリクスがターゲット追跡に使用できるわけではありません。メトリクスは、有効な使用率メトリクスで、インスタンスの使用頻度を示す必要があります。Aurora DB クラスター内の Aurora レプリカの数に比例してメトリクスの値を増減する必要があります。この比例的な増加または減少は、比例的にスケールアウトするため、または Aurora レプリカの数にメトリクスデータを使用するために必要です。

Example

次の例では、スケーリングポリシーのターゲット追跡設定について説明します。この設定では、カスタムメトリクスにより、my-db-cluster という名前の Aurora DB クラスター内のすべての Aurora レプリカでの平均 CPU 使用率 50% に基づいて、Aurora DB クラスターが調整されます。

```
{
    "TargetValue": 50,
    "CustomizedMetricSpecification":
    {
        "MetricName": "CPUUtilization",
        "Namespace": "AWS/RDS",
        "Dimensions": [
            {"Name": "DBClusterIdentifier","Value": "my-db-cluster"},
            {"Name": "Role","Value": "READER"}
        ],
        "Statistic": "Average",
        "Unit": "Percent"
    }
}
```

クールダウン期間の使用

ScaleOutCooldown 値を秒単位で指定して、Aurora DB クラスターをスケールアウトするためのクールダウン期間を追加することができます。ScaleInCooldown 値を秒単位で追加して、Aurora DB クラスターをスケールするためのクールダウン期間を追加することができます。ScaleInCooldown と ScaleOutCooldown の詳細については、Application Auto Scaling API リファレンスの「TargetTrackingScalingPolicyConfiguration」を参照してください。

Example

次の例では、スケーリングポリシーのターゲット追跡設定について説明します。この設定では、RDSReaderAverageCPUUtilization 事前定義メトリクスを使用して、Aurora DB クラスターのすべての Aurora レプリカでの平均 CPU 使用率 40% に基づいて Aurora DB クラスターが調整されます。この設定では、10 分間のスケールインのクールダウン期間と 5 分間のスケールアウトのクールダウン期間が提供されます。

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
```

```
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    },
    "ScaleInCooldown": 600,
    "ScaleOutCooldown": 300
}
```

スケールインアクティビティの無効化

スケールインアクティビティを無効にすることにより、Aurora DB クラスターでターゲット追跡スケーリングポリシー設定がスケーリングされないようにできます。スケールインアクティビティを無効にすると、スケーリングポリシーによって Aurora レプリカが削除されることなく、スケーリングポリシーによって必要に応じて作成されます。

DisableScaleIn ブール値を指定して、Aurora DB クラスターのアクティビティのスケールを有効または無効にすることができます。DisableScaleIn の詳細については、Application Auto Scaling API リファレンスの 「TargetTrackingScalingPolicyConfiguration」を参照してください。

Example

次の例では、スケーリングポリシーのターゲット追跡設定について説明します。この設定では、RDSReaderAverageCPUUtilization 事前定義メトリクスは、Aurora DB クラスターのすべての Aurora レプリカでの平均 CPU 使用率 40% に基づいて Aurora DB クラスターを調整します。この設定では、スケーリングポリシーのスケールインアクティビティが無効になります。

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    },
    "DisableScaleIn": true
}
```

Aurora DB クラスターへのスケーリングポリシーの適用

Aurora DB クラスターを アプリケーションの Auto Scaling に登録し、スケーリングポリシーを定義した後、登録された Aurora DB クラスターにスケーリングポリシーを適用します。Aurora DB クラスターにスケーリングポリシーを適用する際には、AWS CLI または アプリケーションの Auto Scaling API を使用できます。

AWS CLI

スケーリングポリシーを Aurora DB クラスターに適用するには、以下のパラメータが指定された コマンドをput-scaling-policyAWS CLI使用します。

- `--policy-name` – スケーリングポリシーの名前。
- `--policy-type` この値は に設定します。`TargetTrackingScaling`
- `--resource-id` – Aurora DB クラスターのリソース識別子です。このパラメータでは、リソースタイプは `cluster` で、一意の識別子は Aurora DB クラスターの名前、例えば `cluster:myscalablecluster` です。
- `--service-namespace` この値は に設定します。`rds`
- `--scalable-dimension` この値は に設定します。`rds:cluster:ReadReplicaCount`
- `--target-tracking-scaling-policy-configuration` – Aurora DB クラスターに使用するターゲット追跡スケーリングポリシー設定。

Example

次の例では、`myscalablepolicy` というターゲット追跡スケーリングポリシーを `myscalablecluster` という名前の Aurora DB クラスターに アプリケーションの Auto Scaling; を使用して適用します。そのためには、`config.json` という名前のファイルに保存されているポリシー設定を使用します。

Linux、macOS、Unix の場合:

```
aws application-autoscaling put-scaling-policy \
    --policy-name myscalablepolicy \
    --policy-type TargetTrackingScaling \
    --resource-id cluster:myscalablecluster \
    --service-namespace rds \
    --scalable-dimension rds:cluster:ReadReplicaCount \
    --target-tracking-scaling-policy-configuration file://config.json
```

Windows の場合:

```
aws application-autoscaling put-scaling-policy ^
    --policy-name myscalablepolicy ^
```

```
    --policy-type TargetTrackingScaling ^
    --resource-id cluster:myscalablecluster ^
    --service-namespace rds ^
    --scalable-dimension rds:cluster:ReadReplicaCount ^
    --target-tracking-scaling-policy-configuration file://config.json
```

Application Auto Scaling API

アプリケーションの Auto Scaling API を使用してスケーリングポリシーを Aurora DB クラスターに適用するには、以下のパラメータを指定して PutScalingPolicy アプリケーションの Auto Scaling API オペレーションを使用します。

- PolicyName – スケーリングポリシーの名前。
- ServiceNamespace この値は に設定します。rds
- ResourceID – Aurora DB クラスターのリソース識別子です。このパラメータでは、リソースタイプは cluster で、一意の識別子は Aurora DB クラスターの名前、例えば cluster:myscalablecluster です。
- ScalableDimension この値は に設定します。rds:cluster:ReadReplicaCount
- PolicyType この値は に設定します。TargetTrackingScaling
- TargetTrackingScalingPolicyConfiguration – Aurora DB クラスターに使用するターゲット追跡スケーリングポリシー設定。

Example

次の例では、myscalablepolicy というターゲット追跡スケーリングポリシーを myscalablecluster という名前の Aurora DB クラスターに アプリケーションの Auto Scaling; を使用して適用します。RDSReaderAverageCPUUtilization 事前定義メトリクスに基づいてポリシー設定を使用します。

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.PutScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
```

```
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount",
    "PolicyType": "TargetTrackingScaling",
    "TargetTrackingScalingPolicyConfiguration": {
        "TargetValue": 40.0,
        "PredefinedMetricSpecification":
        {
            "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
        }
    }
}
```

スケーリングポリシーの編集

AWS Management Console、AWS CLI、または アプリケーションの Auto Scaling API を使用してスケーリングポリシーを編集できます。

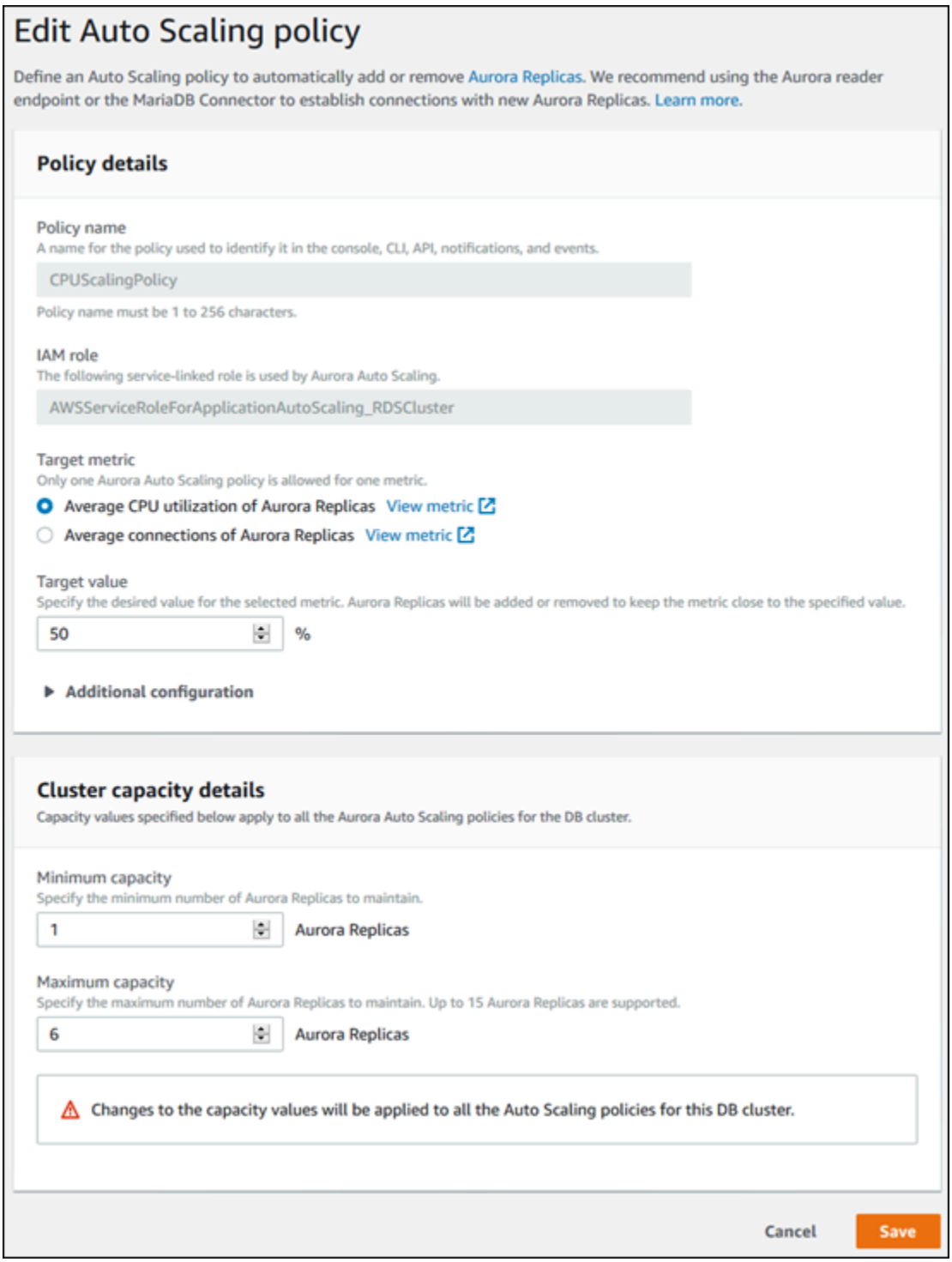
コンソール

AWS Management Console を使用してスケーリングポリシーを編集できます。

Aurora DB クラスターの Auto Scaling ポリシーを編集するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. Auto Scaling ポリシーを編集する Aurora DB クラスターを選択します。
4. [ログとイベント] タブを選択します。
5. [Auto Scaling Policies (Auto Scaling ポリシー)] セクションで Auto Scaling ポリシーを選択してから [Edit (編集)] を選択します。
6. ポリシーを変更します。
7. [Save] を選択します。

以下は、[Edit Auto Scaling policy] ダイアログボックスのサンプルです。



AWS CLI または Application Auto Scaling API

AWS CLI または アプリケーションの Auto Scaling API を使用して、スケーリングポリシーを適用するのと同じ方法でスケーリングポリシーを編集できます。

- AWS CLI を使用する場合は、編集するポリシーの名前を `--policy-name` パラメータで指定します。変更するパラメータの新しい値を指定します。
- アプリケーションの Auto Scaling API を使用する場合は、編集するポリシーの名前を `PolicyName` パラメータで指定します。変更するパラメータの新しい値を指定します。

詳細については、「Aurora DB クラスターへのスケーリングポリシーの適用」を参照してください。

スケーリングポリシーの削除

AWS Management Console、AWS CLI、または アプリケーションの Auto Scaling API を使用してスケーリングポリシーを削除できます。

コンソール

AWS Management Console を使用してスケーリングポリシーを削除できます。

Aurora DB クラスターの Auto Scaling ポリシーを削除するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. Auto Scaling ポリシーを削除する Aurora DB クラスターを選択します。
4. [ログとイベント] タブを選択します。
5. [Auto Scaling Policies (Auto Scaling ポリシー)] セクションで Auto Scaling ポリシーを選択してから [Delete (削除)] を選択します。

AWS CLI

スケーリングポリシーを Aurora DB クラスターから削除するには、以下のパラメータを指定しながら、AWS CLI コマンドの `delete-scaling-policy` を使用します。

- `--policy-name` – スケーリングポリシーの名前。
- `--resource-id` – Aurora DB クラスターのリソース識別子です。このパラメータでは、リソースタイプは `cluster` で、一意の識別子は Aurora DB クラスターの名前、例えば `cluster:myscalablecluster` です。
- `--service-namespace` この値は に設定します。`rds`
- `--scalable-dimension` この値は に設定します。`rds:cluster:ReadReplicaCount`

Example

次の例では、myscalablepolicy というターゲット追跡スケーリングポリシーを myscalablecluster という名前の Aurora DB クラスターから削除します。

Linux、macOS、Unix の場合:

```
aws application-autoscaling delete-scaling-policy \
    --policy-name myscalablepolicy \
    --resource-id cluster:myscalablecluster \
    --service-namespace rds \
    --scalable-dimension rds:cluster:ReadReplicaCount \
```

Windows の場合:

```
aws application-autoscaling delete-scaling-policy ^
    --policy-name myscalablepolicy ^
    --resource-id cluster:myscalablecluster ^
    --service-namespace rds ^
    --scalable-dimension rds:cluster:ReadReplicaCount ^
```

Application Auto Scaling API

スケーリングポリシーを Aurora DB クラスターから削除するには、以下のパラメータを指定して DeleteScalingPolicy アプリケーションの Auto Scaling API オペレーションを使用します。

- PolicyName – スケーリングポリシーの名前。
- ServiceNamespace この値は に設定します。rds
- ResourceID – Aurora DB クラスターのリソース識別子です。このパラメータでは、リソースタイプは cluster で、一意の識別子は Aurora DB クラスターの名前、例えば cluster:myscalablecluster です。
- ScalableDimension この値は に設定します。rds:cluster:ReadReplicaCount

Example

次の例では、アプリケーションの Auto Scaling API を使用して、myscalablepolicy というターゲット追跡スケーリングポリシーを myscalablecluster という名前の Aurora DB クラスターから削除します。

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.DeleteScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount"
}
```

DB インスタンス ID とタグ付け

Aurora Auto Scaling によってレプリカが追加されると、application-autoscaling- によってその DB インスタンス ID に application-autoscaling-61aabbcc-4e2f-4c65-b620-ab7421abc123 などのプレフィックスが付けられます。

次のタグは DB インスタンスに自動的に追加されます。DB インスタンスの詳細ページの [タグ] タブで確認できます。

Tag	値
application-autoscaling:resourceId	cluster:mynewcluster-cluster

Amazon RDS リソースタグの詳細については、「Amazon RDS リソースのタグ付け」を参照してください。

Aurora Auto Scaling と Performance Insights

Performance Insights を使用すると、他の Aurora リーダー DB インスタンスと同様に、Aurora Auto Scaling によって追加されたレプリカをモニタリングできます。

Aurora DB クラスターの Performance Insights を有効にすることはできません。DB クラスター内の各 DB インスタンスに対して、Performance Insights を手動でオンにすることができます。

Aurora DB クラスターのライター DB インスタンスの Performance Insights を有効にしても、リーダー DB インスタンスの Performance Insights は自動的にオンにはなりません。既存のリーダー DB インスタンスと Aurora Auto Scaling によって追加された新しいレプリカについては、Performance Insights を手動でオンにする必要があります。

Performance Insights を使用して Aurora DB クラスターをモニタリングする方法の詳細については、「Amazon Aurora での Performance Insights を使用したDB 負荷のモニタリング」を参照してください。

Amazon Aurora DB クラスターのメンテナンス

Amazon RDS では、Amazon RDS リソースのメンテナンスを定期的に実行します。メンテナンスでは、ほとんどの場合、DB クラスターの以下のリソースの更新が行われます。

- 基盤となるハードウェア
- 基盤となるオペレーティングシステム (OS)
- データベースエンジンのバージョン

通常、オペレーティングシステムのアップデートはセキュリティ問題に関連しています。できるだけ早く実行する必要があります。

一部のメンテナンス項目では、Amazon RDS が DB クラスターを少しの間オフラインにする必要があります。リソースをオフラインにする必要があるメンテナンス項目には、必要なオペレーティングシステムやデータベースのパッチが含まれます。セキュリティやインスタンスの信頼性に関連するパッチのみ、必須のパッチ適用として自動的にスケジューリングされます。そのようなパッチ適用はまれであり、通常は数か月に一度です。メンテナンスウィンドウのごくわずかしか必要としないことがほとんどです。

すぐに適用しないことを選択した遅延 DB クラスターおよびインスタンスの変更も、メンテナンス期間中に適用されます。例えば、メンテナンス期間中に DB インスタンスクラス、クラスターまたは DB パラメータグループの変更を選択できます。保留中の再起動設定を使用して指定した変更は、[保留中のメンテナンス] リストに表示されません。DB クラスターの変更については、「Amazon Aurora DB クラスターの変更」を参照してください。

次のメンテナンスウィンドウに向けて保留中の変更を確認するには、describe-db-clusters AWS CLI コマンドを使用して `PendingModifiedValues` フィールドを確認します。

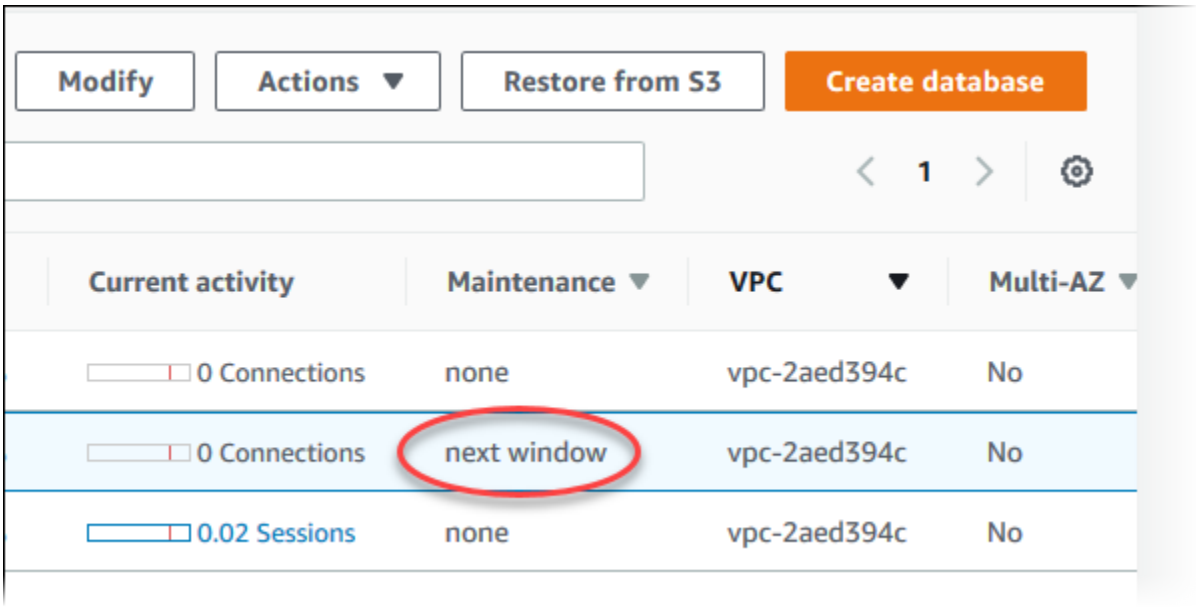
トピック

- 保留中のメンテナンスの表示
- DB クラスターのアップデートを適用する
- Amazon RDS メンテナンスウィンドウ
- DB クラスターの適切なメンテナンスウィンドウの調整
- Aurora DB クラスターのマイナーバージョン自動アップグレード
- Aurora MySQL メンテナンスのアップデート頻度を選択する

- オペレーティングシステムアップデートの操作

保留中のメンテナンスの表示

DB クラスターでメンテナンスによるアップデートが利用可能かどうかは、RDS コンソール、AWS CLI、または Amazon RDS API を使用して確認します。アップデートが利用できる場合は、次に示すように、Amazon RDS コンソールで DB クラスターの [メンテナンス] 列に表示されます。



DB クラスターのメンテナンスアップデートが利用できない場合は、列の値が [なし] になります。

DB クラスターのメンテナンスアップデートが利用できる場合は、列の値が以下のようになります。

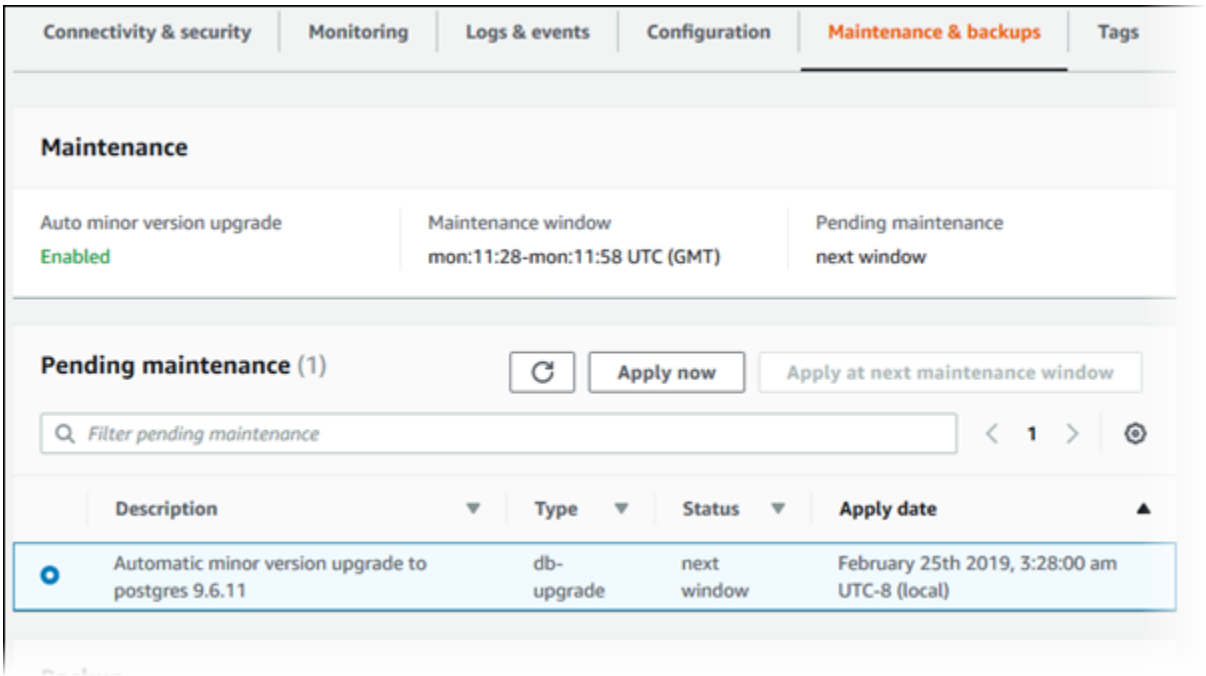
- 必須 - メンテナンスアクションはリソースに適用され、無期限に延期することはできません。
- 利用可能 - メンテナンスアクションは利用可能ですが、自動的にはリソースに適用されません。手動で適用できます。
- 次のウィンドウ - メンテナンスアクションは次回のメンテナンスウィンドウ中にリソースに適用されます。
- 進行中 - メンテナンスアクションはリソースに適用中です。

アップデートを利用できる場合は、いずれかのアクションを実行できます。

- メンテナンス値が [次のウィンドウ] である場合は、[アクション] から [後でアップグレード] を選択してメンテナンス項目を延期します。既にスタートしているメンテナンスアクションは延期できません。

- メンテナンス項目をすぐに適用します。
- メンテナンス項目を次のメンテナンスウィンドウ中にスタートするようにスケジュールを設定します。
- 何のアクションも実行しません。

アクションを実行するには、DB クラスターを選択してその詳細を表示し、次に [メンテナンス & バックアップ] を選択します。保留中のメンテナンス項目が表示されます。



メンテナンスウィンドウは、保留中のオペレーションをスタートする時刻を決定しますが、オペレーションの総実行時間を制限しません。メンテナンスオペレーションは、メンテナンスウィンドウが終了するまでに完了するかどうかは保証されておらず、指定終了時間を超える場合もあります。詳細については、「Amazon RDS メンテナンスウィンドウ」を参照してください。

Amazon Aurora エンジンに対するアップデートと、それらのアップグレードおよびパッチ適用の手順については、「Amazon Aurora MySQL のデータベースエンジンの更新」および「Amazon Aurora PostgreSQL の更新」を参照してください。

また、DB クラスターでメンテナンスによるアップデートが利用可能かどうかは、AWS CLI コマンドの `describe-pending-maintenance-actions` を使用して確認できます。

DB クラスターのアップデートを適用する

Amazon RDS を使用すると、メンテナンスオペレーションを適用するタイミングを選択できます。RDS コンソール、AWS Command Line Interface (AWS CLI)、または RDS API を使用して、Amazon RDS にアップデートを適用するタイミングを指定できます。

> **Note**
>
> RDS for SQL Server の場合、DB インスタンスを停止して起動するか、DB インスタンスクラスをスケールアップしてから再びスケールダウンすることで、基盤となるオペレーティングシステムの更新を適用できます。

コンソール

DB クラスターのアップデートを管理するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. アップデートが必要な DB クラスターを選択します。
4. [アクション] で、以下のいずれかのオプションを選択します。
   - 今すぐアップグレード
   - 次のウィンドウでアップグレード

   > **Note**
   >
   > [次のウィンドウでアップグレード] を選択して、アップデートを延期する場合は、[後でアップグレード] を選択します。既にスタートしているメンテナンスアクションは延期できません。
   > メンテナンスアクションをキャンセルするには、DB インスタンスを変更し、[マイナーバージョン自動アップグレード] を無効にします。

AWS CLI

保留中のアップデートを DB クラスターに適用するには、AWS CLI コマンドの apply-pending-maintenance-action を使用します。

Example

Linux、macOS、Unix の場合:

```
aws rds apply-pending-maintenance-action \
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db \
    --apply-action system-update \
    --opt-in-type immediate
```

Windows の場合:

```
aws rds apply-pending-maintenance-action ^
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db ^
    --apply-action system-update ^
    --opt-in-type immediate
```

> **Note**
>
> メンテナンスアクションを延期するには、`--opt-in-type` として `undo-opt-in` を指定します。メンテナンスアクションが既にスタートされている場合は、`--opt-in-type` として `undo-opt-in` を指定できません。
> メンテナンスアクションをキャンセルするには、modify-db-instance AWS CLI コマンドを実行し、`--no-auto-minor-version-upgrade` を指定します。

少なくとも 1 つの保留中のアップデートがあるリソースのリストを取得するには、describe-pending-maintenance-actions AWS CLI コマンドを使用します。

Example

Linux、macOS、Unix の場合:

```
aws rds describe-pending-maintenance-actions \
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db
```

Windows の場合:

```
aws rds describe-pending-maintenance-actions ^
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db
```

AWS CLI コマンド describe-pending-maintenance-actions の --filters パラメータを指定して、DB クラスターのリソースのリストを取得することもできます。--filters コマンドの形式は、Name=*filter-name*,Value=*resource-id*,... です。

以下は、フィルターの Name パラメータの許容値です。

- db-instance-id - DB インスタンス識別子または Amazon リソースネーム (ARN) のリストが許容されます。返されるリストには、これらの ID または ARN で識別された DB インスタンスの保留中のメンテナンスアクションのみが含まれます。
- db-cluster-id - DB クラスター識別子または Amazon Aurora 用の ARN のリストが許容されます。返されるリストには、これらの ID または ARN で識別された DB クラスターの保留中のメンテナンスアクションのみが含まれます。

次の例では、sample-cluster1 DB クラスターと sample-cluster2 DB クラスターの保留中のメンテナンスアクションが返されます。

Example

Linux、macOS、Unix の場合:

```
aws rds describe-pending-maintenance-actions \
 --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

Windows の場合:

```
aws rds describe-pending-maintenance-actions ^
 --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

RDS API

アップデートを DB クラスターに適用するには、Amazon RDS API の ApplyPendingMaintenanceAction オペレーションを呼び出します。

少なくとも 1 つの保留中のアップデートがあるリソースのリストを返すには、Amazon RDS API の DescribePendingMaintenanceActions オペレーションを呼び出します。

Amazon RDS メンテナンスウィンドウ

メンテナンスウィンドウは、システムの変更が適用される週単位の時間間隔です。各 DB クラスターには、週ごとのメンテナンスウィンドウがあります。メンテナンスウィンドウは、変更やソフトウェアのパッチなどが実行されるタイミングをコントロールする機会です。

メンテナンスの適用中は、RDS で DB クラスターのリソースの一部が使用されます。わずかながらパフォーマンスに影響が出る場合があります。DB インスタンスでは、まれに、メンテナンスによるアップデートを完了するためにマルチ AZ フェイルオーバーが必要になる場合があります。

メンテナンスイベントを特定の週に予定した場合、そのイベントはユーザーが指定した 30 分のメンテナンスウィンドウ中にスタートされます。ほとんどのメンテナンスイベントは 30 分のメンテナンスウィンドウ中に完了しますが、大規模なメンテナンスイベントは 30 分以上かかる場合があります。DB クラスターが停止すると、メンテナンスウィンドウは一時停止されます。

30 分のメンテナンスウィンドウは、リージョンごとに決められた 8 時間の中でランダムに選択されます。DB クラスターの作成時にメンテナンスウィンドウを指定しないと、RDS でランダムに選択された曜日に 30 分のメンテナンスウィンドウが割り当てられます。

以下では、各リージョンでデフォルトのメンテナンスウィンドウを割り当てる時間帯を確認できます。

リージョン名	リージョン	時間ブロック
米国東部 (オハイオ)	us-east-2	03:00 ~ 11:00 UTC
米国東部 (バージニア北部)	us-east-1	03:00 ~ 11:00 UTC
米国西部 (北カリフォルニア)	us-west-1	06:00 ~ 14:00 UTC
米国西部 (オレゴン)	us-west-2	06:00 ~ 14:00 UTC
アフリカ (ケープタウン)	af-south-1	03:00 ~ 11:00 UTC
アジアパシフィック (香港)	ap-east-1	06:00 ~ 14:00 UTC

リージョン名	リージョン	時間ブロック
アジアパシフィック (ハイデラバード)	ap-south-2	06:30 ~ 14:30 UTC
アジアパシフィック (ジャカルタ)	ap-southeast-3	08:00 ~ 16:00 UTC
アジアパシフィック (メルボルン)	ap-southeast-4	11:00 ~ 19:00 UTC
アジアパシフィック (ムンバイ)	ap-south-1	06:00 ~ 14:00 UTC
アジアパシフィック (大阪)	ap-northeast-3	22:00 ~ 06:00 UTC
アジアパシフィック (ソウル)	ap-northeast-2	13:00 ~ 21:00 UTC
アジアパシフィック (シンガポール)	ap-southeast-1	14:00 ~ 22:00 UTC
アジアパシフィック (シドニー)	ap-southeast-2	12:00 ~ 20:00 UTC
アジアパシフィック (東京)	ap-northeast-1	13:00 ~ 21:00 UTC
カナダ (中部)	ca-central-1	03:00 ~ 11:00 UTC
カナダ西部 (カルガリー)	ca-west-1	18:00 ~ 02:00 UTC
中国 (北京)	cn-north-1	06:00 ~ 14:00 UTC
中国 (寧夏)	cn-northwest-1	06:00 ~ 14:00 UTC
欧州 (フランクフルト)	eu-central-1	13:00 ~ 21:00 UTC

リージョン名	リージョン	時間ブロック
欧州 (アイルランド)	eu-west-1	22:00 ~ 06:00 UTC
欧州 (ロンドン)	eu-west-2	22:00 ~ 06:00 UTC
欧州 (ミラノ)	eu-south-1	13:00 ~ 21:00 UTC
欧州 (パリ)	eu-west-3	23:00 ~ 07:00 UTC
欧州 (スペイン)	eu-south-2	13:00 ~ 21:00 UTC
欧州 (ストックホルム)	eu-north-1	23:00 ~ 07:00 UTC
欧州 (チューリッヒ)	eu-central-2	13:00 ~ 21:00 UTC
イスラエル (テルアビブ)	il-central-1	03:00 ~ 11:00 UTC
中東 (バーレーン)	me-south-1	06:00 ~ 14:00 UTC
中東 (アラブ首長国連邦)	me-central-1	05:00 ~ 13:00 UTC
南米 (サンパウロ)	sa-east-1	13:00 ~ 21:00 UTC
AWS GovCloud (米国東部)	us-gov-east-1	01:00 ~ 09:00 UTC
AWS GovCloud (米国西部)	us-gov-west-1	06:00 ~ 14:00 UTC

DB クラスターの適切なメンテナンスウィンドウの調整

Aurora DB クラスターのメンテナンスウィンドウは、使用率の最も低い時間帯に設定する必要があるため、状況に応じて時間の変更が必要になる場合があります。適用するアップデートのために機能停止が必要な場合は、このメンテナンスウィンドウ中、DB クラスターは使用できなくなります。機能停止は、アップデートのための最小所要時間とします。

コンソール

DB クラスターの適切なメンテナンスウィンドウを調整するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. メンテナンスウィンドウを変更する DB クラスターを選択します。
4. Modify を選択します。
5. [メンテナンス] セクションで、メンテナンスウィンドウをアップデートします。
6. Continue (続行) をクリックします。

   確認ページで、変更内容を確認します。
7. 変更をメンテナンスウィンドウにすぐに適用するには、[変更のスケジューリング] セクションで [今すぐ] を選択します。
8. [クラスターの変更] を選択して、変更を保存します。

   または、[戻る] を選択して変更を編集するか、[キャンセル] を選択して変更をキャンセルします。

AWS CLI

DB クラスターに設定するメンテナンスウィンドウを調整するには、以下のパラメータを指定して AWS CLI の `modify-db-cluster` コマンドを使用します

- `--db-cluster-identifier`
- `--preferred-maintenance-window`

Example

次のコード例では、メンテナンスウィンドウを火曜日の午前 4:00 から 4:30 UTC に設定します。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster \
--db-cluster-identifier my-cluster \
--preferred-maintenance-window Tue:04:00-Tue:04:30
```

Windows の場合:

```
aws rds modify-db-cluster ^
--db-cluster-identifier my-cluster ^
--preferred-maintenance-window Tue:04:00-Tue:04:30
```

RDS API

DB クラスターに設定するメンテナンスウィンドウを調整するには、以下のパラメータで Amazon RDS の `ModifyDBCluster` API オペレーションを使用します。

- `DBClusterIdentifier`
- `PreferredMaintenanceWindow`

Aurora DB クラスターのマイナーバージョン自動アップグレード

[自動マイナーバージョンアップグレード] 設定では、Aurora で DB クラスターにアップグレードを自動的に適用するかどうかを指定します。これらのアップグレードには、追加機能とバグ修正などのパッチを含む新しいマイナーバージョンが含まれます。

この設定は、デフォルトでオンになっています。新しい DB クラスターごとに、この設定に適切な値を選択します。この値は、その重要性、予想される有効期間、各アップグレード後に行う検証テストの量に基づいています。

[自動マイナーバージョンアップグレード] 設定をオンまたはオフにする手順については、以下を参照してください。

- Aurora DB クラスターの自動マイナーバージョンアップグレードを有効にする
- Aurora DB クラスター内の個々の DB インスタンスの自動マイナーバージョンアップグレードを有効にする

> **Important**
>
> 新規および既存の DB クラスターでは、この設定はクラスター内の DB インスタンスに個別に適用するのではなく、DB クラスターに適用することを強くお勧めします。この設定がオフになっている DB インスタンスがクラスター内にある場合、DB クラスターは自動アップグレードされません。

次の表は、[自動マイナーバージョンアップグレード] 設定をクラスターレベルとインスタンスレベルで適用した場合にどのように機能するかを示しています。

アクション	クラスター設定	インスタンス設定	クラスターは自動的にアップグレードされたか。
DB クラスターで True に設定する。	True	新規および既存のすべてのインスタンスで True	はい
DB クラスターで False に設定する。	False	新規および既存のすべてのインスタンスで False	いいえ
DB クラスターでは以前は True に設定されていた。 少なくとも 1 つの DB インスタンスで False に設定する。	False への変更	1 つ以上のインスタンスでは False	いいえ
DB クラスターでは以前は False に設定されていた。 すべてのインスタンスではなく、少なくとも 1 つの DB インスタンスで True に設定する。	False	1 つ以上のインスタンスで True (すべてのインスタンスではない)	いいえ
DB クラスターでは以前は False に設定されていた。	True への変更	すべてのインスタンスで True	はい

アクション	クラスター設定	インスタンス設定	クラスターは自動的にアップグレードされたか。
すべての DB インスタンスで True に設定する。			

マイナーバージョンの自動アップグレードは、カテゴリが `maintenance` で ID が `RDS-EVENT-0156` の Amazon RDS DB クラスターイベントを通じて事前に通知されます。詳細については、「Amazon RDS のイベントカテゴリとイベントメッセージ」を参照してください。

自動アップグレードはメンテナンスウィンドウ中に実行されます。DB クラスター内の個々の DB インスタンスのメンテナンスウィンドウがクラスターメンテナンスウィンドウと異なる場合は、クラスターメンテナンスウィンドウが優先されます。

Aurora PostgreSQL のエンジンに関するアップデートの詳細については、「Amazon Aurora PostgreSQL の更新」を参照してください。

Aurora MySQL の [マイナーバージョン自動アップグレード] 設定の詳細については、「Aurora MySQL マイナーバージョン間の自動アップグレードの有効化」を参照してください。Aurora MySQL のエンジンのアップデートに関する一般的な情報については、「Amazon Aurora MySQL のデータベースエンジンの更新」を参照してください。

Aurora DB クラスターの自動マイナーバージョンアップグレードを有効にする

コンソール、CLI、API を使用した DB クラスターの変更 の一般的な手順に従います。

コンソール

[DB クラスターの変更] ページの [メンテナンス] セクションで、[マイナーバージョン自動アップグレードの有効化] チェックボックスを選択します。

AWS CLI

modify-db-cluster AWS CLI コマンドを呼び出します。`--db-cluster-identifier` オプションで DB クラスターの名前を指定し、`--auto-minor-version-upgrade` オプションで `true` を指定します。必要に応じて、この設定を DB クラスターですぐに有効にするための `--apply-immediately` オプションを指定します。

RDS API

ModifyDBCluster API オペレーションを呼び出して、DBClusterIdentifier パラメータの DB クラスター名、AutoMinorVersionUpgrade パラメータの true を指定します。必要に応じて、ApplyImmediately パラメータを true に設定し、この設定を DB クラスターですぐに有効にします。

Aurora DB クラスター内の個々の DB インスタンスの自動マイナーバージョンアップグレードを有効にする

DB クラスター内の DB インスタンスの変更 の一般的な手順に従います。

コンソール

[DB インスタンスの変更] ページの [メンテナンス] セクションで、[マイナーバージョン自動アップグレードの有効化] チェックボックスを選択します。

AWS CLI

modify-db-instance AWS CLI コマンドを呼び出します。--db-instance-identifier オプションで DB インスタンスの名前を指定し、true オプションで --auto-minor-version-upgrade を指定します。必要に応じて、この設定を DB インスタンスですぐに有効にするための --apply-immediately オプションを指定します。クラスター内の DB インスタンスごとに modify-db-instance コマンドを別個に実行します。

RDS API

ModifyDBInstance API オペレーションを呼び出して、DBInstanceIdentifier パラメータの DB クラスター名、AutoMinorVersionUpgrade パラメータの true を指定します。必要に応じて、ApplyImmediately パラメータを true に設定し、この設定を DB インスタンスですぐに有効にします。クラスター内の各 DB インスタンスで ModifyDBInstance オペレーションを個別に呼び出します。

次のような CLI コマンドを使用すると、Aurora MySQL クラスター内のすべての DB インスタンスで AutoMinorVersionUpgrade 設定のステータスを確認できます。

```
aws rds describe-db-instances \
  --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVer
```

このコマンドでは、次のような出力が生成されます。

```
[
  {
      "DBInstanceIdentifier": "db-writer-instance",
      "DBClusterIdentifier": "my-db-cluster-57",
      "AutoMinorVersionUpgrade": true
  },
  {
      "DBInstanceIdentifier": "db-reader-instance1",
      "DBClusterIdentifier": "my-db-cluster-57",
      "AutoMinorVersionUpgrade": false
  },
  {
      "DBInstanceIdentifier": "db-writer-instance2",
      "DBClusterIdentifier": "my-db-cluster-80",
      "AutoMinorVersionUpgrade": true
  },
... output omitted ...
```

この例では、DB クラスター `my-db-cluster-57` の [マイナーバージョン自動アップグレードの有効化] がオフになっています。これは、クラスター内の DB インスタンスの 1 つでオフになっているためです。

Aurora MySQL メンテナンスのアップデート頻度を選択する

各 DB クラスターごとに、Aurora MySQL のアップグレードを頻繁に行うか、またはほとんど行わないかを制御できます。最善の選択肢は、Aurora MySQL の使用状況と、Aurora 上で実行しているアプリケーションの優先順位によります。アップグレード頻度の低い Aurora MySQL 長期安定版 (LTS) リリースについては、Aurora MySQL 長期サポート (LTS) リリースを参照してください。

以下の条件の一部またはすべてが適用される場合、Aurora MySQL のクラスターのアップグレードを頻繁にしなくてもよい場合があります。

- アプリケーションのテストサイクルは、Aurora MySQL データベースエンジンのアップデートごとに時間がかかります。
- DB クラスターや同じ Aurora MySQL のバージョンで実行するアプリケーションがありますが、すべての DB クラスターと関連するアプリケーションは同時にアップグレードすることが可能です。
- Aurora MySQL と RDS for MySQL の両方を使用しています。MySQL と同じレベルで互換性のある Aurora MySQL クラスターおよび RDS MySQL DB インスタンスを保持します。

- Aurora MySQL アプリケーションが本番環境にあるか、業務上必要なものです。重要なパッチのまれな発生以外のアップグレードのために、ダウンタイムに対応する余裕はありません。
- Aurora MySQL アプリケーションは、Aurora MySQL 以降のバージョンで対応する性能の問題または機能のギャップに限られません。

上記の要因が自分の状況に当てはまる場合、Aurora MySQL DB クラスターの強制アップグレード回数を制限することが可能です。DB クラスターを作成およびアップグレードする際に、これを実行するには、「長期サポート (LTS)」という特定の Aurora MySQL のバージョンを選択してください。この場合、アップグレードのサイクル、テストサイクルそしてその DB クラスターに対するアップグレードが要因の停止時間を最小限に抑えます。

以下の条件の一部またはすべてに該当する場合、Aurora MySQL のクラスターを頻繁にアップグレードすることを選択できます。

- アプリケーションのテストサイクルは、簡単かつ手短なものでかまいません。
- アプリケーションは、まだ開発段階です。
- データベース環境では、Aurora MySQL の様々なバージョンや、Aurora MySQL および RDS for MySQL のバージョンを使用します。各 Aurora MySQL クラスターには各自のアップグレードサイクルがあります。
- Aurora MySQL の利用率を増幅する前に、特定の性能や機能の改善を待ちます。

上記の要因が自分の状況に当てはまる場合、Aurora を有効にし、より頻繁に重要なアップグレードを実施することが可能です。そのためには、Aurora MySQL DB クラスターを、LTS バージョンより新しい Aurora MySQL のバージョンに Aurora MySQL DB クラスターをアップグレードします。これを実行することで、性能が最新に強化され、バグが修正され、機能をより迅速に利用できるようになります。

オペレーティングシステムアップデートの操作

Aurora MySQL および Aurora PostgreSQL DB クラスターの DB インスタンスでは、オペレーティングシステムの更新が必要になる場合があります。Amazon RDS は、データベースパフォーマンスと顧客の全体的なセキュリティ体制改善のために、OS を新しいバージョンにアップグレードします。通常、アップデートには約 10 分かかります。オペレーティングシステムのアップデートでは、DB インスタンスの DB エンジンのバージョンまたは DB インスタンスクラスは変更されません。

最初に DB クラスターのリーダー DB インスタンスを更新し、次にライター DB インスタンスを更新することをお勧めします。フェイルオーバーが発生するとダウンタイムが発生する可能性があるため、リーダーインスタンスとライターインスタンスを同時に更新することはお勧めしません。

データベースのフェイルオーバーを高速化するには、AWS JDBC ドライバーを使用することをお勧めします。詳細については、「MySQL 用の AWS JDBC ドライバー」および「PostgreSQL 用の AWS JDBC ドライバー」を参照してください。

オペレーティングシステムの更新には 2 種類あり、DB インスタンスの保留中メンテナンスアクションに表示される説明によって区別されます。

- オペレーティングシステムのディストリビューションのアップグレード — サポートされている最新のメジャーバージョンの Amazon Linux への移行に使用します。保留中のメンテナンスアクションの説明は `New Operating System upgrade is available` です。
- オペレーティングシステムパッチ — さまざまなセキュリティ修正を適用するために使用され、場合によってはデータベースのパフォーマンスを向上させるために使用されます。保留中のメンテナンスアクションの説明は `New Operating System patch is available` です。

オペレーティングシステムのアップデートは、オプションの場合も必須の場合もあります。

- オプションのアップデートは、随時適用できます。これらのアップデートはオプションですが、RDS フリートを最新の状態に保つために定期的に適用することをお勧めします。RDS は、これらのアップデートを自動的に適用しません。

  新しいオプションのオペレーティングシステムパッチが利用可能になったときに通知を受けるには、セキュリティパッチイベントカテゴリの RDS-EVENT-0230 をサブスクライブできます。RDS イベントにサブスクライブする方法については、「Amazon RDS イベント通知にサブスクライブする」を参照してください。

  > **Note**
  >
  > `RDS-EVENT-0230` は、オペレーティングシステムのディストリビューションのアップグレードに適用されません。

> **Note**
>
> RDS for SQL Server DB インスタンスについて RDS-EVENT-0230 を受け取った場合は、apply-pending-maintenance アクションを使用して OS のアップデートを適用することはできません。詳細については、「DB クラスターのアップデートを適用する」を参照してください。

- 必須のアップデートが必要であり、必須のアップデートの前に通知を送信します。通知には期日が含まれている場合があります。この期日より前にアップデートをスケジュールするように計画してください。指定した期日後、割り当てられたメンテナンスウィンドウ中に、Amazon RDS は DB インスタンスのオペレーティングシステムを最新バージョンに自動的にアップグレードします。

  オペレーティングシステム配布アップグレードは必須です。

> **Note**
>
> さまざまなコンプライアンス義務を果たすためには、すべてのオプションおよび必須のアップデートを最新の状態に保つことが必要になる場合があります。RDS によって提供されるすべてのアップデートは、メンテナンス期間中に定期的に適用することをお勧めします。

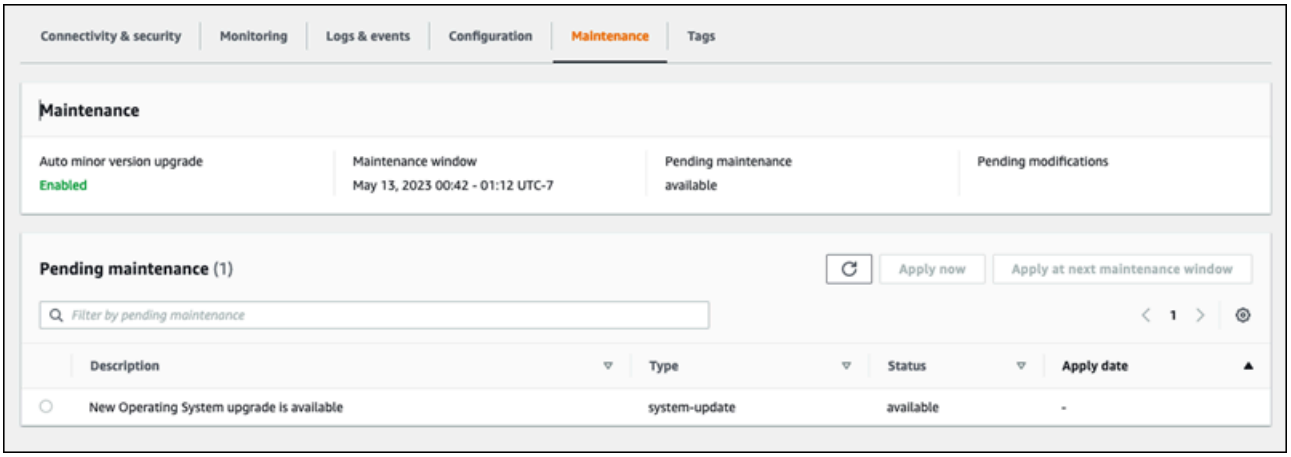
AWS Management Console または AWS CLI を使用すると、オペレーティングシステムのアップグレードの種類に関する情報を取得できます。

コンソール

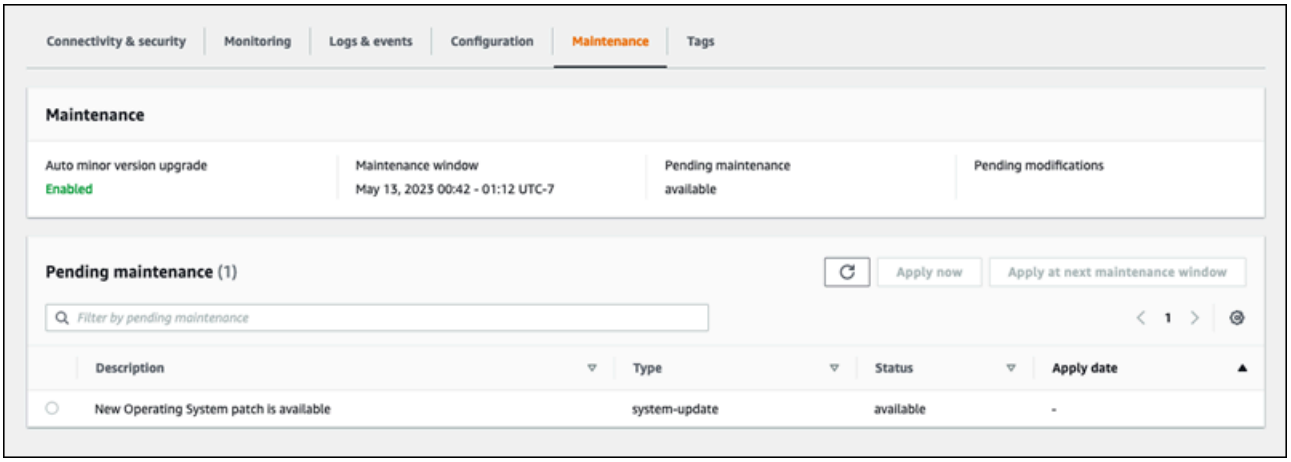
AWS Management Console を使用してアップデート情報を取得するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[データベース] を選択し、DB インスタンスを選択します。
3. [Maintenance (メンテナンスとバックアップ) を選択します。
4. [保留中のメンテナンス] セクションで、オペレーティングシステムのアップデートを検索し、[説明] の値をチェックします。

AWS Management Console では、次の図に示すように、オペレーティングシステムのディストリビューションアップグレードの [説明] が [新しいオペレーティングシステムのアップグレードが可能] に設定されています。このアップグレードは必須です。



次の図に示すように、オペレーティングシステムパッチの [説明] が [新しいオペレーティングシステムパッチが入手可能] に設定されています。



AWS CLI

AWS CLI からアップデート情報を取得するには、describe-pending-maintenance-actions コマンドを使用します。

```
aws rds describe-pending-maintenance-actions
```

次の出力は、オペレーティングシステム配布のアップグレードを示しています。

```
{
  "ResourceIdentifier": "arn:aws:rds:us-east-1:123456789012:db:mydb1",
  "PendingMaintenanceActionDetails": [
```

```
    {
      "Action": "system-update",
      "Description": "New Operating System upgrade is available"
    }
  ]
}
```

次の出力は、オペレーティングシステムパッチを示しています。

```
{
  "ResourceIdentifier": "arn:aws:rds:us-east-1:123456789012:db:mydb2",
  "PendingMaintenanceActionDetails": [
    {
      "Action": "system-update",
      "Description": "New Operating System patch is available"
    }
  ]
}
```

オペレーティングシステムのアップデートの可用性

オペレーティングシステムのアップデートは、DB エンジンのバージョンと DB インスタンスクラスに固有です。したがって、DB インスタンスは、異なる時間にアップデートを受信または要求します。そのエンジンのバージョンとインスタンスクラスに基づいた DB インスタンスにオペレーティングシステムのアップデートがある場合は、アップデートがコンソールに表示されます。AWS CLI describe-pending-maintenance-actions コマンドを実行するか、または RDS DescribePendingMaintenanceActions API オペレーションを呼び出すことによっても表示できます。インスタンスでアップデートが利用可能である場合、DB クラスターのアップデートを適用する の手順に従って OS をアップデートできます。

Amazon Aurora DB クラスターまたは Amazon Aurora DB インスタンスの再起動

通常はメンテナンス上の理由から、DB クラスターまたはクラスター内の一部のインスタンスを再起動する必要がある場合があります。例えば、パラメータグループ内のパラメータを変更したり、別のパラメータグループをクラスターに関連付けるとします。このような場合、変更を有効にするには、クラスターを再起動する必要があります。同様に、クラスター内の 1 つ以上のリーダー DB インスタンスを再起動することもできます。クラスター全体のダウンタイムを最小限に抑えるために、個々のインスタンスの再起動オペレーションを調整できます。

クラスター内の各 DB インスタンスを再起動するために必要な時間は、再起動時のデータベースアクティビティによって異なります。また、特定の DB エンジンの復旧プロセスにも依拠します。実用的であれば、再起動プロセスを開始する前に、その特定のインスタンスのデータベースアクティビティを減らしてください。そうすることで、データベースの再起動に必要な時間を短縮できます。

クラスター内の各 DB インスタンスは、使用可能な状態にある場合にのみ再起動できます。DB インスタンスは、いくつかの理由で利用できない場合があります。これには、停止状態にあるクラスター、インスタンスに適用される変更、バージョンアップグレードなどのメンテナンスウィンドウアクションが含まれます。

DB インスタンスを再起動すると、データベースエンジンプロセスが再起動されます。DB インスタンスを再起動すると一時的に機能停止になります。その間、DB インスタンスのステータスは [rebooting] に設定されます。

> **Note**
>
> DB インスタンスが、その関連付けられた DB パラメータグループに対する最新の変更を使用していない場合、AWS Management Console は、DB パラメータグループのステータスを [再起動の保留中] と表示します。パラメータグループの [再起動の保留中] のステータスにより、次回のメンテナンスウィンドウで自動的に再起動されることはありません。パラメータの最新の変更を DB インスタンスに適用するには、DB インスタンスを手動で再起動します。パラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。

トピック

- 読み取り可用性機能のない Aurora クラスターの再起動
- Aurora クラスターとインスタンスの稼働時間のチェック
- Aurora 再起動オペレーションの例

Aurora クラスター内の DB インスタンスの再起動

この手順は、Aurora で再起動を実行するときに実行する最も重要なオペレーションです。メンテナンス手順の多くでは、特定の順序で 1 つ以上の Aurora DB インスタンスの再起動が関係します。

コンソール

DB インスタンスを再起動するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択し、再起動する DB インスタンスを選択します。
3. [アクション] で、[再起動] を選択します。

   [Reboot DB Instance] ページが表示されます。
4. [Reboot] を選択して DB インスタンスを再起動します。

   または、[Cancel] (キャンセル) を選択します。

AWS CLI

AWS CLI を使用して DB インスタンスを再起動するには、`reboot-db-instance` コマンドを呼び出します。

Example

Linux、macOS、Unix の場合:

```
aws rds reboot-db-instance \
    --db-instance-identifier mydbinstance
```

Windows の場合:

```
aws rds reboot-db-instance ^
```

```
    --db-instance-identifier mydbinstance
```

RDS API

Amazon RDS API を使用して DB インスタンスを再起動するには、RebootDBInstance オペレーションを呼び出します。

読み取り可用性機能のある Aurora クラスターの再起動

読み取り可用性機能があると、プライマリまたはセカンダリクラスター内のリーダーインスタンスを再起動せずに Aurora クラスターのライターインスタンスを再起動できます。そうすることで、ライターインスタンスを再起動している間、読み込みオペレーション用にクラスターの高可用性を維持するのに役立ちます。リーダーインスタンスは後で都合の良いスケジュールで再起動できます。例えば、本番クラスターの場合、リーダーインスタンスを一度に 1 つずつ再起動し、プライマリインスタンスの再起動が完了した後にのみ開始できます。再起動する各 DB インスタンスについて、「Aurora クラスター内の DB インスタンスの再起動」 の手順に従います。

プライマリ DB クラスターの読み取り可用性機能は Aurora MySQL バージョン 2.10 以降で使用できます。セカンダリ DB クラスターの読み取り可用性は Aurora MySQL バージョン 3.06 以降で使用できます。

Aurora PostgreSQL では、この関数は次のバージョンでデフォルトで使用できます。

- バージョン 15 の 15.2 以降
- バージョン 14 の 14.7 以降
- バージョン 13 の 13.10 以降
- バージョン 12 の 12.14 以降

Aurora PostgreSQL の読み取り可用性機能の詳細については、「Aurora レプリカの読み取り可用性の向上」を参照してください。

この機能の前では、プライマリインスタンスを再起動すると、各リーダーインスタンスの再起動が同時に発生していました。Aurora クラスターで古いバージョンを実行している場合は、代わりに 読み取り可用性機能のない Aurora クラスターの再起動 の再起動の手順を使用します。

> **Note**
>
> 読み取り可用性のある Aurora DB クラスターの再起動動作の変更は、3.06 以前のバージョンの Aurora MySQL の Aurora グローバルデータベースでは異なります。Aurora グローバル

データベース内のプライマリクラスターのライターインスタンスを再起動しても、プライマリクラスター内のリーダーインスタンスは引き続き使用できます。ただし、セカンダリクラスター内の DB インスタンスは同時に再起動します。
読み取り可用性機能の向上した限定バージョンは、Aurora PostgreSQL バージョン 12.16、13.12、14.9、15.4 以降の、Aurora グローバルデータベースでサポートされています。

クラスターパラメータグループを変更した後、頻繁にクラスターを再起動します。パラメータを変更するには、「「パラメータグループを使用する」」 の手順に従います。Aurora クラスター内のライター DB インスタンスを再起動して、クラスターパラメータに変更を適用するとします。一部またはすべてのリーダー DB インスタンスは、古いパラメータ設定を引き続き使用することがあります。ただし、異なるパラメータ設定は、クラスターのデータ整合性には影響しません。データファイルの編成に影響するクラスターパラメータは、ライター DB インスタンスによってのみ使用されます。

例えば、Aurora MySQL クラスターで、リーダーインスタンスの前に、ライターインスタンスで `binlog_format` や `innodb_purge_threads` などのクラスターパラメータを更新できます。ライターインスタンスだけがバイナリログを書き込み、元に戻すレコードを消去しています。クエリの SQL ステートメントまたはクエリ出力の解釈方法を変更するパラメータについては、リーダーインスタンスを直ちに再起動するように注意する必要がある場合があります。これは、クエリ中の予期しないアプリケーションの動作を回避するために行います。例えば、`lower_case_table_names` パラメータを変更してライターインスタンスを再起動するとします。この場合、リーダーインスタンスは、すべてが再起動されるまで、新しく作成されたテーブルにアクセスできない場合があります。

すべての Aurora MySQL クラスタパラメータのリストについては、クラスターレベルのパラメータを参照してください。

すべての Aurora PostgreSQL クラスターパラメータのリストについては、Aurora PostgreSQL クラスターレベルのパラメータ を参照してください。

Tip

Aurora MySQL は、クラスターが高スループットでワークロードを処理している場合、ライターインスタンスとともに一部のリーダーインスタンスを再起動することがあります。
フェイルオーバー操作中に、再起動回数が減少することもあります。Aurora MySQL は、フェイルオーバー中にはライター DB インスタンスとフェイルオーバーターゲットのみを再起動します。クラスター内の他のリーダー DB インスタンスは、リーダーエンドポイントへの接続を通じてクエリの処理を続行するために引き続き使用できます。したがって、クラス

ター内に複数のリーダー DB インスタンスを持つことで、フェイルオーバー中の可用性を向上させることができます。

読み取り可用性機能のない Aurora クラスターの再起動

読み取り可用性機能がない場合、Aurora DB クラスター全体を再起動するには、そのクラスターのライター DB インスタンスを再起動します。これを行うには、「Aurora クラスター内の DB インスタンスの再起動」の手順に従います。

ライター DB インスタンスを再起動すると、クラスター内の各リーダー DB インスタンスの再起動も開始されます。これにより、クラスター全体のパラメータの変更がすべての DB インスタンスに同時に適用されます。ただし、すべての DB インスタンスの再起動により、クラスターのための短時間の停止が発生します。リーダー DB インスタンスは、ライター DB インスタンスの再起動が完了して利用可能になるまで利用できません。

この再起動の動作は、Aurora MySQL バージョン 2.9 以前で作成したすべての DB クラスターに適用されます。

Aurora PostgreSQL の場合、この動作は次のバージョンに適用されます。

- 14.6 以前の 14 バージョン
- 13.9 以前の 13 バージョン
- 12.13 以前の 12 バージョン
- すべての PostgreSQL 11 バージョン

RDS コンソールでは、ライター DB インスタンスは、[Databases] (データベース) ページの [Role] (ロール) 列に [Writer] (ライター) という値を有しています。RDS CLI では、`describe-db-clusters` コマンドの出力にセクション `DBClusterMembers` が含まれます。ライター DB インスタンスを表す `DBClusterMembers` エレメントには、`true` フィールドの `IsClusterWriter` の値があります。

> **Important**
>
> 読み取り可用性機能がある場合、Aurora MySQL および Aurora PostgreSQL での再起動の動作は異なります。通常、ライターインスタンスを再起動する間、リーダー DB インスタンスは引き続き使用できます。その後、都合の良いタイミングでリーダーインスタンスを再起動

できます。一部のリーダーインスタンスを常に使用できるようにする場合は、リーダーインスタンスを相互に重ならないスケジュールで再起動できます。詳細については、「読み取り可用性機能のある Aurora クラスターの再起動」を参照してください。

Aurora クラスターとインスタンスの稼働時間のチェック

Aurora クラスター内の各 DB インスタンスについて、最後の再起動からの時間の長さを確認してモニタリングできます。Amazon CloudWatch メトリクス `EngineUptime` は、DB インスタンスが最後に起動されてからの秒数をレポートします。このメトリクスをある時点で調べて、DB インスタンスの稼働時間を調べることができます。また、このメトリクスを経時的にモニタリングして、インスタンスが再起動されるタイミングを検出することもできます。

クラスターレベルで `EngineUptime` メトリクスを調べることもできます。`Minimum` および `Maximum` ディメンションは、クラスター内のすべての DB インスタンスの稼働時間の最小値および最大値をレポートします。クラスター内のリーダーインスタンスが再起動された最近の時期、または別の理由で再起動された最近の時期を確認するには、`Minimum` ディメンションを使用してクラスターレベルのメトリクスをモニタリングします。再起動せずにクラスター内のどのインスタンスが最長になったかを確認するには、`Maximum` ディメンションを使用してクラスターレベルのメトリクスをモニタリングします。例えば、設定の変更後に、クラスター内のすべての DB インスタンスが再起動されたことを確認したい場合があります。

> **Tip**
>
> 長期モニタリングでは、クラスターレベルではなく、個々のインスタンスの `EngineUptime` メトリクスをモニタリングすることをお勧めします。クラスターレベルの `EngineUptime` メトリクスは、新しい DB インスタンスがクラスターに追加されたときにゼロに設定されます。このようなクラスターの変更は、Auto Scaling で実行されるようなメンテナンスおよびスケーリングオペレーションの一環として生じる可能性があります。

次の CLI の例は、クラスター内のライターインスタンスとリーダーインスタンスの `EngineUptime` メトリクスを調べる方法を示しています。この例では、`tpch100g` という名前のクラスターを使用しています。このクラスターにはライター DB インスタンス `instance-1234` があります。また、`instance-7448` と `instance-6305` の 2 つのリーダー DB インスタンスがあります。

まず、`reboot-db-instance` コマンドはリーダーインスタンスの 1 つを再起動します。`wait` コマンドは、インスタンスの再起動が完了するまで待機します。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6305
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-6305",
        "DBInstanceStatus": "rebooting",
...
$ aws rds wait db-instance-available --db-instance-id instance-6305
```

CloudWatch get-metric-statistics コマンドは、直近 5 分間の EngineUptime メトリクスを 1 分間隔で調べます。instance-6305 インスタンスの稼働時間はゼロにリセットされ、再びカウントを開始します。Linux のこの AWS CLI の例では、$( ) 変数置換を使用して CLI コマンドに適切なタイムスタンプを挿入します。また、Linux sort コマンドを使用して、メトリクスが収集された時間で出力を順序付けます。そのタイムスタンプ値は、出力の各行の 3 番目のフィールドです。

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBInstanceIdentifier,Value=instance-6305 --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 231.0 2021-03-16T18:19:00+00:00 Seconds
DATAPOINTS 291.0 2021-03-16T18:20:00+00:00 Seconds
DATAPOINTS 351.0 2021-03-16T18:21:00+00:00 Seconds
DATAPOINTS 411.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 471.0 2021-03-16T18:23:00+00:00 Seconds
```

クラスター内のインスタンスの 1 つが再起動されたため、クラスターの最小稼働時間はゼロにリセットされます。クラスター内の少なくとも 1 つの DB インスタンスが利用可能なままであるため、クラスターの最長稼働時間はリセットされません。

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Minimum \
  --dimensions Name=DBClusterIdentifier,Value=tpch100g --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 63099.0 2021-03-16T18:12:00+00:00 Seconds
DATAPOINTS 63159.0 2021-03-16T18:13:00+00:00 Seconds
DATAPOINTS 63219.0 2021-03-16T18:14:00+00:00 Seconds
DATAPOINTS 63279.0 2021-03-16T18:15:00+00:00 Seconds
DATAPOINTS 51.0 2021-03-16T18:16:00+00:00 Seconds
```

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBClusterIdentifier,Value=tpch100g --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 63389.0 2021-03-16T18:16:00+00:00 Seconds
DATAPOINTS 63449.0 2021-03-16T18:17:00+00:00 Seconds
DATAPOINTS 63509.0 2021-03-16T18:18:00+00:00 Seconds
DATAPOINTS 63569.0 2021-03-16T18:19:00+00:00 Seconds
DATAPOINTS 63629.0 2021-03-16T18:20:00+00:00 Seconds
```

その後、別の reboot-db-instance コマンドがクラスターのライターインスタンスを再起動します。別の wait コマンドは、ライターインスタンスの再起動が終了するまで一時停止します。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-1234
{
  "DBInstanceIdentifier": "instance-1234",
  "DBInstanceStatus": "rebooting",
...
$ aws rds wait db-instance-available --db-instance-id instance-1234
```

これで、ライターインスタンスの EngineUptime メトリクスに、インスタンス instance-1234 が最近再起動されたことが示されます。リーダーインスタンス instance-6305 も、ライターインスタンスとともに自動的に再起動されました。このクラスターは Aurora MySQL 2.09 を実行しており、ライターインスタンスが再起動してもリーダーインスタンスの実行は維持されません。

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBInstanceIdentifier,Value=instance-1234 --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 63749.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 63809.0 2021-03-16T18:23:00+00:00 Seconds
DATAPOINTS 63869.0 2021-03-16T18:24:00+00:00 Seconds
DATAPOINTS 41.0 2021-03-16T18:25:00+00:00 Seconds
DATAPOINTS 101.0 2021-03-16T18:26:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
```

```
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBInstanceIdentifier,Value=instance-6305 --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 411.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 471.0 2021-03-16T18:23:00+00:00 Seconds
DATAPOINTS 531.0 2021-03-16T18:24:00+00:00 Seconds
DATAPOINTS 49.0 2021-03-16T18:26:00+00:00 Seconds
```

Aurora 再起動オペレーションの例

次の Aurora MySQL の例は、Aurora DB クラスター内のリーダーとライター DB インスタンスの再起動オペレーションのさまざまな組み合わせを示しています。再起動するたびに、SQL クエリはクラスター内のインスタンスの稼働時間を示します。

トピック

- Aurora クラスターのライターインスタンスとリーダーインスタンスの検索
- 1 つのリーダーインスタンスの再起動
- ライターインスタンスの再起動
- ライターとリーダーの独立した再起動
- Aurora MySQL バージョン 2.10 クラスターへのクラスターパラメータの変更の適用

Aurora クラスターのライターインスタンスとリーダーインスタンスの検索

複数の DB インスタンスを持つ Aurora MySQL クラスターでは、どれがライターで、どれがリーダーであるかを知ることが重要です。ライターインスタンスとリーダーインスタンスは、フェイルオーバーオペレーションが発生したときにロールを切り替えることができます。したがって、ライターまたはリーダーのインスタンスを必要とするオペレーションを実行する前に、次のようなチェックを実行することをお勧めします。この場合、`False` の `IsClusterWriter` の値は、リーダーインスタンス、`instance-6305`、および `instance-7448` を識別します。`True` の値は、ライターインスタンス `instance-1234` を識別します。

```
$ aws rds describe-db-clusters --db-cluster-id tpch100g \
  --query "*[].['Cluster:',DBClusterIdentifier,DBClusterMembers[*].
['Instance:',DBInstanceIdentifier,IsClusterWriter]]" \
  --output text
Cluster:     tpch100g
```

```
Instance:    instance-6305    False
Instance:    instance-7448    False
Instance:    instance-1234    True
```

再起動の例を開始する前に、ライターインスタンスの稼働時間は約 1 週間です。この例の SQL クエリは、MySQL 固有の稼働時間をチェックする方法を示しています。この手法は、データベースアプリケーションで使用する場合があります。AWS CLI を使用し、両方の Aurora エンジン用に機能する別のテクニックについては、Aurora クラスターとインスタンスの稼働時間のチェック を参照してください。

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status
    -> where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-08 17:49:06.000000 | 174h 42m|
+----------------------------+---------+
```

1 つのリーダーインスタンスの再起動

この例では、リーダーの DB インスタンスの 1 つを再起動します。おそらく、このインスタンスは、大きなクエリまたは多数の同時実行接続によってオーバーロードされた可能性があります。あるいは、ネットワークの問題が原因で、ライターインスタンスに遅れた可能性があります。再起動オペレーションを開始した後、例では、インスタンスが使用可能になるまで一時停止する `wait` コマンドを使用します。その時点まで、インスタンスの稼働時間は数分です。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6305
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-6305",
        "DBInstanceStatus": "rebooting",
...
    }
}
$ aws rds wait db-instance-available --db-instance-id instance-6305
$ mysql -h instance-6305.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
```

```
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status
    -> where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:35:02.000000 | 00h 03m |
+----------------------------+---------+
```

リーダーインスタンスを再起動しても、ライターインスタンスの稼働時間には影響しませんでした。まだ約 1 週間の稼働時間があります。

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+----------+
| Last Startup               | Uptime   |
+----------------------------+----------+
| 2021-03-08 17:49:06.000000 | 174h 49m |
+----------------------------+----------+
```

ライターインスタンスの再起動

この例では、ライターインスタンスを再起動します。このクラスターは Aurora MySQL バージョン 2.09 を実行しています。Aurora MySQL バージョンが 2.10 より低いため、ライターインスタンスを再起動すると、クラスター内のリーダーインスタンスも再起動されます。

`wait` コマンドは再起動が完了するまで一時停止します。これで、そのインスタンスの稼働時間はゼロにリセットされます。ライター DB インスタンスとリーダー DB インスタンスでは、再起動オペレーションにかかる時間が大幅に異なる場合があります。ライターおよびリーダー DB インスタンスは、ロールに応じて異なる種類のクリーンアップオペレーションを実行します。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-1234
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-1234",
        "DBInstanceStatus": "rebooting",
...
```

```
    }
}
$ aws rds wait db-instance-available --db-instance-id instance-1234
$ mysql -h instance-1234.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:40:27.000000 | 00h 00m |
+----------------------------+---------+
```

ライター DB インスタンスの再起動後、両方のリーダー DB インスタンスの稼働時間がリセットされます。ライターインスタンスを再起動すると、リーダーインスタンスも再起動しました。この動作は、Aurora PostgreSQL クラスターおよびバージョン 2.10 より前の Aurora MySQL クラスターに適用されます。

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:40:35.000000 | 00h 00m |
+----------------------------+---------+

$ mysql -h instance-6305.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:40:33.000000 | 00h 01m |
+----------------------------+---------+
```

ライターとリーダーの独立した再起動

次の例は、Aurora MySQL バージョン 2.10 を実行するクラスターを示しています。この Aurora MySQL バージョン以降では、すべてのリーダーインスタンスの再起動を行わずにライターインスタンスを再起動できます。これにより、ライターインスタンスを再起動しても、クエリを大量に消費するアプリケーションが停止することはありません。リーダーインスタンスは後で再起動できます。これらの再起動は、クエリトラフィックが少ない時に実行できます。リーダーインスタンスを一度に 1 つずつ再起動することもできます。これにより、アプリケーションのクエリアントラフィックのために、少なくとも 1 つのリーダーインスタンスが常に利用可能になります。

次の例では、Aurora MySQL バージョン `cluster-2393` を実行している `5.7.mysql_aurora.2.10.0` という名前のクラスターを使用しています。このクラスターには、`instance-9404` という名前のライターインスタンスと、`instance-6772`、`instance-2470`、`instance-5138` という名前の 3 つのリーダーインスタンスがあります。

```
$ aws rds describe-db-clusters --db-cluster-id cluster-2393 \
  --query "*[].['Cluster:',DBClusterIdentifier,DBClusterMembers[*].
['Instance:',DBInstanceIdentifier,IsClusterWriter]]" \
  --output text
Cluster:        cluster-2393
Instance:       instance-5138        False
Instance:       instance-2470        False
Instance:       instance-6772        False
Instance:       instance-9404        True
```

`uptime` コマンドを使用して各データベースインスタンスの `mysql` の値をチェックすると、各データベースインスタンスの稼働時間がほぼ同じであることが表示されます。例えば、`instance-5138` の稼働時間は次のとおりです。

```
mysql> SHOW GLOBAL STATUS LIKE 'uptime';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Uptime        | 3866  |
+---------------+-------+
```

CloudWatch を使用すると、実際にインスタンスにログインしなくても、対応する稼働時間の情報を取得できます。これにより、管理者はデータベースをモニタリングできますが、テーブルデータを表

示または変更することはできません。この場合、5 分間の期間を指定し、毎分稼働時間の値をチェックします。稼働時間の値の増加は、その期間中にインスタンスが再起動されなかったことを示しています。

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-9404 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 4648.0 2021-03-17T23:42:00+00:00 Seconds
DATAPOINTS 4708.0 2021-03-17T23:43:00+00:00 Seconds
DATAPOINTS 4768.0 2021-03-17T23:44:00+00:00 Seconds
DATAPOINTS 4828.0 2021-03-17T23:45:00+00:00 Seconds
DATAPOINTS 4888.0 2021-03-17T23:46:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-6772 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 4315.0 2021-03-17T23:42:00+00:00 Seconds
DATAPOINTS 4375.0 2021-03-17T23:43:00+00:00 Seconds
DATAPOINTS 4435.0 2021-03-17T23:44:00+00:00 Seconds
DATAPOINTS 4495.0 2021-03-17T23:45:00+00:00 Seconds
DATAPOINTS 4555.0 2021-03-17T23:46:00+00:00 Seconds
```

ここで、リーダーインスタンス instance-5138 のいずれかを再起動します。再起動後、インスタンスが再び利用可能になるまで待機します。5 分間の稼働時間をモニタリングすると、その間に稼働時間がゼロにリセットされたことがわかります。最新の稼働時間の値は、再起動が完了してから 5 秒後に測定されました。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-5138

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
```

```
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-5138 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 4500.0 2021-03-17T23:46:00+00:00 Seconds
DATAPOINTS 4560.0 2021-03-17T23:47:00+00:00 Seconds
DATAPOINTS 4620.0 2021-03-17T23:48:00+00:00 Seconds
DATAPOINTS 4680.0 2021-03-17T23:49:00+00:00 Seconds
DATAPOINTS  5.0 2021-03-17T23:50:00+00:00 Seconds
```

次に、ライターインスタンス instance-9404 の再起動を実行します。ライターインスタンスとリーダーインスタンスの 1 つの稼働時間の値を比較します。これにより、ライターを再起動してもリーダーは再起動されないことがわかります。Aurora MySQL 2.10 より前のバージョンでは、すべてのリーダーの稼働時間の値はライターと同時にリセットされます。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-9404
{
  "DBInstanceIdentifier": "instance-9404",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-9404

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-9404 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 371.0 2021-03-17T23:57:00+00:00 Seconds
DATAPOINTS 431.0 2021-03-17T23:58:00+00:00 Seconds
DATAPOINTS 491.0 2021-03-17T23:59:00+00:00 Seconds
DATAPOINTS 551.0 2021-03-18T00:00:00+00:00 Seconds
DATAPOINTS  37.0  2021-03-18T00:01:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-6772 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 5215.0 2021-03-17T23:57:00+00:00 Seconds
DATAPOINTS 5275.0 2021-03-17T23:58:00+00:00 Seconds
DATAPOINTS 5335.0 2021-03-17T23:59:00+00:00 Seconds
```

```
DATAPOINTS 5395.0 2021-03-18T00:00:00+00:00 Seconds
DATAPOINTS 5455.0 2021-03-18T00:01:00+00:00 Seconds
```

すべてのリーダーインスタンスで、設定パラメータへの変更がライターインスタンスとすべて同じであることを確認するには、ライターの後にすべてのリーダーインスタンスを再起動します。次の例は、すべてのリーダーを再起動し、すべてのリーダーが使用可能になるまで待機してから続行します。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6772
{
  "DBInstanceIdentifier": "instance-6772",
  "DBInstanceStatus": "rebooting"
}

$ aws rds reboot-db-instance --db-instance-identifier instance-2470
{
  "DBInstanceIdentifier": "instance-2470",
  "DBInstanceStatus": "rebooting"
}

$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceStatus": "rebooting"
}

$ aws rds wait db-instance-available --db-instance-id instance-6772
$ aws rds wait db-instance-available --db-instance-id instance-2470
$ aws rds wait db-instance-available --db-instance-id instance-5138
```

これで、ライター DB インスタンスの稼働時間が最も長いことがわかります。このインスタンスの稼働時間の値は、モニタリング期間を通じて着実に増加しました。リーダー DB インスタンスはすべて、リーダーの後に再起動されました。各リーダーが再起動され、その稼働時間がゼロにリセットされたモニタリング期間内の時点を確認できます。

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-9404 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 457.0 2021-03-18T00:08:00+00:00 Seconds
```

```
DATAPOINTS 517.0 2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 577.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS 637.0 2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS  697.0 2021-03-18T00:12:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-2470 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 5819.0 2021-03-18T00:08:00+00:00 Seconds
DATAPOINTS  35.0  2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 95.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS 155.0 2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS 215.0 2021-03-18T00:12:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-5138 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 1085.0 2021-03-18T00:08:00+00:00 Seconds
DATAPOINTS 1145.0 2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 1205.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS  49.0  2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS 109.0 2021-03-18T00:12:00+00:00 Seconds
```

Aurora MySQL バージョン 2.10 クラスターへのクラスターパラメータの変更の適用

次の例では、Aurora MySQL 2.10 クラスター内のすべての DB インスタンスにパラメータの変更を適用する方法を示します。この Aurora MySQL バージョンでは、ライターインスタンスとすべてのリーダーインスタンスを個別に再起動します。

この例では、説明のために MySQL 設定パラメータ `lower_case_table_names` を使用します。このパラメータ設定がライターとリーダーの DB インスタンス間で異なる場合、大文字、または大文字と小文字が混在した名前で宣言されたテーブルにクエリがアクセスできないことがあります。または、2 つのテーブル名で異なるのが大文字と小文字のみである場合、クエリが間違ったテーブルにアクセスする可能性があります。

この例では、各インスタンスの IsClusterWriter 属性を調べて、クラスター内のライターインスタンスとリーダーインスタンスを特定する方法を示します。クラスターには cluster-2393 という名前が付けられています。クラスターには、instance-9404 という名前のライターインスタンスがあります。クラスター内のリーダーインスタンスには、instance-5138 と instance-2470 という名前が付けられています。

```
$ aws rds describe-db-clusters --db-cluster-id cluster-2393 \
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' \
  --output text
cluster-2393
instance-5138        False
instance-2470        False
instance-9404        True
```

lower_case_table_names パラメータの変更の影響を示すために、2 つの DB クラスターパラメータグループを設定します。lower-case-table-names-0 パラメータグループでは、このパラメータが 0 に設定されています。lower-case-table-names-1 パラメータグループでは、このパラメータグループが 1 に設定されています。

```
$ aws rds create-db-cluster-parameter-group --description 'lower-case-table-names-0' \
  --db-parameter-group-family aurora-mysql5.7 \
  --db-cluster-parameter-group-name lower-case-table-names-0
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "lower-case-table-names-0",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "lower-case-table-names-0"
    }
}

$ aws rds create-db-cluster-parameter-group --description 'lower-case-table-names-1' \
  --db-parameter-group-family aurora-mysql5.7 \
  --db-cluster-parameter-group-name lower-case-table-names-1
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "lower-case-table-names-1",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "lower-case-table-names-1"
    }
}
```

```
$ aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name lower-case-table-names-0 \
  --parameters
 ParameterName=lower_case_table_names,ParameterValue=0,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "lower-case-table-names-0"
}

$ aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name lower-case-table-names-1 \
    --parameters
 ParameterName=lower_case_table_names,ParameterValue=1,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "lower-case-table-names-1"
}
```

lower_case_table_names のデフォルト値は 0 です。このパラメータを設定すると、テーブル foo はテーブル FOO とは区別されます。この例では、パラメータが引き続きデフォルト設定になっていることを確認します。その後、名前の大文字と小文字だけが異なる 3 つのテーブルを作成します。

```
mysql> create database lctn;
Query OK, 1 row affected (0.07 sec)

mysql> use lctn;
Database changed
mysql> select @@lower_case_table_names;
+--------------------------+
| @@lower_case_table_names |
+--------------------------+
|                        0 |
+--------------------------+

mysql> create table foo (s varchar(128));
mysql> insert into foo values ('Lowercase table name foo');

mysql> create table Foo (s varchar(128));
mysql> insert into Foo values ('Mixed-case table name Foo');

mysql> create table FOO (s varchar(128));
mysql> insert into FOO values ('Uppercase table name FOO');

mysql> select * from foo;
```

```
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+---------------------------+
| s                         |
+---------------------------+
| Mixed-case table name Foo |
+---------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Uppercase table name FOO |
+--------------------------+
```

次に、DB パラメータグループをクラスターに関連付けて、lower_case_table_names パラメータを 1 に設定します。この変更は、各 DB インスタンスが再起動された後にのみ有効になります。

```
$ aws rds modify-db-cluster --db-cluster-identifier cluster-2393 \
  --db-cluster-parameter-group-name lower-case-table-names-1
{
  "DBClusterIdentifier": "cluster-2393",
  "DBClusterParameterGroup": "lower-case-table-names-1",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.10.0"
}
```

実行する最初の再起動は、ライター DB インスタンスのためのものです。その後、インスタンスが再び利用可能になるのを待ちます。その時点で、ライターエンドポイントに接続し、ライターインスタンスに変更されたパラメータ値があることを確認します。SHOW TABLES コマンドは、データベースに 3 つの異なるテーブルが含まれていることを確認します。ただし、foo、Foo、または FOO という名前のテーブルを参照するクエリはすべて、名前がすべて小文字のテーブル foo にアクセスします。

```
# Rebooting the writer instance
$ aws rds reboot-db-instance --db-instance-identifier instance-9404
```

```
$ aws rds wait db-instance-available --db-instance-id instance-9404
```

これで、クラスターエンドポイントを使用するクエリは、パラメータの変更の影響を示すようになりました。クエリ内のテーブル名が大文字、小文字、または大文字と小文字の混在のいずれであっても、SQL ステートメントは、名前がすべて小文字のテーブルにアクセスします。

```
mysql> select @@lower_case_table_names;
+--------------------------+
| @@lower_case_table_names |
+--------------------------+
|                        1 |
+--------------------------+

mysql> use lctn;
mysql> show tables;
+----------------+
| Tables_in_lctn |
+----------------+
| FOO            |
| Foo            |
| foo            |
+----------------+

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
```

```
+--------------------------+
```

次の例は、前のクエリと同じクエリを示しています。この場合、クエリはリーダーエンドポイントを使用し、リーダー DB インスタンスの 1 つで実行されます。それらのインスタンスはまだ再起動されていません。したがって、lower_case_table_names パラメータのための元の設定が残っています。つまり、クエリは、foo、Foo、および FOO の各テーブルにアクセスできます。

```
mysql> select @@lower_case_table_names;
+--------------------------+
| @@lower_case_table_names |
+--------------------------+
|                        0 |
+--------------------------+

mysql> use lctn;

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+---------------------------+
| s                         |
+---------------------------+
| Mixed-case table name Foo |
+---------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Uppercase table name FOO |
+--------------------------+
```

次に、リーダーインスタンスの 1 つを再起動し、それが再び利用可能になるまで待機します。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-2470
{
  "DBInstanceIdentifier": "instance-2470",
```

```
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-2470
```

instance-2470 のインスタンスエンドポイントに接続している間、クエリは新しいパラメータが有効であることを示します。

```
mysql> select @@lower_case_table_names;
+---------------------------+
| @@lower_case_table_names |
+---------------------------+
|                         1 |
+---------------------------+
```

この時点で、クラスター内の 2 つのリーダーインスタンスが異なる lower_case_table_names 設定で実行されています。したがって、クラスターのリーダーエンドポイントへの接続は、予測不可能なこの設定の値を使用します。もう一方のリーダーインスタンスを直ちに再起動して、両方とも一貫した設定となるようにすることが重要です。

```
$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-5138
```

次の例では、すべてのリーダーインスタンスが lower_case_table_names パラメータ用に同じ設定となっていることを確認します。コマンドは、各リーダーインスタンスの lower_case_table_names 設定値をチェックします。その後、リーダーエンドポイントを使用する同じコマンドで、リーダーエンドポイントへの各接続でリーダーインスタンスの 1 つが使用されることが示されますが、どれを使用するのかは予測できません。

```
# Check lower_case_table_names setting on each reader instance.

$ mysql -h instance-5138.a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+---------------------------+---------------------------+
| @@aurora_server_id        | @@lower_case_table_names |
+---------------------------+---------------------------+
| instance-5138             |                         1 |
```

```
+--------------------------+--------------------------+

$ mysql -h instance-2470.a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-2470            |                        1 |
+--------------------------+--------------------------+

# Check lower_case_table_names setting on the reader endpoint of the cluster.

$ mysql -h cluster-2393.cluster-ro-a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-5138            |                        1 |
+--------------------------+--------------------------+

# Run query on writer instance

$ mysql -h cluster-2393.cluster-a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-9404            |                        1 |
+--------------------------+--------------------------+
```

パラメータの変更をあらゆる場所に適用することで、lower_case_table_names=1 の設定の効果を確認できます。テーブルが foo、Foo、FOO のいずれとして参照されるかによって、クエリは名前を foo に変換し、各場合で同じテーブルにアクセスします。

```
mysql> use lctn;

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+
```

```
mysql> select * from Foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+
```

Aurora DB クラスターと DB インスタンスを削除する

Aurora DB クラスターは、不要になったときに削除できます。クラスターを削除すると、すべてのデータとともにクラスターボリュームが削除されます。クラスターを削除する前に、データをスナップショットとして保存できます。後でスナップショットを復元することで、削除したクラスターと同じデータを含む、新しいクラスターを作成できます。

クラスター自体と、そのクラスターに含まれるデータを保持しながら、クラスターから DB インスタンスのみを削除することもできます。DB インスタンスを削除すると、クラスターがビジー状態でない場合や、複数の DB インスタンスのためにコンピューティング容量を必要としていない場合の料金を削減できます。

トピック

Aurora DB クラスターの削除

Aurora には、DB クラスターを削除するための、単一的なステップによるメソッドは用意されていません。設計上のこの選択は、誤ってデータを喪失したり、アプリケーションをオフラインにしたりするのを防ぐことを目的としています。Aurora アプリケーションは一般的にミッションクリティカルであり、高い可用性を必要とします。したがって Auroraでは、DB インスタンスの追加と削除により、クラスターの容量を簡単に拡張および縮小できるようにしています。クラスター自体を削除するには、個別の削除を行う必要があります。

クラスターからすべての DB インスタンスを削除してから、クラスター自体を削除する場合は、次の一般的な手順に従います。

1. クラスター内のリーダーインスタンスをすべて削除します。Aurora DB クラスターからの DB インスタンスの削除 にある手順を実行します。

   クラスターにリーダーインスタンスがある場合、いずれかのインスタンスを削除すると、クラスターのコンピューティング容量が削減されるだけです。初期にリーダーインスタンスを削除する

と、プロシージャ全体でクラスターが使用可能なまま、不要なフェールオーバー操作は実行されなくなります。

2. クラスターからライターインスタンスを削除します。再度、Aurora DB クラスターからの DB インスタンスの削除 の手順を使用します。

   すべての DB インスタンスを削除した後も、クラスターとそれに関連付けられたクラスターボリュームはそのまま残ります。

3. DB クラスターを削除する

   - AWS Management Console — クラスターを選択し、[アクション] から [削除] を選択します。後で必要になった場合に備えて、以下のオプションを選択してクラスターのデータを保存できます。
     - クラスターボリュームの最終スナップショットを作成します。デフォルト設定では、最終スナップショットを作成します。
     - 自動バックアップの保持 デフォルト設定では、自動バックアップを保持しません。

     > **Note**
     >
     > Aurora Serverless v1 DB クラスターの自動バックアップは保持されません。

     また、Aurora はユーザーに対し、クラスターを削除することの確認も促します。
   - CLI と API — `delete-db-cluster` CLI コマンドまたは `DeleteDBCluster` API オペレーションを呼び出します。後で必要になった場合に備えて、以下のオプションを選択してクラスターのデータを保存できます。
     - クラスターボリュームの最終スナップショットを作成します。
     - 自動バックアップの保持

     > **Note**
     >
     > Aurora Serverless v1 DB クラスターの自動バックアップは保持されません。

トピック

- 空の Aurora クラスターの削除
- 単一の DB インスタンスを使用している Aurora クラスターの削除
- 複数の DB インスタンスを持つ Aurora クラスターの削除

空の Aurora クラスターの削除

AWS Management Console、AWS CLI、または Amazon RDS API を使用して、空の DB クラスターを削除できます。

> **Tip**
>
> クラスターを DB インスタンスがない状態で維持すると、クラスターの CPU 料金を発生させずにデータを保持できます。クラスターに 1 つ以上の新しい DB インスタンスを作成することで、クラスターの使用をすぐに再開できます。関連付けられた DB インスタンスがない場合でも、クラスターに対しては Aurora 固有の管理操作を実行できます。ただ、データにアクセスしたり、DB インスタンスへの接続を必要とするオペレーションを実行したりすることはできません。

コンソール

DB クラスターを削除するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択して、削除する DB クラスターを選択します。
3. [アクション] で、[削除] を選択します。
4. DB クラスターの最終 DB スナップショットを作成するには、「最終スナップショットを作成しますか」を選択します。これはデフォルトの設定です。
5. 最終スナップショットの作成を選択した場合は、[Final snapshot name (最終スナップショット名)] を入力します。
6. 自動バックアップを保持するには、[Retain automated backups (自動バックアップの保持)] を選択します。これはデフォルトの設定ではありません。
7. ボックスに「**`delete me`**」と入力します。
8. [削除] を選択します。

CLI

AWS CLI を使用して、空の Aurora DB クラスターを削除するには、delete-db-cluster コマンドを呼び出します。

空のクラスター deleteme-zero-instances は、開発およびテストにのみ使用されるもので、重要なデータを保持していないと想定できます。この場合、クラスターを削除する際に、クラスターボリュームのスナップショットを保存しておく必要はありません。次に、クラスターが DB インスタンスを含んでおらず、また最終スナップショットを作成せず、自動バックアップを保持せずに、この空のクラスターを削除する場合の例を示します。

```
$ aws rds describe-db-clusters --db-cluster-identifier deleteme-zero-instances --output
 text \
  --query '*[].["Cluster:",DBClusterIdentifier,DBClusterMembers[*].
["Instance:",DBInstanceIdentifier,IsClusterWriter]]
Cluster:        deleteme-zero-instances

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-zero-instances \
  --skip-final-snapshot \
  --delete-automated-backups
{
  "DBClusterIdentifier": "deleteme-zero-instances",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

RDS API

Amazon RDS API を使用して空の Aurora DB クラスターを削除するには、DeleteDBCluster オペレーションを呼び出します。

単一の DB インスタンスを使用している Aurora クラスターの削除

DB クラスターで削除保護が有効になっている場合でも、最後の DB インスタンスを削除できます。この場合、DB クラスター自体は存続し、データは保持されます。クラスターに新しい DB インスタンスをアタッチすることで、再度データにアクセスできます。

次に、クラスターにまだ DB インスタンスが関連付けられているために、delete-db-cluster コマンドが機能しない場合の例を示します。このクラスターには、ライター DB インスタンスが 1 つあります。クラスター内の DB インスタンスを確認するには、各インスタンスの IsClusterWriter 属性をチェックします。クラスターには、0 個または 1 個のライター DB インスタンスを保持することができます。値が true であれば、それがライター DB インスタンスであることを意味します。値が false であれば、それがリーダー DB インスタンスであることを意味します。リーダー DB インスタンスの場合は、0 もしくは 1 個以上の複数を、クラスターで保持できます。この場合、delete-db-instance コマンドを使用してライター DB インスタンスを削除しま

す。DB インスタンスが deleting 状態になれば、その時点でクラスターも削除できます。この例では、クラスターに保持する価値のあるデータが含まれていないと仮定しています。そのため、クラスターボリュームのスナップショットを作成したり、自動バックアップを保存したりしません。

```
$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-only --skip-final-
snapshot
An error occurred (InvalidDBClusterStateFault) when calling the DeleteDBCluster
 operation:
  Cluster cannot be deleted, it still contains DB instances in non-deleting state.

$ aws rds describe-db-clusters --db-cluster-identifier deleteme-writer-only \
  --query '*[].[DBClusterIdentifier,Status,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]'
[
    [
        "deleteme-writer-only",
        "available",
        [
            [
                "instance-2130",
                true
            ]
        ]
    ]
]

$ aws rds delete-db-instance --db-instance-identifier instance-2130
{
  "DBInstanceIdentifier": "instance-2130",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-only \
  --skip-final-snapshot \
  --delete-automated-backups
{
  "DBClusterIdentifier": "deleteme-writer-only",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

複数の DB インスタンスを持つ Aurora クラスターの削除

クラスターに含まれる DB インスタンスが複数ある場合、それらは通常、1 つのライターインスタンスと 1 つ以上のリーダーインスタンスで構成されます。リーダーインスタンスは、ライターインスタンスで問題が発生した場合にそれを引き継ぐためにスタンバイ状態を取ることで、高可用性を支援しています。また、リーダーインスタンスを使用して、ライターインスタンスにオーバーヘッドを追加せずに、読み取り回数が多いワークロードを処理するようにクラスターをスケールアップすることもできます。

複数のリーダー DB インスタンスを保持するクラスターを削除するには、まずリーダーインスタンスを削除した上で、ライターインスタンスを削除します。ライターインスタンスを削除しても、クラスターとそのデータは所定の位置に残ります。クラスターは別のアクションで削除します。

- Aurora DB インスタンスを削除する手順については、「Aurora DB クラスターからの DB インスタンスの削除」を参照してください。
- Aurora クラスター内のライター DB インスタンスを削除する手順については、「単一の DB インスタンスを使用している Aurora クラスターの削除」を参照してください。
- 空の Aurora クラスターを削除する手順については、「空の Aurora クラスターの削除」を参照してください。

この CLI 例では、ライター DB インスタンスに加え、1 つのリーダー DB インスタンスを含むクラスターを削除する方法を示します。`describe-db-clusters` の出力により、`instance-7384` がライターインスタンスであり、`instance-1039` リーダーインスタンスであることが確認できます。この例のような場合、リーダーインスタンスがまだ存在している間にライターインスタンスを削除すると、フェールオーバー操作が発生するため、リーダーインスタンスを初期に削除します。そのインスタンス自体も削除する予定がある場合は、リーダーインスタンスをライターに昇格させても意味がありません。また、これらの `db.t2.small` インスタンスは開発とテストにのみ使用されると仮定しているため、削除操作では最終スナップショットをスキップし、自動バックアップを保持しません。

```
$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-and-reader --skip-
final-snapshot

An error occurred (InvalidDBClusterStateFault) when calling the DeleteDBCluster
 operation:
  Cluster cannot be deleted, it still contains DB instances in non-deleting state.
```

```
$ aws rds describe-db-clusters --db-cluster-identifier deleteme-writer-and-reader --
output text \
  --query '*[].["Cluster:",DBClusterIdentifier,DBClusterMembers[*].
["Instance:",DBInstanceIdentifier,IsClusterWriter]]
Cluster:        deleteme-writer-and-reader
Instance:       instance-1039  False
Instance:       instance-7384   True

$ aws rds delete-db-instance --db-instance-identifier instance-1039
{
  "DBInstanceIdentifier": "instance-1039",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-7384
{
  "DBInstanceIdentifier": "instance-7384",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-and-reader \
  --skip-final-snapshot \
  --delete-automated-backups
{
  "DBClusterIdentifier": "deleteme-writer-and-reader",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

次の例は、ライター DB インスタンスと複数のリーダー DB インスタンスを含む DB クラスターを削除する方法を示しています。ここでは、describe-db-clustersコマンドからの簡潔な出力を使用して、ライターとリーダーインスタンスそれぞれに関するレポートを取得します。繰り返しますが、ライター DB インスタンスを削除する前に、すべてのリーダー DB インスタンスを削除する必要があります。リーダー DB インスタンスを削除する順序は任意です。

複数の DB インスタンスを持つこのクラスターには、保存する価値のあるデータが含まれていると考えられます。したがって、作成するスナップショットの詳細を設定するために、この例の delete-db-cluster コマンドには --no-skip-final-snapshot パラメータと --final-db-snapshot-identifier パラメータが含まれています。また、自動バックアップを保持するための --no-delete-automated-backups パラメータも含まれています。

```
$ aws rds describe-db-clusters --db-cluster-identifier deleteme-multiple-readers --
output text \
  --query '*[].["Cluster:",DBClusterIdentifier,DBClusterMembers[*].
["Instance:",DBInstanceIdentifier,IsClusterWriter]]
Cluster:        deleteme-multiple-readers
Instance:       instance-1010   False
Instance:       instance-5410   False
Instance:       instance-9948   False
Instance:       instance-8451   True

$ aws rds delete-db-instance --db-instance-identifier instance-1010
{
  "DBInstanceIdentifier": "instance-1010",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-5410
{
  "DBInstanceIdentifier": "instance-5410",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-9948
{
  "DBInstanceIdentifier": "instance-9948",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-8451
{
  "DBInstanceIdentifier": "instance-8451",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-multiple-readers \
  --no-delete-automated-backups \
  --no-skip-final-snapshot \
  --final-db-snapshot-identifier deleteme-multiple-readers-final-snapshot
{
```

```
  "DBClusterIdentifier": "deleteme-multiple-readers",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

次に、要求されたスナップショットが Aurora により作成されたことを、確認する方法の例を示します。deleteme-multiple-readers-final-snapshot 識別子を指定することで、特定のスナップショットの詳細を要求できます。deleteme-multiple-readers クラスター識別子を指定することで、削除されたクラスターのすべてのスナップショットのレポートを取得することもできます。これらのコマンドはどちらも、同じスナップショットに関する情報を返しています。

```
$ aws rds describe-db-cluster-snapshots \
  --db-cluster-snapshot-identifier deleteme-multiple-readers-final-snapshot
{
    "DBClusterSnapshots": [
        {
            "AvailabilityZones": [],
            "DBClusterSnapshotIdentifier": "deleteme-multiple-readers-final-snapshot",
            "DBClusterIdentifier": "deleteme-multiple-readers",
            "SnapshotCreateTime": "11T01:40:07.354000+00:00",
            "Engine": "aurora-mysql",
...

$ aws rds describe-db-cluster-snapshots --db-cluster-identifier deleteme-multiple-
readers
{
    "DBClusterSnapshots": [
        {
            "AvailabilityZones": [],
            "DBClusterSnapshotIdentifier": "deleteme-multiple-readers-final-snapshot",
            "DBClusterIdentifier": "deleteme-multiple-readers",
            "SnapshotCreateTime": "11T01:40:07.354000+00:00",
            "Engine": "aurora-mysql",
...
```

Aurora クラスターの削除保護

削除保護が有効になっているクラスターは削除できません。そのクラスター内の DB インスタンスは削除が可能ですが、クラスター自体は削除できません。これにより、クラスターボリュームに保存されたすべてのデータが、誤って削除されることを防止できます。コンソール、AWS CLI、あるいは

RDS APIを使用して DB クラスターを削除しようとした場合でも、クラスターに対しては、削除保護が Aurora により強制的に適用されます。

AWS Management Console を使用して本稼働 DB クラスターを作成する場合は、デフォルトで削除保護は有効です。ただし、AWS CLI または API を使用してクラスターを作成する場合は、デフォルトで削除保護が無効になります。削除保護を有効または無効にしても、停止は発生しません。クラスターを削除できるようにするには、クラスターの設定を変更し、削除保護を無効にします。削除保護のオン/オフの切り替えの詳細については、「コンソール、CLI、API を使用した DB クラスターの変更」を参照してください。

> **Tip**
>
> すべての DB インスタンスが削除された後でも、クラスター内に新しい DB インスタンスを作成することで、データに再びアクセスできます。

停止した Aurora クラスターの削除

状態が `stopped` となっているクラスターは削除できません。この場合は、クラスターを削除する前にクラスターを起動します。詳細については、「Aurora DB クラスターの開始」を参照してください。

リードレプリカである Aurora MySQL クラスターの削除

Aurora MySQL では、以下の条件の両方に該当した場合、DB クラスター内の DB インスタンスを削除できません。

- DB クラスターは別の Aurora DB クラスターのリードレプリカです。
- DB インスタンスは、DB クラスター内の唯一のインスタンスです。

この場合に DB インスタンスを削除するには、まず DB クラスターを昇格させます。そうすれば、リードレプリカではなくなります。昇格が完了したら、DB クラスター内の最終 DB インスタンスを削除できます。詳細については、「AWS リージョン 間での Amazon Aurora MySQL DB クラスターのレプリケーション」を参照してください。

クラスター削除時の最終スナップショット

このセクションでは、Aurora クラスターを削除する際に最終スナップショットを作成するかどうかを選択する方法の例を示します。最終スナップショットを作成することを選択した場合でも、そこ

で指定した名前と同じ名前の既存のスナップショットが存在する場合には、このオペレーションはエラーとともに停止します。この場合、スナップショットの詳細を調査し、それが現在の状態に対応しているものかどうか、または古いスナップショットのものかどうかを確認します。既存のスナップショットに、保存すべき最新のデータが含まれていない場合は、スナップショットの名前を変更した上で再試行するか、[最終スナップショット] パラメータに別の名前を設定します。

Aurora DB クラスターからの DB インスタンスの削除

クラスター全体を削除するプロセスの一環として、Aurora DB クラスターから DB インスタンスを削除できます。クラスターに一定数の DB インスタンスが含まれている場合、クラスターを削除するには、各 DB インスタンスを削除していく必要があります。クラスターを実行したままの状態で、クラスターから 1 つ以上のリーダーインスタンスを削除することもできます。コンピューティング容量とそれに関連する料金を削減するために、ビジー状態ではないクラスターで、これを行うことができます。

DB インスタンスを削除するには、インスタンスの名前を指定します。

AWS Management Console、AWS CLI、RDS API を使用して DB インスタンスを削除できます。

> **Note**
>
> Aurora レプリカが削除されるとそのインスタンスエンドポイントは直ちに削除され、Aurora レプリカも読み込みエンドポイントから削除されます。削除中の Aurora レプリカで実行されているステートメントがある場合は、削除までに 3 分の猶予期間があります。既存のステートメントは、猶予期間中に終了する場合があります。猶予期間が終了すると、Aurora レプリカはシャットダウンし、削除されます。

Aurora DB クラスターの場合、DB インスタンスを削除しても、クラスター全体が必ずしも削除されるとは限りません。Aurora クラスターがビジーでない状態で、そのクラスター内の DB インスタンスを削除すると、コンピューティング容量とそれに関連する料金を削減できます。1 個または 0 個の DB インスタンスを持つ Aurora クラスターでの特殊な状況については、「単一の DB インスタンスを使用している Aurora クラスターの削除」および「空の Aurora クラスターの削除」を参照してください。

> **Note**
>
> 削除保護が有効になっている DB クラスターを削除することはできません。詳細については、「Aurora クラスターの削除保護」を参照してください。

DB クラスターの設定を変更することで、削除保護を無効にできます。詳細については、「Amazon Aurora DB クラスターの変更」を参照してください。

コンソール

DB クラスター内の DB インスタンスを削除するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択し、削除する DB インスタンスを選択します。
3. [アクション] で、[削除] を選択します。
4. ボックスに「**delete me**」と入力します。
5. [削除] を選択します。

AWS CLI

AWS CLI を使用して DB インスタンスを削除するには、delete-db-instance コマンドを呼び出して `--db-instance-identifier` 値を指定します。

Example

Linux、macOS、Unix の場合:

```
aws rds delete-db-instance \
    --db-instance-identifier mydbinstance
```

Windows の場合:

```
aws rds delete-db-instance ^
    --db-instance-identifier mydbinstance
```

RDS API

Amazon RDS API を使用して DB インスタンスを削除するには、 DeleteDBInstance オペレーションを呼び出して、`DBInstanceIdentifier` パラメータを指定します。

> **Note**
>
> DB インスタンスのステータスが `deleting` の場合、その CA 認定の値は、RDS コンソールにも、AWS CLI コマンドまたは RDS API オペレーションの出力にも表示されません。CA 認定の詳細については、「SSL/TLS を使用した DB クラスターへの接続の暗号化」を参照してください。

Amazon RDS リソースのタグ付け

Amazon RDS タグを使用して Amazon RDS リソースにメタデータを追加できます。タグを使用して、データベースインスタンス、スナップショット、Aurora クラスターなどに関する独自の表記を追加できます。そうすることで、Amazon RDS リソースを文書化することができます。また、自動メンテナンスの手順でタグを使用することもできます。

特に、これらのタグは IAM ポリシーで使用できます。これらを使用して、RDS リソースへのアクセスを管理したり、RDS リソースに適用できるアクションを制御したりできます。また、これらのタグを使用して、類似のリソースの費用をグループ化することで、コストを追跡できます。

次の Amazon RDS リソースにタグ付けができます。

- DB インスタンス
- DB クラスター
- DB クラスターエンドポイント
- リードレプリカ
- DB スナップショット
- DB クラススナップショット
- リザーブド DB インスタンス
- イベントサブスクリプション
- DB オプショングループ
- DB パラメータグループ
- DB クラスターのパラメータグループ
- DB サブネットグループ
- RDS プロキシ
- RDS Proxy エンドポイント
- ブルー/グリーンデプロイ
- ゼロ ETL 統合 (プレビュー)

> **Note**
>
> 現在、AWS Management Console を使用して、RDS プロキシおよび RDS プロキシエンドポイントにタグ付けすることはできません。

トピック

Amazon RDS リソースタグの概要

グリーン環境の設定を指定します。Amazon RDS タグは、Amazon RDS リソースを定義してそのリソースに関連付ける名前と値のペアです。その名前はキーと呼ばれます。キーの値の指定は省略可能です。タグを使用して、Amazon RDS リソースに任意の情報を割り当てることができます。例えば、タグキーを使用してカテゴリを定義し、タグ値をそのカテゴリのアイテムにすることができます。例えば、「project」というタグキーと「Salix」というタグ値を定義することができます。この場合、これらは Amazon RDS リソースが Salix プロジェクトに割り当てられていることを示しています。また、`environment=test` や `environment=production` などのタグキーを使用して Amazon RDS リソースがテスト用であるか本番稼働用であるかを示すこともできます。Amazon RDS リソースに関連付けられているメタデータの追跡が簡単になるように、一貫した一連のタグキーを使用することをお勧めします。

さらに、IAM ポリシーで条件を使用して、AWS リソースへのアクセスをそのリソースのタグに基づき制御できます。これを行うには、グローバル条件キー `aws:ResourceTag/`*`tag-key`* を使用します。詳細については、「AWS Identity and Access Management ユーザーガイド」の「AWS のリソースへのアクセスの制御」をご参照ください。

各 Amazon RDS リソースにはタグセットがあり、それぞれの Amazon RDS リソースに割り当てられているすべてのタグが含まれています。タグセットには最大 50 個のタグを含めることができ、空にすることもできます。既存のリソースタグと同じキーを持つタグを RDS リソースに追加した場合、既存の値は新しい値によって上書きされます。

AWS は、タグに意味を適用しません。タグは文字列として厳密に解釈されます。RDS は DB インスタンスまたはその他の RDS リソースにタグを設定できます。タグ設定は、リソースの作成時に使用するオプションによって異なります。例えば、Amazon RDS によって DB インスタンスが本稼働用またはテスト用であることを示すタグが追加されることがあります。

- タグキーは、必須のタグ名です。文字列値は、1 ~ 128 文字の Unicode 文字です。aws: または rds: をプレフィックスとして使用することはできません。文字列には、一連の Unicode 文字、数字、空白、「_」、「.」、「:」、「/」、「=」、「+」、「-」、「@」 (Java 正規表現: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-@]*)") のみ使用できます。

- タグ値は、タグの省略可能な文字列値です。文字列値は、1 ~ 256 文字の Unicode 文字です。文字列には、一連の Unicode 文字、数字、空白、「_」、「.」、「:」、「/」、「=」、「+」、「-」、「@」 (Java 正規表現: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-@]*)") のみ使用できます。

  値はタグセット内で一意である必要はなく、null を指定できます。例えば、project=Trinity と cost-center=Trinity のタグセット内に 1 つのキーと値のペアを使用できます。

AWS Management Console、AWS CLI、または Amazon RDS API を使用して、Amazon RDS リソースに対してタグを追加、一覧表示、削除できます。CLI または API を使用するときは、操作する RDS リソースの Amazon リソースネーム (ARN) を指定する必要があります。ARN の作成の詳細については、「Amazon RDS 用 ARN の構築」を参照してください。

タグは承認用にキャッシュに格納されます。そのため、Amazon RDS リソースに対するタグの追加や更新には数分かかることがあります。

IAM でのアクセスコントロールのタグ使用

IAM ポリシーでタグを使用して Amazon RDS リソースへのアクセスを管理できるようになりました。また、タグを使用して、Amazon RDS リソースに適用できるアクションを制御できます。

IAM ポリシーでタグ付きリソースへのアクセスを管理する方法については、「Amazon Aurora での Identity and Access Management」を参照してください。

タグを使用した請求明細レポートの作成

また、タグを使用して、類似のリソースの費用をグループ化することで、コストを追跡できます。

タグを使用して、自分のコスト構造を反映するように AWS 請求書を整理します。そのためには、サインアップして、タグキー値が含まれた AWS アカウント の請求書を取得する必要があります。次に、結合したリソースのコストを見るには、同じタグキー値のリソースに従って請求書情報を整理します。例えば、複数のリソースに特定のアプリケーション名のタグを付け、請求情報を整理することで、複数のサービスを利用しているアプリケーションの合計コストを確認することができます。詳細については、AWS Billingユーザーガイド の「コスト配分タグの使用」をご参照ください。

> **Note**
>
> DB クラスター スナップショットにタグを追加できますが、請求書にはこのグループが反映されません。
> コスト配分タグを DB クラススナップショットに適用するには、タグを親 DB クラスターにアタッチし、親クラスターがスナップショットと同じ AWS リージョン に存在する必要があります。孤立したスナップショットのコストは、タグのない単一の項目に集約されます。

タグの追加、リスト化、削除

次の手順では、DB インスタンスおよび Aurora DB クラスターに関連するリソースに対して一般的なタグ付け操作を実行する方法を示しています。

コンソール

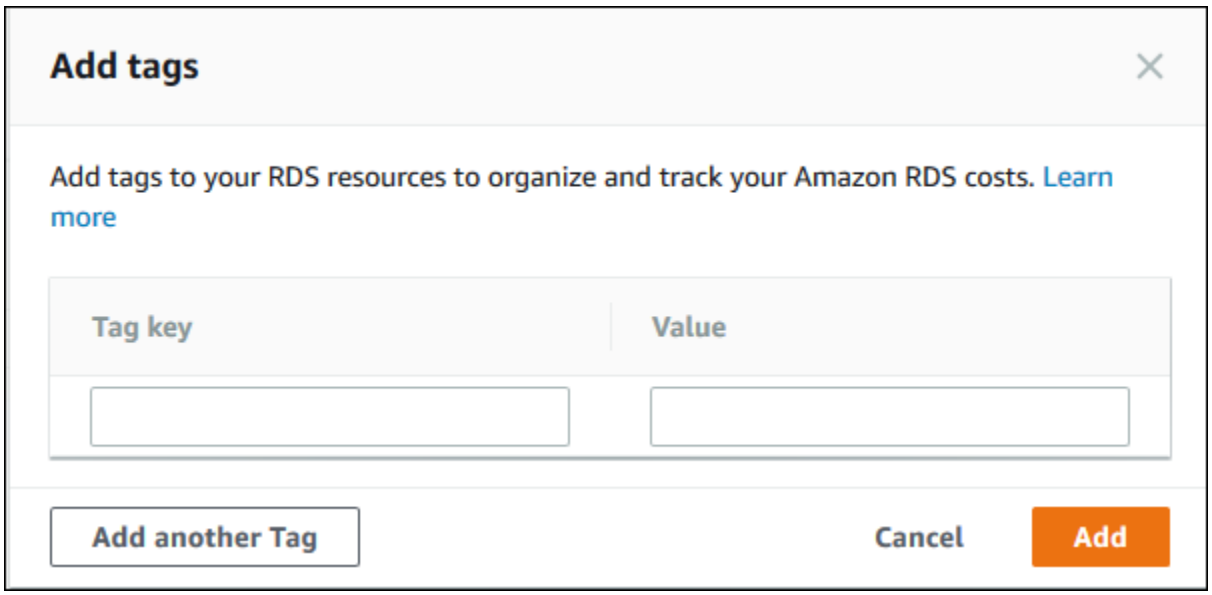
Amazon RDS リソースにタグを追加するプロセスはすべてのリソースで同様です。以下の手順では、Amazon RDS DB インスタンスにタグを付加する方法を示します。

DB インスタンスにタグを追加するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択します。

   > **Note**
   >
   > [Filter databases (データベースのフィルター)] ペインで DB インスタンスの一覧をフィルターするには、[Filter databases (データベースのフィルター)] のテキスト文字列を入力します。その文字列を含む DB インスタンスのみが表示されます。

3. タグ付けする DB インスタンスの名前を選択して、その詳細を表示します。
4. 詳細セクションで、下にスクロールし、[タグ] を選択します。
5. 追加] を選択します。[タグの追加] ウィンドウが表示されます。



6. [タグキー] と [値] の値を入力します。
7. 別のタグを追加するには、[別のタグを追加] を選択し、[タブキー] と [値] の値を入力します。

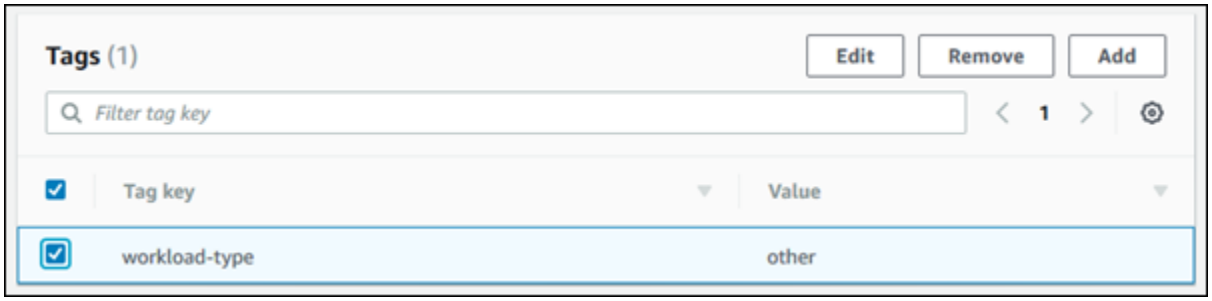
   このステップを必要な回数繰り返します。
8. 追加] を選択します。

DB インスタンスからタグを削除するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択します。

> **Note**
>
> [Filter databases (データベースのフィルター)] ペインで DB インスタンスの一覧をフィルターするには、[Filter databases (データベースのフィルター)] ボックスにテキスト文字列を入力します。その文字列を含む DB インスタンスのみが表示されます。

3. DB インスタンスの名前を選択して、その詳細を表示します。
4. 詳細セクションで、下にスクロールし、[タグ] を選択します。
5. 削除するタグを選択します。



6. [削除] を選択し、[Delete tags] (タグの削除) ウィンドウから [削除] を選択します。

AWS CLI

AWS CLI を使用して DB インスタンスのタグを追加、一覧表示、または削除できます。

- Amazon RDS リソースに 1 つ以上のタグを追加するには、AWS CLI コマンド `add-tags-to-resource` を使用します。
- Amazon RDS リソースのタグを一覧表示するには、AWS CLI コマンド `list-tags-for-resource` を使用します。
- Amazon RDS リソースから 1 つ以上のタグを削除するには、AWS CLI コマンド `remove-tags-from-resource` を使用します。

必要な ARN を作成する方法の詳細については、「Amazon RDS 用 ARN の構築」を参照してください。

RDS API

Amazon RDS API を使用して DB インスタンスのタグを追加、一覧表示、または削除できます。

- Amazon RDS リソースにタグを追加するには、`AddTagsToResource` オペレーションを使用します。
- Amazon RDS リソースに割り当てられているタグを一覧表示するには、`ListTagsForResource` を使用します。
- Amazon RDS リソースからタグを削除するには、`RemoveTagsFromResource` オペレーションを使用します。

必要な ARN を作成する方法の詳細については、「Amazon RDS 用 ARN の構築」を参照してください。

Amazon RDS API を使用して XML を操作する場合、タグでは以下のスキーマを使用します。

```
<Tagging>
    <TagSet>
        <Tag>
            <Key>Project</Key>
            <Value>Trinity</Value>
        </Tag>
        <Tag>
            <Key>User</Key>
            <Value>Jones</Value>
        </Tag>
    </TagSet>
</Tagging>
```

以下の表に示しているのは、使用可能な XML タグとその特性のリストです。キーと値では大文字と小文字が区別されます。例えば、project=Trinity と PROJECT=Trinity は 2 つの別個のタグです。

タグ付け要素	説明
タグセット	タグセットは、Amazon RDS リソースに割り当てられるすべてのタグのコンテナです。リソースごとに割り当て可能なのは 1 つのタグセットのみです。Amazon RDS API によってのみタグセットを操作できます。
Tag	タグはユーザー定義のキーと値のペアです。1～50 個のタグをタグセットに含めることができます。
キー	キーはタグの必須の名前です。文字列値は、1〜128 文字の Unicode 文字です。`aws:` または `rds:` をプレフィックスとして使用することはできません。文字列には、一連の Unicode 文字、数字、空白、「_」、「.」、「/」、「=」、「+」、「-」 (Java 正規表現: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)") のみ使用できます。 キーはタグセットに対して一意である必要があります。例えば、タグセットでキーが同じで値が異なるキーと値のペアは使用できません。例えば、project/Trinity や project/Xanadu は使用できません。
値	値はタグの省略可能な値です。文字列値は、1～256 文字の Unicode 文字です。`aws:` または `rds:` をプレフィックスとして使用することはできません。文字列には、一連の Unicode 文字、数字、空白、「_」、

タグ付け要素	説明
	「.」、「/」、「=」、「+」、「-」 (Java 正規表現: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)") のみ使用できます。 値はタグセット内で一意である必要はなく、null を指定できます。例えば、project/Trinity と cost-center/Trinity のタグセット内に 1 つのキーと値のペアを使用できます。

AWS タグエディタの使用

AWS Management Console タグエディタを使用して、AWS で RDS リソースのタグを参照および編集できます。詳細については、AWS リソースグループユーザーガイドのタグエディタを参照してください。

DB クラスタースナップショットへのタグのコピー

DB クラスターを作成または復元するときに、DB クラスターのタグを DB クラスターのスナップショットにコピーするように指定できます。タグをコピーすると、DB スナップショットとソース DB クラスターのメタデータが確実に一致するようになります。また、DB スナップショットとソース DB クラスターのアクセスポリシーが確実に一致するようになります。タグは、デフォルトではコピーされません。

次のアクションでタグが DB スナップショットにコピーされるように指定できます。

- DB クラスターの作成。
- DB クラスターを復元します。
- リードレプリカの作成。
- DB クラスターのスナップショットのコピー。

> **Note**
>
> 場合によっては、create-db-snapshot AWS CLI コマンドの `--tags` パラメータに値を含めることができます。または、CreateDBSnapshot API オペレーションに少なくとも 1 つのタグを指定することもできます。このような場合、RDS はソース DB インスタンスから新しい DB スナップショットにタグをコピーしません。この機能は、ソース DB インスタンスの `--`

copy-tags-to-snapshot (CopyTagsToSnapshot) オプションが有効になっている場合でも、適用されます。
このアプローチを使用すると、DB スナップショットから DB インスタンスのコピーを作成できます。この方法では、新しい DB インスタンスに適用されないタグを追加する必要がなくなります。DB スナップショットは、AWS CLIcreate-db-snapshotコマンド (またはCreateDBSnapshot RDS API オペレーション) を使用して作成します。DB スナップショットを作成した後、このトピックで説明しているように、タグを追加することができます。

チュートリアル: タグを使用して、停止する Aurora DB クラスターを指定します

開発環境またはテスト環境で多数の Aurora DB クラスターを作成するとします。これらのクラスターをすべて数日間保持する必要があります。一部のクラスターは、夜間にテストを実施します。他のクラスターは、夜間に停止し、翌日に再び開始することができます。次の例では、夜間の停止に適したクラスターにタグを割り当てる方法を示しています。次に、この例では、スクリプトがそのタグを持つクラスターを検出し、それらのクラスターを停止する方法を示しています。この例では、キーと値のペアでの値の部分は重要ではありません。stoppable タグが存在するということは、クラスターがこのユーザー定義プロパティを持っていることを示します。

停止する Aurora DB クラスターを指定するには

1. 停止可能として指定するクラスターの ARN を決めます。

   タグ付け用のコマンドと API は、ARN で使用できます。そうすることで、AWS リージョン、AWS アカウント、および同様の短い名前を持つ可能性のあるさまざまなタイプのリソース間でシームレスに機能できます。クラスターで動作する CLI コマンドでは、クラスター ID の代わりに ARN を指定できます。*dev-test-cluster* は、独自のクラスターの名前に置き換えます。ARN パラメータを使用する後続のコマンドでは、独自のクラスターの ARN を置き換えます。ARN には、独自の AWS アカウント ID と、クラスターが配置されている AWS リージョンの名前が含まれます。

   ```
   $ aws rds describe-db-clusters --db-cluster-identifier dev-test-cluster \
     --query "*[].{DBClusterArn:DBClusterArn}" --output text
   arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
   ```

2. このクラスターにタグ stoppable を追加します。

このタグの名前を選択します。このアプローチにより、すべての関連情報を名前にエンコードする命名規則を考案する必要がなくなります。このような規則では、DB インスタンス名または他のリソースの名前に情報をエンコードすることができます。この例では、タグが存在するか存在しないかの属性として扱うため、Value= パラメータの --tags 部分を省略します。

```
$ aws rds add-tags-to-resource \
  --resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster \
  --tags Key=stoppable
```

3. タグがクラスターに存在することを確認します。

   これらのコマンドは、クラスターのタグ情報を JSON 形式およびタブ区切りのテキストで取得します。

```
$ aws rds list-tags-for-resource \
  --resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
{
    "TagList": [
        {
            "Key": "stoppable",
            "Value": ""

        }
    ]
}
$ aws rds list-tags-for-resource \
  --resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster --output
 text
TAGLIST stoppable
```

4. stoppable に指定されているすべてのクラスターを停止するには、すべてのクラスターのリストを準備します。リストをループし、各クラスターが関連する属性でタグ付けされているかどうかを確認します。

   この Linux の例では、シェルスクリプトを使用してクラスター ARN のリストを一時ファイルに保存し、クラスターごとに CLI コマンドを実行します。

```
$ aws rds describe-db-clusters --query "*[].[DBClusterArn]" --output text >/tmp/
cluster_arns.lst
$ for arn in $(cat /tmp/cluster_arns.lst)
do
```

```
  match="$(aws rds list-tags-for-resource --resource-name $arn --output text | grep
 'TAGLIST\tstoppable')"
  if [[ ! -z "$match" ]]
  then
      echo "Cluster $arn is tagged as stoppable. Stopping it now."
# Note that you can specify the full ARN value as the parameter instead of the
 short ID 'dev-test-cluster'.
      aws rds stop-db-cluster --db-cluster-identifier $arn
  fi
done

Cluster arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster is tagged as
 stoppable. Stopping it now.
{
    "DBCluster": {
        "AllocatedStorage": 1,
        "AvailabilityZones": [
            "us-east-1e",
            "us-east-1c",
            "us-east-1d"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "dev-test-cluster",
        ...
```

このようなスクリプトを 1 日の終わりに実行して、重要でないクラスターが停止していることを確認できます。cron などのユーティリティを使用してジョブのスケジュールを組み、毎晩そのような確認を実行することもできます。例えば、一部のクラスターが誤って実行されたままになった場合にこれを行うことができます。ここでは、確認するクラスターのリストを準備するコマンドを微調整できます。

次のコマンドは、クラスターのリストを生成しますが、available 状態のクラスターのみを生成します。スクリプトでは、すでに停止しているクラスターを無視できます。これは、stopped または stopping などのステータス値が異なるためです。

```
$ aws rds describe-db-clusters \
  --query '*[].{DBClusterArn:DBClusterArn,Status:Status}|[?Status == `available`]|[].
{DBClusterArn:DBClusterArn}' \
  --output text
arn:aws:rds:us-east-1:123456789:cluster:cluster-2447
arn:aws:rds:us-east-1:123456789:cluster:cluster-3395
```

```
arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
arn:aws:rds:us-east-1:123456789:cluster:pg2-cluster
```

> **Tip**
>
> タグを割り当てて、それらのタグを持つクラスターを検索して、他の方法でコストを削減することができます。例えば、開発とテストに使用される Aurora DB クラスターのシナリオを考えてみましょう。ここでは、一部のクラスターを 1 日の終わりに削除するか、読み取り DB インスタンスを 1 日の終わりに削除するように指定することができます。あるいは、使用率が低いと予想される時間帯に DB インスタンスを小さな DB インスタンスクラスに変更するように指定することもできます。

Amazon RDS の Amazon リソースネーム (ARN) の使用

Amazon Web Services で作成されたリソースは、Amazon リソースネーム (ARN) によってそれぞれ一意に識別されます。特定の Amazon RDS オペレーションでは、ARN を指定して、Amazon RDS リソースを一意に識別する必要があります。例えば、RDS DB インスタンスのリードレプリカを作成する場合、ソース DB インスタンスの ARN を指定する必要があります。

Amazon RDS 用 ARN の構築

Amazon Web Services で作成されたリソースは、Amazon リソースネーム (ARN) によってそれぞれ一意に識別されます。次の構文を使用して Amazon RDS リソースの ARN を構築できます。

```
arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

リージョン名	リージョン	エンドポイント	プロトコル	
米国東部 (オハイオ)	us-east-2	rds.us-east-2.amazonaws.com	HTTPS	
		rds-fips.us-east-2.api.aws	HTTPS	
		rds.us-east-2.api.aws	HTTPS	
		rds-fips.us-east-2.amazonaws.com	HTTPS	
米国東部 (バージニア北部)	us-east-1	rds.us-east-1.amazonaws.com	HTTPS	
		rds-fips.us-east-1.api.aws	HTTPS	
		rds-fips.us-east-1.amazonaws.com	HTTPS	
		rds.us-east-1.api.aws	HTTPS	
米国西部 (北カリフォルニア)	us-west-1	rds.us-west-1.amazonaws.com	HTTPS	
		rds.us-west-1.api.aws	HTTPS	
		rds-fips.us-west-1.amazonaws.com	HTTPS	
		rds-fips.us-west-1.api.aws	HTTPS	

リージョン名	リージョン	エンドポイント	プロトコル
米国西部 (オレゴン)	us-west-2	rds.us-west-2.amazonaws.com	HTTPS
		rds-fips.us-west-2.amazonaws.com	HTTPS
		rds.us-west-2.api.aws	HTTPS
		rds-fips.us-west-2.api.aws	HTTPS
アフリカ (ケープタウン)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS
		rds.af-south-1.api.aws	HTTPS
アジアパシフィック (香港)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS
		rds.ap-east-1.api.aws	HTTPS
アジアパシフィック (ハイデラバード)	ap-south-2	rds.ap-south-2.amazonaws.com	HTTPS
		rds.ap-south-2.api.aws	HTTPS
アジアパシフィック (ジャカルタ)	ap-southeast-3	rds.ap-southeast-3.amazonaws.com	HTTPS
		rds.ap-southeast-3.api.aws	HTTPS
アジアパシフィック (メルボルン)	ap-southeast-4	rds.ap-southeast-4.amazonaws.com	HTTPS
		rds.ap-southeast-4.api.aws	HTTPS
アジアパシフィック (ムンバイ)	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS
		rds.ap-south-1.api.aws	HTTPS

リージョン名	リージョン	エンドポイント	プロトコル
アジアパシフィック (大阪)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com	HTTPS
		rds.ap-northeast-3.api.aws	HTTPS
アジアパシフィック (ソウル)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS
		rds.ap-northeast-2.api.aws	HTTPS
アジアパシフィック (シンガポール)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS
		rds.ap-southeast-1.api.aws	HTTPS
アジアパシフィック (シドニー)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS
		rds.ap-southeast-2.api.aws	HTTPS
アジアパシフィック (東京)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS
		rds.ap-northeast-1.api.aws	HTTPS
カナダ (中部)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS
		rds.ca-central-1.api.aws	HTTPS
		rds-fips.ca-central-1.api.aws	HTTPS
		rds-fips.ca-central-1.amazonaws.com	HTTPS
カナダ西部 (カルガリー)	ca-west-1	rds.ca-west-1.amazonaws.com	HTTPS
		rds-fips.ca-west-1.amazonaws.com	HTTPS

リージョン名	リージョン	エンドポイント	プロトコル
欧州 (フランクフルト)	eu-central-1	rds.eu-central-1.amazonaws.com rds.eu-central-1.api.aws	HTTPS HTTPS
欧州 (アイルランド)	eu-west-1	rds.eu-west-1.amazonaws.com rds.eu-west-1.api.aws	HTTPS HTTPS
欧州 (ロンドン)	eu-west-2	rds.eu-west-2.amazonaws.com rds.eu-west-2.api.aws	HTTPS HTTPS
ヨーロッパ (ミラノ)	eu-south-1	rds.eu-south-1.amazonaws.com rds.eu-south-1.api.aws	HTTPS HTTPS
欧州 (パリ)	eu-west-3	rds.eu-west-3.amazonaws.com rds.eu-west-3.api.aws	HTTPS HTTPS
欧州 (スペイン)	eu-south-2	rds.eu-south-2.amazonaws.com rds.eu-south-2.api.aws	HTTPS HTTPS
欧州 (ストックホルム)	eu-north-1	rds.eu-north-1.amazonaws.com rds.eu-north-1.api.aws	HTTPS HTTPS
欧州 (チューリッヒ)	eu-central-2	rds.eu-central-2.amazonaws.com rds.eu-central-2.api.aws	HTTPS HTTPS
イスラエル (テルアビブ)	il-central-1	rds.il-central-1.amazonaws.com rds.il-central-1.api.aws	HTTPS HTTPS

リージョン名	リージョン	エンドポイント	プロトコル
中東 (バーレーン)	me-south-1	rds.me-south-1.amazonaws.com	HTTPS
		rds.me-south-1.api.aws	HTTPS
中東 (アラブ首長国連邦)	me-central-1	rds.me-central-1.amazonaws.com	HTTPS
		rds.me-central-1.api.aws	HTTPS
南米 (サンパウロ)	sa-east-1	rds.sa-east-1.amazonaws.com	HTTPS
		rds.sa-east-1.api.aws	HTTPS
AWS GovCloud (米国東部)	us-gov-east-1	rds.us-gov-east-1.amazonaws.com	HTTPS
		rds.us-gov-east-1.api.aws	HTTPS
AWS GovCloud (米国西部)	us-gov-west-1	rds.us-gov-west-1.amazonaws.com	HTTPS
		rds.us-gov-west-1.api.aws	HTTPS

次の表に、特定の Amazon RDS リソースの ARN の構築時に使用する形式を示します。

リソースタイプ	ARN 形式
DB インスタンス	arn:aws:rds:*<region>*:*<account>* :db:*<name>* 例: `arn:aws:rds: us-east-2 :123456789012 :db:my-mysql-instance-1`
DB クラスター	arn:aws:rds:*<region>*:*<account>* :cluster:*<name>*

リソースタイプ	ARN 形式
	例: `arn:aws:rds: us-east-2 :123456789012 :cluster: my-aurora-cluster-1`
イベントサブスクリプション	arn:aws:rds:*<region>*:*<account>* :es:*<name>* 例: `arn:aws:rds: us-east-2 :123456789012 :es:my-subscription`
DB パラメータグループ	arn:aws:rds:*<region>*:*<account>* :pg:*<name>* 例: `arn:aws:rds: us-east-2 :123456789012 :pg:my-param-enable-logs`
DB クラスターのパラメータグループ	arn:aws:rds:*<region>*:*<account>* :cluster-pg:*<name>* 例: `arn:aws:rds: us-east-2 :123456789012 :cluster-pg: my-cluster-param-timezone`
リザーブド DB インスタンス	arn:aws:rds:*<region>*:*<account>* :ri:*<name>* 例: `arn:aws:rds: us-east-2 :123456789012 :ri:my-reserved-postgresql`

リソースタイプ	ARN 形式
DB セキュリティグループ	arn:aws:rds:*<region>*:*<account>* :secgrp:*<name>* 例: `arn:aws:rds: us-east-2 :123456789012 :secgrp:my-public`
Automated DB スナップショット	arn:aws:rds:*<region>*:*<account>* :snapshot:rds:*<name>* 例: `arn:aws:rds: us-east-2 :123456789012 :snapshot:rds: my-mysql-db-2019-07-22-07-23`
自動 DB クラススナップショット	arn:aws:rds:*<region>*:*<account>* :cluster-snapshot:rds:*<name>* 例: `arn:aws:rds: us-east-2 :123456789012 :cluster-snapshot:rds: my-aurora-cluster-2019-07-22-16-16`
手動 DB スナップショット	arn:aws:rds:*<region>*:*<account>* :snapshot:*<name>* 例: `arn:aws:rds: us-east-2 :123456789012 :snapshot: my-mysql-db-snap`
手動の DB クラススナップショット	arn:aws:rds:*<region>*:*<account>* :cluster-snapshot:*<name>* 例: `arn:aws:rds: us-east-2 :123456789012 :cluster-snapshot: my-aurora-cluster-snap`

リソースタイプ	ARN 形式
DB サブネットグループ	arn:aws:rds:*<region>*:*<account>* :subgrp:*<name>* 例: `arn:aws:rds: us-east-2 :123456789012 :subgrp:my-subnet-10`

既存の ARN の取得

AWS Management Console、AWS Command Line Interface (AWS CLI)、または RDS API を使用して、RDS リソースの ARN を取得できます。

コンソール

AWS Management Console から ARN を取得するには、ARN を取得したいリソースに移動し、リソースの詳細を表示します。

例えば、DB クラスターの詳細の [設定] タブから、DB クラスターの ARN を取得できます。

AWS CLI

特定の RDS リソースの AWS CLI から ARN を取得するには、そのリソースに対して `describe` コマンドを使用します。次の表に、各 AWS CLI コマンド、および ARN を取得するコマンドで使用された ARN のプロパティを示します。

AWS CLI コマンド	ARN プロパティ
describe-event-subscriptions	EventSubscriptionArn
describe-certificates	CertificateArn
describe-db-parameter-groups	DBParameterGroupArn
describe-db-cluster-parameter-groups	DBClusterParameterGroupArn
describe-db-instances	DBInstanceArn

AWS CLI コマンド	ARN プロパティ
describe-db-security-groups	DBSecurityGroupArn
describe-db-snapshots	DBSnapshotArn
describe-events	SourceArn
describe-reserved-db-instances	ReservedDBInstanceArn
describe-db-subnet-groups	DBSubnetGroupArn
describe-db-clusters	DBClusterArn
describe-db-cluster-snapshots	DBClusterSnapshotArn

例えば、次の AWS CLI コマンドで DB インスタンスの ARN を取得します。

Example

Linux、macOS、Unix の場合:

```
aws rds describe-db-instances \
--db-instance-identifier DBInstanceIdentifier \
--region us-west-2 \
--query "*[].{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceArn:DBInstanceArn}"
```

Windows の場合:

```
aws rds describe-db-instances ^
--db-instance-identifier DBInstanceIdentifier ^
--region us-west-2 ^
--query "*[].{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceArn:DBInstanceArn}"
```

このコマンドの出力は次のようになります。

```
[
    {
        "DBInstanceArn": "arn:aws:rds:us-west-2:account_id:db:instance_id",
        "DBInstanceIdentifier": "instance_id"
```

```
    }
]
```

RDS API

特定の RDS リソースの ARN を取得するには、次の RDS API のオペレーションを呼び出し、次に示す ARN のプロパティを使用できます。

RDS API オペレーション	ARN プロパティ
DescribeEventSubscriptions	EventSubscriptionArn
DescribeCertificates	CertificateArn
DescribeDBParameterGroups	DBParameterGroupArn
DescribeDBClusterParameterGroups	DBClusterParameterGroupArn
DescribeDBInstances	DBInstanceArn
DescribeDBSecurityGroups	DBSecurityGroupArn
DescribeDBSnapshots	DBSnapshotArn
DescribeEvents	SourceArn
DescribeReservedDBInstances	ReservedDBInstanceArn
DescribeDBSubnetGroups	DBSubnetGroupArn
DescribeDBClusters	DBClusterArn
DescribeDBClusterSnapshots	DBClusterSnapshotArn

Amazon Aurora の更新

Amazon Aurora は定期的に更新をリリースします。更新はシステムメンテナンスの時間中に Amazon Aurora DB クラスターに適用されます。更新が適用されるタイミングは、DB クラスターのリージョンやメンテナンスウィンドウの設定、および更新のタイプによって異なります。更新にはデータベースの再起動が必要になるため、通常、20〜30 秒のダウンタイムが発生します。このダウンタイム後に、DB クラスターの使用を再開できます。AWS Management Consoleでメンテナンスウィンドウの設定を表示または変更できます。

> **Note**
>
> DB インスタンスの再起動に必要な時間は、クラッシュ回復プロセス、再起動時のデータベースアクティビティ、および特定の DB エンジンの動作によって異なります。再起動時間を短くするには、再起動プロセス中のデータベースアクティビティをできる限り減らすことをお勧めします。データベースアクティビティを減らすと、未完了のトランザクションのロールバックアクティビティが減少します。

Amazon Aurora のオペレーティングシステムアップデートについては、「オペレーティングシステムアップデートの操作」を参照してください。

更新の一部は、Aurora でサポートされるデータベースエンジンに固有です。データベースエンジンの更新の詳細については、次の表を参照してください。

データベースエンジン	更新
Amazon Aurora MySQL	「Amazon Aurora MySQL のデータベースエンジンの更新」を参照してください。
Amazon Aurora PostgreSQL	「Amazon Aurora PostgreSQL の更新」を参照してください。

Amazon Aurora バージョンの確認

Amazon Aurora には、Aurora としては一般的で、すべての Aurora DB クラスターで使用できる一定の機能が含まれています。その他として Aurora には、Aurora がサポートする特定のデータベースエンジン固有の機能が含まれます。これらの機能は、各データベースエンジン (Aurora PostgreSQL など) を使用する Aurora DB クラスターでのみ使用できます。

Aurora DB インスタンスには、Aurora バージョン番号と Aurora データベースエンジンバージョン番号の 2 つのバージョン番号があります。Aurora バージョン番号では以下の形式を使用します。

```
<major version>.<minor version>.<patch version>
```

特定のデータベースエンジンを使用する Aurora DB インスタンスの Aurora バージョン番号を取得するには、以下のクエリのいずれかを使用します。

データベースエンジン	クエリ
Amazon Aurora MySQL	`SELECT AURORA_VERSION();` `SHOW @@aurora_version;`
Amazon Aurora PostgreSQL	`SELECT AURORA_VERSION();`

Amazon RDS 延長サポートの使用

Amazon RDS 延長サポートを利用すると、Aurora標準サポート終了日以降も、データベースを以前のメジャーエンジンバージョンで引き続き実行できます (追加料金がかかります)。Aurora 標準サポート終了日に、Amazon Aurora はデータベースを RDS 延長サポートに自動的に登録します。RDS 延長サポートへの自動登録は、データベースエンジンを変更せず、DB インスタンスの稼働時間やパフォーマンスにも影響しません。

この有料サービスを利用すると、サポートされているメジャーエンジンバージョンへのアップグレードにかかる時間が長くなります。

たとえば、Aurora MySQL バージョン 2 の Aurora 標準サポート終了日は 2024 年 10 月 31 日です。ただし、その日付より前に Aurora MySQL version 3 に手動でアップグレードすることはできません。この場合、Amazon Aurora は、2024 年 10 月 31 日にクラスターを RDS 延長サポートに自動的に登録するため、引き続き Aurora MySQL バージョン 2 を実行できます。2024 年 12 月 1 日以降、Amazon Aurora によって RDS 延長サポートの料金が自動的に請求されます。

RDS 延長サポートは、Aurora のメジャーエンジンバージョン終了日から最大 3 年間ご利用いただけます (Aurora MySQL バージョン 2 では 3 年 4 か月)。この期間が過ぎてもメジャーエンジンバージョンをサポート対象バージョンにアップグレードしていない場合、Amazon Aurora によってメジャーエンジンバージョンが自動的にアップグレードされます。サポート対象のメジャーエンジンバージョンへできるだけ早くアップグレードすることをお勧めします。

トピック

- Amazon RDS 延長サポートの概要
- Amazon RDS 延長サポートでの Aurora DB クラスターまたはグローバルクラスターの作成
- Amazon RDS 延長サポートでの Aurora DB クラスターまたはグローバルクラスターの登録を確認します。
- Amazon RDS 延長サポートでの Aurora DB クラスターまたはグローバルクラスターの復元

Amazon RDS 延長サポートの概要

Aurora 標準サポート終了日が過ぎると、Amazon Aurora はデータベースを RDS 延長サポートに自動的に登録します。Aurora は、DB インスタンスを、Aurora 標準サポート終了日より前にリリースされた最新のマイナーバージョンに自動的にアップグレードします (該当バージョンをまだ実行して

いない場合)。Amazon Aurora は、メジャーエンジンバージョンの Aurora 標準サポート終了日が過ぎるまでは、マイナーバージョンをアップグレードしません。

Aurora 標準サポート終了日に達したメジャーエンジンバージョンで新しいデータベースを作成できます。Aurora は、これらの新しいデータベースを RDS 延長サポートに自動的に登録し、このサービスの料金を請求します。

Aurora 標準サポート終了日前に Aurora 標準サポートがまだ適用されているエンジンにアップグレードすると、Amazon Aurora はエンジンを RDS 延長サポートに登録しません。

Aurora 標準サポート終了日を過ぎているが、RDS 延長サポートに登録していないエンジンと互換性があるデータベースのスナップショットを復元しようとすると、Amazon Aurora は、 Aurora 標準サポートがまだ適用さている最新のエンジンバージョンとの互換性を持つようにスナップショットをアップグレードしようとします。復元に失敗すると、Amazon Aurora は、スナップショットと互換性があるバージョンを使用してエンジンを RDS 延長サポートに自動的に登録します。

RDS 延長サポートへの登録はいつでも終了できます。登録を終了するには、登録した各エンジンを、Aurora 標準サポートがまだ適用されている、より新しいエンジンバージョンにアップグレードします。RDS 延長サポートへの登録の終了は、Aurora 標準サポートがまだ適用されている、より新しいエンジンバージョンへのアップグレードを完了した日から有効になります。

トピック

- Amazon RDS 延長サポート料金
- Amazon RDS 延長サポートが適用されるバージョン
- Amazon RDS 延長サポートにおける Amazon Aurora とお客様の責任

Amazon RDS 延長サポート料金

RDS 延長サポートに登録しているすべてのエンジンには、Aurora 標準サポート終了日の翌日から、料金が発生します。Aurora 標準サポート終了日については、「Amazon Aurora メジャーバージョン」を参照してください。

RDS 延長サポートの追加料金は、以下のいずれかのアクションを実行すると、自動的に停止します。

- 標準サポートの対象となるエンジンバージョンにアップグレードします。
- Aurora 標準サポート終了日を過ぎてメジャーバージョンを実行しているデータベースを削除します。

将来、ターゲットエンジンバージョンが延長サポートに移行すると、料金が再び発生します。

例えば、Aurora PostgreSQL 11 は 2024 年 3 月 1 日に延長サポートが開始されますが、2024 年 4 月 1 日まで課金は開始されません。2024 年 4 月 30 日に、Aurora PostgreSQL 11 データベースを Aurora PostgreSQL 12 にアップグレードします。Aurora PostgreSQL 11 の延長サポートに対しては 30 日間のみ課金されます。RDS 標準サポート終了日である 2025 年 2 月 28 日が過ぎても、この DB インスタンスで Aurora PostgreSQL 12 を引き続き実行します。データベースには、2025 年 3 月 1 日から RDS 延長サポート料金が再び発生します。

詳細については、「「Amazon Aurora 価格設定」を参照してください。

Amazon RDS 延長サポート料金の回避

RDS 延長サポート料金を回避するには、Aurora 標準サポート終了日が過ぎたら、 Aurora で Aurora DB クラスターまたはグローバルクラスターを作成または復元できないようにします。これには、AWS CLI または RDS API を使用します。

AWS CLI の `--engine-lifecycle-support` オプションで、`open-source-rds-extended-support-disabled` と指定します。RDS API で、`LifeCycleSupport` パラメータに `open-source-rds-extended-support-disabled` を指定します。詳細については、「Aurora DB クラスターまたはグローバルクラスターの作成」または「Aurora DB クラスターまたはグローバルクラスターの復元」を参照してください。

Amazon RDS 延長サポートが適用されるバージョン

RDS 延長サポートは、Aurora MySQL バージョン 2 と 3、および Aurora PostgreSQL バージョン 11 以降で利用できます。詳細については、「Amazon Aurora メジャーバージョン」を参照してください。

RDS 延長サポートは、特定のマイナーバージョンでのみ利用できます。詳細については、「Amazon Aurora マイナーバージョン」を参照してください。

RDS 延長サポートは、Aurora Serverless v2 でのみ利用できます。Aurora Serverless v1 では利用できません。

Amazon RDS 延長サポートにおける Amazon Aurora とお客様の責任

RDS 延長サポートにおける Amazon Aurora とお客様の責任について以下に説明します。

トピック

Amazon Aurora の責任

Aurora の標準サポート終了日以降、Amazon Aurora は RDS 延長サポートに登録しているエンジンに対してパッチ、バグ修正、アップグレードを提供します。この期間は、最長で 3 年間、またはエンジンの使用を停止するまでのいずれか早いほうとなります。

パッチは、国家脆弱性データベース (NVD) の CVSS 重大度評価で定義されている重大および高 CVE が対象となります。詳細については、「脆弱性メトリクス」を参照してください。

お客様の責任

お客様は、RDS 延長サポートに登録しているAurora DB クラスターまたはグローバルクラスターに対して提供されるパッチ、バグ修正、アップグレードを適用する責任があります。Amazon Aurora は、このようなパッチ、バグ修正、アップグレードをいつでも変更、置換、または撤回する権利を留保します。セキュリティまたは重大な安定性の問題に対処するためにパッチが必要な場合、Amazon Aurora は、お客様の Aurora DB クラスターまたはグローバルクラスターをパッチで更新するか、お客様にパッチのインストールを要求する権利を留保します。

また、お客様は、RDS 延長サポート終了日より前にお客様のエンジンをより新しいエンジンバージョンにアップグレードする責任もあります。RDS 延長サポート終了日は、通常、コミュニティサポート終了日から 3 年です。。データベースのメジャーエンジンバージョンの RDS 延長サポート終了日については、「Amazon Aurora メジャーバージョン」を参照してください。

お客様がエンジンをアップグレードしない場合、RDS 延長サポート終了日が過ぎると、Amazon Aurora は、エンジンを Aurora 標準サポートの対象である最新のエンジンバージョンにアップグレードしようとします。アップグレードに失敗すると、Amazon Aurora は、Aurora 標準サポート終了日を過ぎてエンジンを実行している Aurora DB クラスターまたはグローバルクラスターを削除する権利を留保します。ただし、その前に、Amazon Aurora はそのエンジンのデータを保存します。

Amazon RDS 延長サポートでの Aurora DB クラスターまたはグローバルクラスターの作成

Aurora DB クラスターまたはグローバルクラスターを作成する場合、コンソールで [RDS 延長サポートを有効にする] を選択するか、AWS CLI の拡張サポートオプションまたは RDS API のパラメータを使用します。

> **Note**
>
> RDS 延長サポートの設定を指定しない場合、Aurora はデフォルトで RDS 延長サポートに移行します。このデフォルトの動作により、Aurora の標準サポート終了日を過ぎてもデータベースの可用性が維持されます。

トピック

RDS 延長サポートに関する考慮事項

Aurora DB クラスターまたはグローバルクラスターを作成する前に、次の点を考慮してください。

- Aurora 標準サポート終了日が過ぎた後、新しい Aurora DB クラスターまたは新しいグローバルクラスターの作成を阻止し、RDS 延長サポート料金を回避できます。これには、AWS CLI または RDS API を使用します。AWS CLI の `--engine-lifecycle-support` オプションで、`open-source-rds-extended-support-disabled` と指定します。RDS API で、`LifeCycleSupport` パラメータに `open-source-rds-extended-support-disabled` を指定します。`open-source-rds-extended-support-disabled` と指定して、Aurora 標準サポート終了日が過ぎると、Aurora DB クラスター、またはグローバルクラスターの作成は常に失敗します。
- RDS 延長サポートはクラスターレベルで設定されます。クラスターのメンバーの RDS 延長サポート設定は、RDS コンソール、AWS CLI の `--engine-lifecycle-support`、RDS API の `EngineLifecycleSupport` で常に同じになります。

詳細については、「Amazon Aurora バージョン」を参照してください。

RDS 延長サポートで Aurora DB クラスターまたはグローバルクラスターを作成する

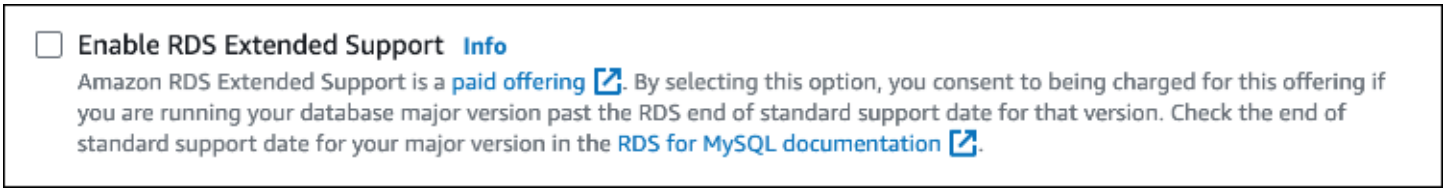
AWS Management Console、AWS CLI、または RDS API で RDS 延長サポートを使用して、Aurora DB クラスターまたはグローバルクラスターを作成する方法について説明します。

> **Note**
>
> AWS CLI の `--engine-lifecycle-support` オプションと RDS API の `EngineLifeCycle` パラメータは、現時点では Aurora PostgreSQL でのみ利用できます。Aurora 標準サポート終了日が近づくと、Aurora MySQL で利用できるようになります。

コンソール

Aurora DB クラスターまたはグローバルクラスターを作成する場合、[エンジンのオプション] セクションで、[RDS 延長サポートの有効化] を選択します。

次の画像は、[RDS 延長サポートを有効にする] 設定を示しています。



AWS CLI

create-db-cluster または create-global-cluster AWS CLI コマンドを使用する場合は、`--engine-lifecycle-support` オプションに `open-source-rds-extended-support` を指定して RDS 延長サポートを選択します。このオプションはデフォルトで `open-source-rds-extended-support` に設定されています。

Aurora 標準サポート終了日以降に、新しい Aurora DB クラスターまたはグローバルクラスターが作成されないようにするには、`--engine-lifecycle-support` オプションを `open-source-rds-extended-support-disabled` と指定します。これにより、関連する RDS 延長サポート料金は発生しません。

RDS API

CreateDBCluster または CreateGlobalCluster Amazon RDS API オペレーションを使用する場合は、`EngineLifecycleSupport` パラメータを `open-source-rds-extended-support` に設定して RDS 延長サポートを選択します。デフォルトでは、このパラメータは `open-source-rds-extended-support` に設定されます。

Aurora 標準サポート終了日以降に、 新しい Aurora DB クラスターまたはグローバルクラスターが作成されないようにするには、`EngineLifecycleSupport` パラメータに `open-source-rds-`

`extended-support-disabled` を指定します。これにより、関連する RDS 延長サポート料金は発生しません。

詳細については、次のトピックを参照してください。

- Aurora DB クラスターを作成するには、「Amazon Aurora DB クラスターの作成」の DB エンジンの手順に従ってください。
- グローバルクラスターを作成するには、「Amazon Aurora Global Database の作成」の DB エンジンの手順に従ってください。

Amazon RDS 延長サポートでの Aurora DB クラスターまたはグローバルクラスターの登録を確認します。

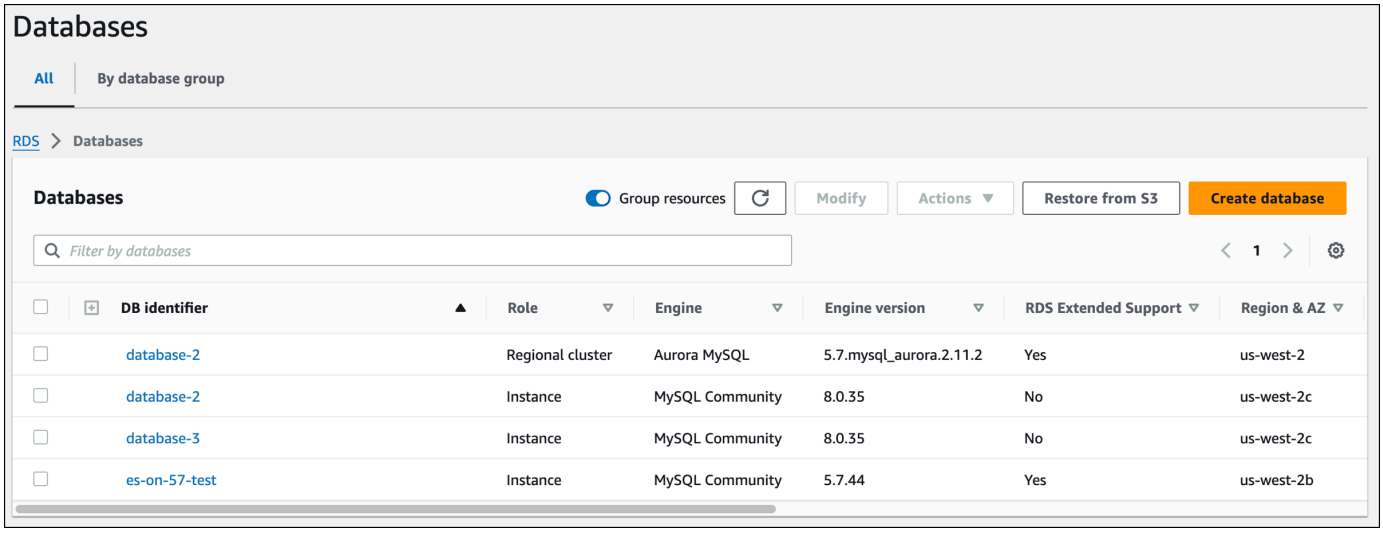
AWS Management Console を使用して、Amazon RDS 延長サポートでの Aurora DB クラスターまたはグローバルクラスターの登録を表示できます。

コンソール

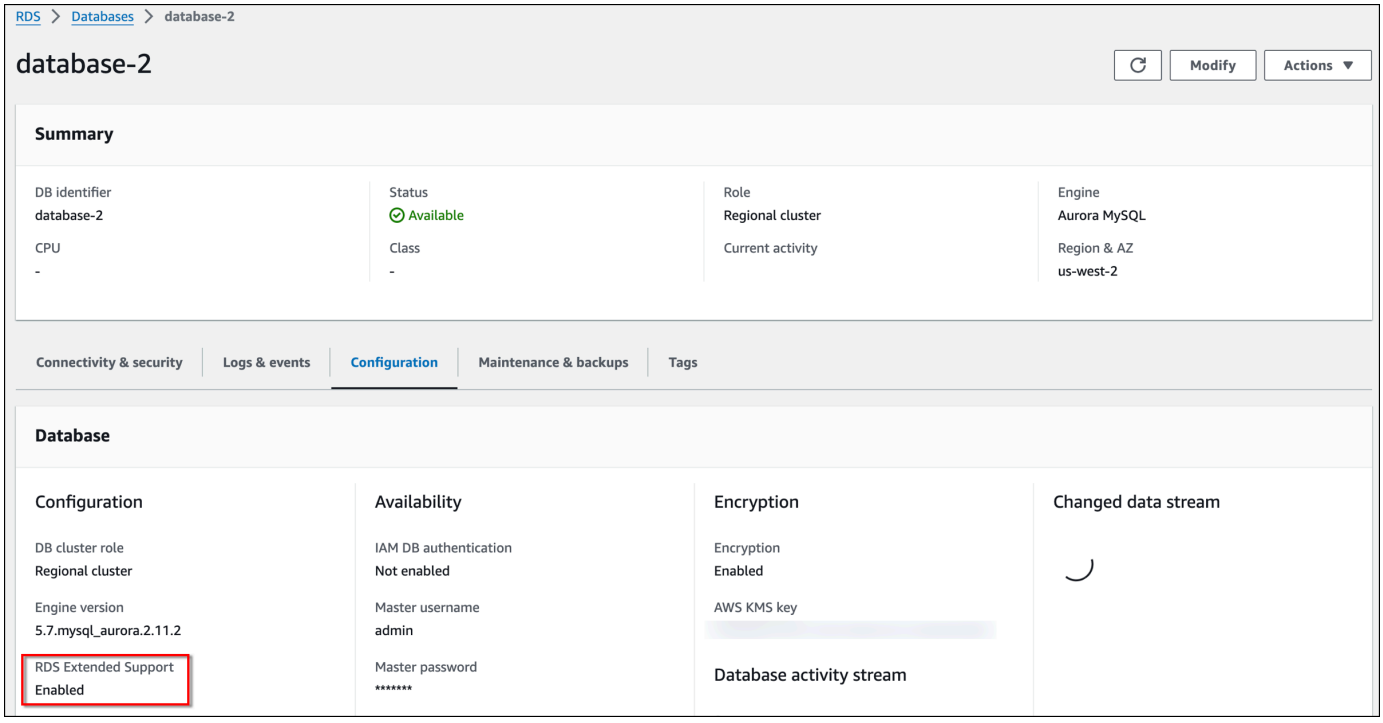
RDS 延長サポートでの Aurora DB クラスターまたはグローバルクラスターの登録を表示するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、データベースを選択します。[RDS 延長サポート] の値は、Aurora DB クラスターまたはグローバルクラスターが RDS 延長サポートに登録されているかどうかを示します。値が表示されない場合は、データベースの RDS 延長サポートを利用できません。

> **Tip**
>
> [RDS 延長サポート] 列が表示されない場合は、[設定] アイコンを選択し、[RDS 延長サポート] をオンにします。



3. 各データベースの [設定] タブで登録を表示することもできます。[DB 識別子] でデータベースを選択します。[設定] タブの [延長サポート] で、データベースが登録されているかどうかを確認します。



Amazon RDS 延長サポートでの Aurora DB クラスターまたはグローバルクラスターの復元

Aurora DB クラスターまたはグローバルクラスターを復元するときは、コンソールで [RDS 延長サポートを有効にする] を選択するか、AWS CLI の拡張サポートオプションまたは RDS API のパラメータを使用します。

> **Note**
>
> RDS 延長サポートの設定を指定しない場合、Aurora はデフォルトで RDS 延長サポートに移行します。このデフォルトの動作により、Aurora の標準サポート終了日を過ぎてもデータベースの可用性が維持されます。

トピック

RDS 延長サポートに関する考慮事項

Aurora DB クラスターまたはグローバルクラスターを復元する前に、次の点を考慮してください。

- Aurora 標準サポート終了日を過ぎた後で、、Aurora DB クラスター、またはグローバルクラスター を Amazon S3 から復元するには、AWS CLI または RDS API のみを使用できます。restore-db-cluster-from-s3 AWS CLI コマンドの `--engine-lifecycle-support` オプション、または RestoreDBClusterFromS3 RDS API オペレーションの `EngineLifecycleSupport` パラメータを使用します。
- Aurora によってデータベースが RDS 延長サポートバージョンに復元されないようにするには、AWS CLI または RDS API で `open-source-rds-extended-support-disabled` を指定します。これにより、関連する RDS 延長サポート料金は発生しません。

  この設定を指定すると、Amazon Aurora は復元されたデータベースを、サポートされている新しいメジャーバージョンに自動的にアップグレードします。アップグレードでアップグレード前の検証が失敗した場合、Amazon Aurora は安全に RDS 延長サポートエンジンのバージョンにロール

バックします。このデータベースは延長サポートモードのままとなり、Amazon Aurora ではデータベースを手動でアップグレードするまで RDS 延長サポートの料金が発生します。

- RDS 延長サポートはクラスターレベルで設定されます。クラスターのメンバーの RDS 延長サポート設定は、RDS コンソール、AWS CLI の `--engine-lifecycle-support`、RDS API の `EngineLifecycleSupport` で常に同じになります。

詳細については、「Amazon Aurora バージョン」を参照してください。

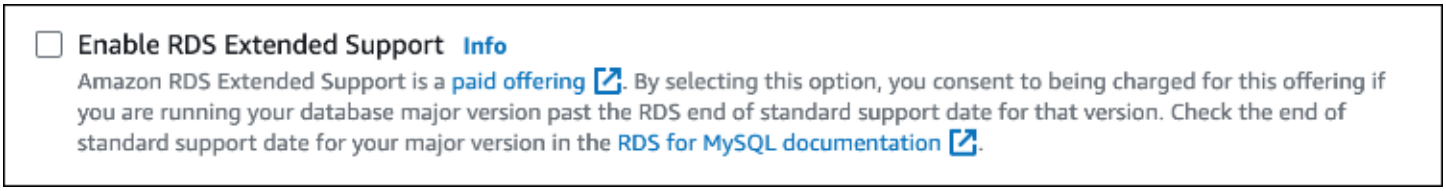
RDS 延長サポートを使用した Aurora DB クラスター DB クラスターまたはグローバルクラスターを復元する

RDS 延長サポートバージョンで Aurora DB クラスターまたはグローバルクラスターを復元するには、AWS Management Console、AWS CLI または RDS API を使用します。

コンソール

Aurora DB クラスターまたはグローバルクラスターを復元する際に、[エンジンオプション] セクションで [RDS 延長サポートを有効にする] を選択します。

次の画像は、[RDS 延長サポートを有効にする] 設定を示しています。



AWS CLI

restore-db-cluster-from-snapshot AWS CLI コマンドを使用する場合は、`--engine-lifecycle-support` オプションに `open-source-rds-extended-support` を指定して RDS 延長サポートを選択します。

RDS 延長サポートに関連する課金を避けたい場合は、`--engine-lifecycle-support` オプションを `open-source-rds-extended-support-disabled` に設定します。このオプションはデフォルトで `open-source-rds-extended-support` に設定されています。

以下の AWS CLI コマンドを使用してこの値を指定することもできます。

- restore-db-cluster-from-s3
- restore-db-cluster-to-point-in-time

RDS API

RestoreDBClusterFromSnapshot RDS API オペレーションを使用する場合は、EngineLifecycleSupport パラメータを open-source-rds-extended-support に設定して RDS 延長サポートを選択します。

RDS 延長サポートに関連する課金を避けたい場合は、EngineLifecycleSupport パラメータを open-source-rds-extended-support-disabled に設定します。デフォルトでは、このパラメータは open-source-rds-extended-support に設定されます。

以下の RDS API オペレーションを使用してこの値を指定することもできます。

- RestoreDBClusterFromS3
- RestoreDBClusterToPointInTime

Aurora DB クラスターの復元の詳細については、「Amazon Aurora DB クラスターのバックアップと復元」の該当 DB エンジンの手順に従ってください。

データベース更新のために Amazon RDS ブルー/グリーンデプロイを使用する

ブルー/グリーンデプロイは、本稼働データベース環境を別の同期されたステージング環境にコピーします。Amazon RDS ブルー/グリーンデプロイを使用すると、本稼働環境に影響を与えずに、ステージング環境のデータベースに変更を加えることができます。例えば、DB エンジンのメジャーまたはマイナーバージョンのアップグレード、データベースパラメータの変更、スキーマの変更をステージング環境で行うことができます。準備ができたら、ステージング環境を新しい本番稼働データベース環境に昇格でき、通常、ダウンタイムは 1 分未満です。

Amazon Aurora は、基盤となるブルー環境の Aurora ストレージボリュームをクローニングすることでステージング環境を作成します。ステージング環境のクラスターボリュームには、その環境に加えられた増分変更のみが保存されます。

> **Note**
>
> 現在、ブルー/グリーンデプロイは、Aurora MySQL および Aurora PostgreSQL Amazon RDS エンジンの可用性については、「Amazon RDS ユーザーガイド」の「データベースの更新のために Amazon RDS ブルー/グリーンデプロイを使用する」を参照してください。

トピック

- Aurora 用 Amazon RDS ブルー/グリーンデプロイの概要
- ブルー/グリーンデプロイの作成
- ブルー/グリーンデプロイの表示
- ブルー/グリーンデプロイの切り替え
- ブルー/グリーンデプロイの削除

Aurora 用 Amazon RDS ブルー/グリーンデプロイの概要

Amazon RDS ブルー/グリーンデプロイを使用すると、本番環境に実装する前に、データベースに変更を加えてテストできます。ブルー/グリーンのデプロイは、本稼働環境をコピーするステージング環境を作成します。ブルー/グリーンデプロイでは、ブルー環境が現在の本稼働環境です。グリーン環境はステージング環境です。ステージング環境は、論理レプリケーションを使用して現在の本稼働環境と同期したままになります。

本稼働環境のワークロードに影響を与えずに、グリーン環境の Aurora DB クラスターに変更を加えることができます。例えば、DB エンジンのメジャーまたはマイナーバージョンのアップグレード、、またはデータベースパラメータの変更をステージング環境で行うことができます。グリーン環境での変化を徹底的にテストできます。準備ができたら、環境を切り替えてグリーン環境を新しい本稼働環境にプロモートできます。切り替えには通常 1 分もかからず、データが失われることはなく、アプリケーションを変更する必要もありません。

グリーン環境は本稼働環境のトポロジのコピーであるため、DB クラスターとそのすべての DB インスタンスはデプロイにコピーされます。グリーン環境には、DB クラスタースナップショット、パフォーマンスインサイト、拡張モニタリング、Aurora Serverless v2 など、DB クラスターで使用される機能も含まれています。

> **Note**
>
> ブルー/グリーンデプロイは、Aurora MySQL と Aurora PostgreSQL でサポートされています。Amazon RDS の可用性については、「Amazon RDS ユーザーガイド」の「データベースの更新のために Amazon RDS ブルー/グリーンデプロイを使用する」を参照してください。

トピック

リージョンとバージョンの可用性

機能の可用性とサポートは、各データベースエンジンの特定のバージョン、および AWS リージョンによって異なります。詳細については、「the section called “ブルー/グリーンデプロイ”」を参照してください。

Amazon RDS ブルー/グリーンデプロイを使用する利点

Amazon RDS ブルー/グリーンデプロイを使用すると、セキュリティパッチを最新の状態に保ち、データベースのパフォーマンスを向上させ、短い予測可能なダウンタイムで新しいデータベース機能を導入できます。ブルー/グリーンデプロイでは、エンジンのメジャーバージョンまたはマイナーバージョンのアップグレードなど、データベース更新のリスクとダウンタイムが軽減されます。

ブルー/グリーンデプロイには次の利点があります。

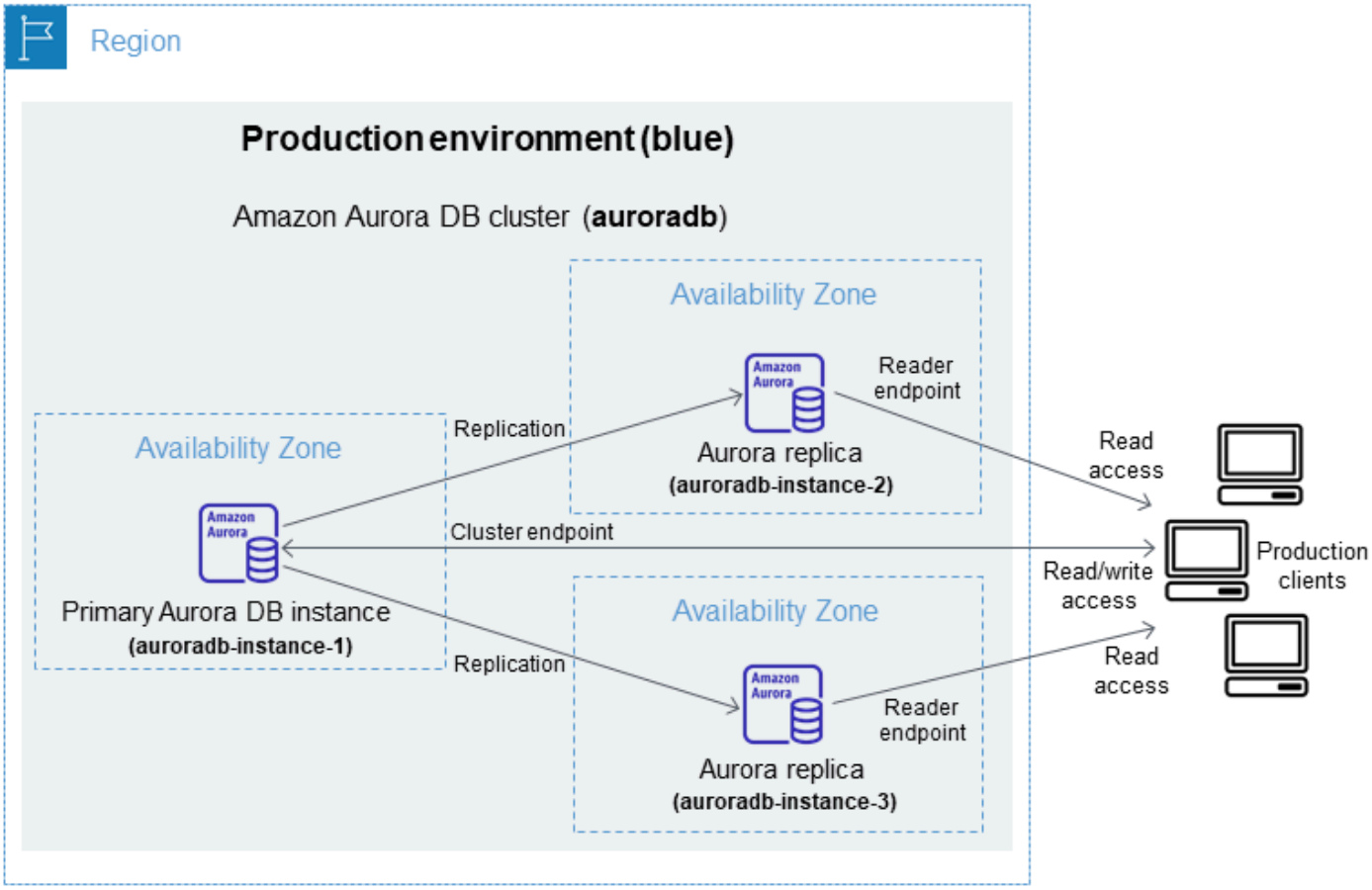
- 本稼働環境に対応したステージング環境を簡単に作成できます。
- データベースの変更を本稼働環境からステージング環境に自動的にレプリケートします。
- 本稼働環境に影響を与えずに、安全なステージング環境でデータベースの変更をテストします。
- データベースパッチとシステムアップデートを最新の状態に保ちます。
- 新しいデータベース機能を実装してテストします。
- アプリケーションを変更することなく、ステージング環境を新しい本稼働環境に切り替えます。
- 組み込みの切り替えガードレールを使用して安全に切り替えることができます。
- 切り替え中のデータ損失をなくします。
- ワークロードにもよりますが、通常は 1 分以内にすばやく切り替えることができます。

ブルー/グリーンデプロイのワークフロー

Aurora DB クラスターの更新にブルー/グリーンデプロイを使用する場合は、次の主要なステップを実行します。

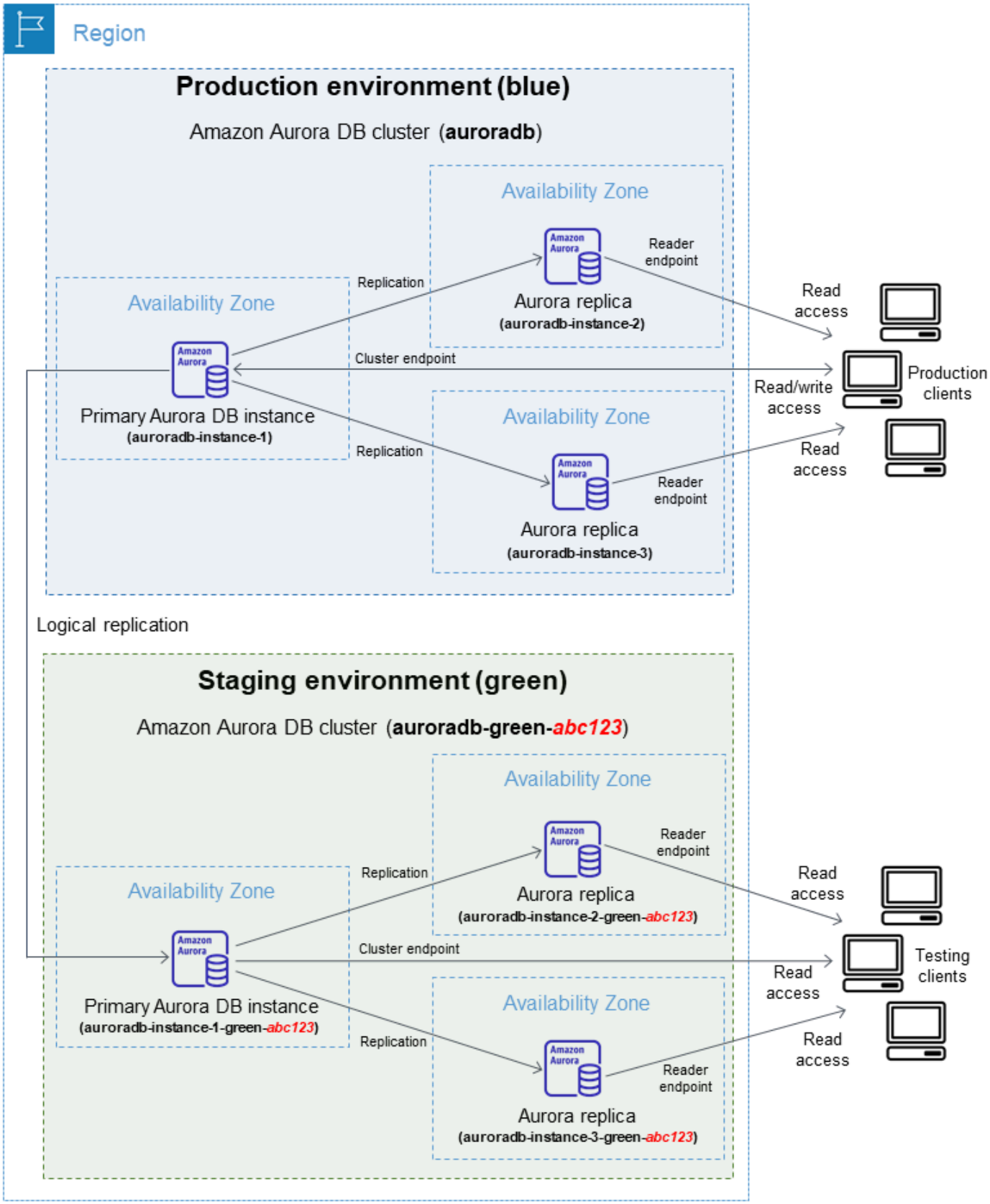
1. 更新が必要な本稼働 DB クラスターを特定します。

   次の図は、本稼働 DB クラスターの例を示しています。



2. ブルー/グリーンデプロイを作成します。手順については、ブルー/グリーンデプロイの作成 を参照してください。

   以下の図は、ステップ 1 の本稼働環境のブルー/グリーンデプロイの例を示しています。ブルー/グリーンデプロイを作成する際、RDS は Aurora DB クラスターのトポロジと構成全体をコピーしてグリーン環境を作成します。コピーされた DB クラスターと DB インスタンスの名前には `-green-random-characters` が付加されます。図のステージング環境には DB クラスター (auroradb-green-*abc123*) が含まれています。また、DB クラスター内の 3 つの DB インスタンス (auroradb-instance1-green-*abc123*、auroradb-instance2-green-*abc123*、auroradb-instance3-green-*abc123*) も含まれています。


Region
Production environment (blue)
Amazon Aurora DB cluster (auroradb)
Availability Zone
Aurora replica
(auroradb-instance-2)
Reader endpoint
Replication
Availability Zone
Primary Aurora DB instance
(auroradb-instance-1)
Cluster endpoint
Read access
Read/write access
Production clients
Availability Zone
Aurora replica
(auroradb-instance-3)
Replication
Reader endpoint
Read access
Logical replication
Staging environment (green)
Amazon Aurora DB cluster (auroradb-green-abc123)
Availability Zone
Aurora replica
(auroradb-instance-2-green-abc123)
Reader endpoint
Replication
Availability Zone
Primary Aurora DB instance
(auroradb-instance-1-green-abc123)
Cluster endpoint
Read access
Testing clients
Availability Zone
Aurora replica
(auroradb-instance-3-green-abc123)
Replication
Reader endpoint
Read access

ブルー/グリーンデプロイを作成すると、グリーン環境で DB クラスターにより高い DB エンジンのバージョンと別の DB パラメータグループを指定できます。DB クラスター内の DB インスタンスに別の DB パラメータグループを指定することもできます。

RDS は、ブルー環境のプライマリ DB インスタンスからグリーン環境のプライマリ DB インスタンスへのレプリケーションも設定します。

> **Important**
>
> Aurora MySQL バージョン 3 では、ブルー/グリーンデプロイを作成すると、グリーン環境の DB クラスターはデフォルトで書き込み操作を許可します。`read_only` パラメータ を `1` に設定し、クラスターを再起動して、DB クラスターを読み取り専用にすることをお勧めします。

3. ステージング環境に変更を加えます。

   例えば、データベースにスキーマの変更を加えたり、グリーン環境の 1 つ以上の DB インスタンスが使用する DB インスタンスクラスを変更したりすることができます。

   DB クラスターの変更については、「Amazon Aurora DB クラスターの変更」を参照してください。

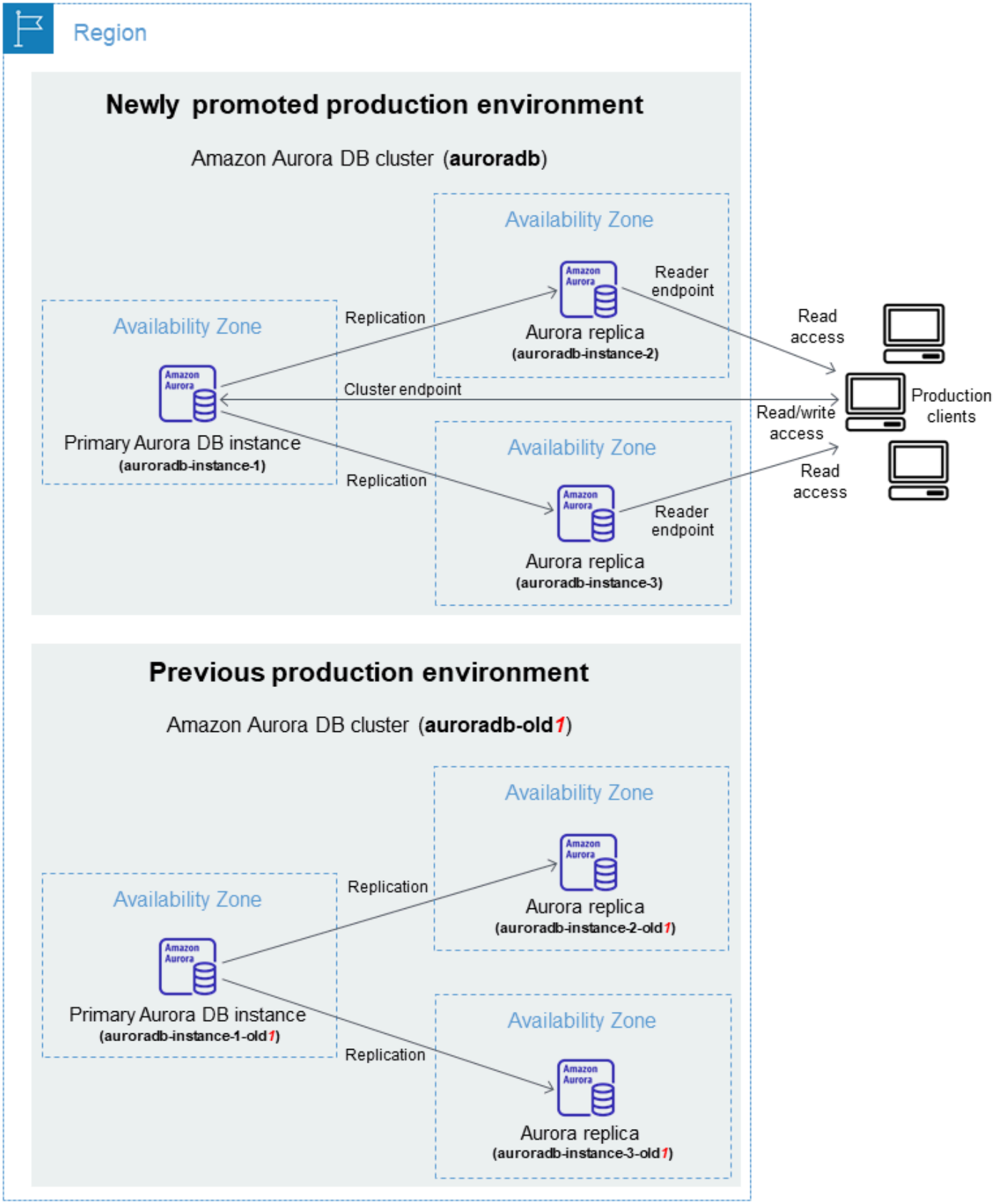
4. ステージング環境をテストします。

   テスト中は、グリーン環境のデータベースを読み取り専用に保つことをお勧めします。グリーン環境ではレプリケーションの競合が発生する可能性があるため、書き込み操作を有効にします。また、スイッチオーバー後に本稼働データベースに意図しないデータが発生する可能性もあります。Aurora MySQL の書き込み操作を有効にするには、`read_only` パラメータを `0` に設定し、DB インスタンスを再起動します。Aurora PostgreSQL の場合、セッションレベルで `default_transaction_read_only` パラメータを `off` に設定します。

5. 準備ができたら切り替えて、ステージング環境を昇格させ、新しい本稼働データベース環境にします。手順については、ブルー/グリーンデプロイの切り替え を参照してください。

   切り替えによりダウンタイムが発生します。ダウンタイムは通常 1 分未満ですが、ワークロードによってはさらに長くなることもあります。

   次の図は、切り替え後の DB クラスターを示しています。


Region
Newly promoted production environment
Amazon Aurora DB cluster (auroradb)
Availability Zone
Availability Zone
Replication
Aurora replica
(auroradb-instance-2)
Reader endpoint
Read access
Cluster endpoint
Production clients
Read/write access
Primary Aurora DB instance
(auroradb-instance-1)
Replication
Availability Zone
Read access
Reader endpoint
Aurora replica
(auroradb-instance-3)
Previous production environment
Amazon Aurora DB cluster (auroradb-old1)
Availability Zone
Availability Zone
Replication
Aurora replica
(auroradb-instance-2-old1)
Primary Aurora DB instance
(auroradb-instance-1-old1)
Availability Zone
Replication
Aurora replica
(auroradb-instance-3-old1)

切り替え後、グリーン環境の Aurora DB クラスターが新しい実稼働 DB クラスターになります。現在の本稼働環境の名前とエンドポイントは、新しく昇格した本稼働環境に割り当てられるため、アプリケーションを変更する必要はありません。その結果、本稼働トラフィックが新しい本稼働環境に流れるようになります。ブルー環境の DB クラスターと DB インスタンスは、現在の名前に `-old`*n* を付加することで名前が変更されます (*n* は数字です)。例えば、ブルー環境の DB インスタンスの名前が `auroradb-instance-1` であるとします。切り替え後、DB インスタンス名は `auroradb-instance-1-old1` になります。

図の例では、切り替え中に次の変更が行われます。

- グリーン環境の DB クラスター `auroradb-green-abc123` は、`auroradb` という名前の付いた本稼働用 DB クラスターになります。
- グリーン環境の `auroradb-instance1-green-abc123` という名前の DB インスタンスは、本稼働 DB インスタンス `auroradb-instance1` になります。
- グリーン環境の `auroradb-instance2-green-abc123` という名前の DB インスタンスは、本稼働 DB インスタンス `auroradb-instance2` になります。
- グリーン環境の `auroradb-instance3-green-abc123` という名前の DB インスタンスは、本稼働 DB インスタンス `auroradb-instance3` になります。
- ブルー環境の `auroradb` という名前の DB クラスターは `auroradb-old1` になります。
- ブルー環境の `auroradb-instance1` という名前の DB インスタンスは `auroradb-instance1-old1` になります。
- ブルー環境の `auroradb-instance2` という名前の DB インスタンスは `auroradb-instance2-old1` になります。
- ブルー環境の `auroradb-instance3` という名前の DB インスタンスは `auroradb-instance3-old1` になります。

6. 不要になったブルー/グリーンデプロイは削除できます。手順については、ブルー/グリーンデプロイの削除 を参照してください。

   切り替え後も以前の本稼働環境は削除されないため、必要に応じてリグレッションテストに使用できます。

ブルー/グリーンデプロイオペレーションへのアクセスの承認

ブルー/グリーンデプロイに関連する操作を実行するには、ユーザーは必要なアクセス許可があることが求められます。指定されたリソースに対して特定の API 操作を実行するために必要なアクセス

許可をユーザーとロールに付与する IAM ポリシーを作成できます。その後、これらのポリシーを、それらのアクセス許可を必要とする IAM アクセス許可セットまたはロールにアタッチできます。詳細については、「Amazon Aurora での Identity and Access Management」を参照してください。

ブルー/グリーンデプロイを作成するユーザーには、次の RDS オペレーションを実行するアクセス許可が必要です。

- `rds:AddTagsToResource`
- `rds:CreateDBCluster`
- `rds:CreateDBInstance`
- `rds:CreateDBClusterEndpoint`

ブルー/グリーンデプロイを切り替えるユーザーには、次の RDS オペレーションを実行するアクセス許可が必要です。

- `rds:ModifyDBCluster`
- `rds:PromoteReadReplicaDBCluster`

ブルー/グリーンデプロイを削除するユーザーには、次の RDS オペレーション (複数可) を実行するアクセス許可が必要です。

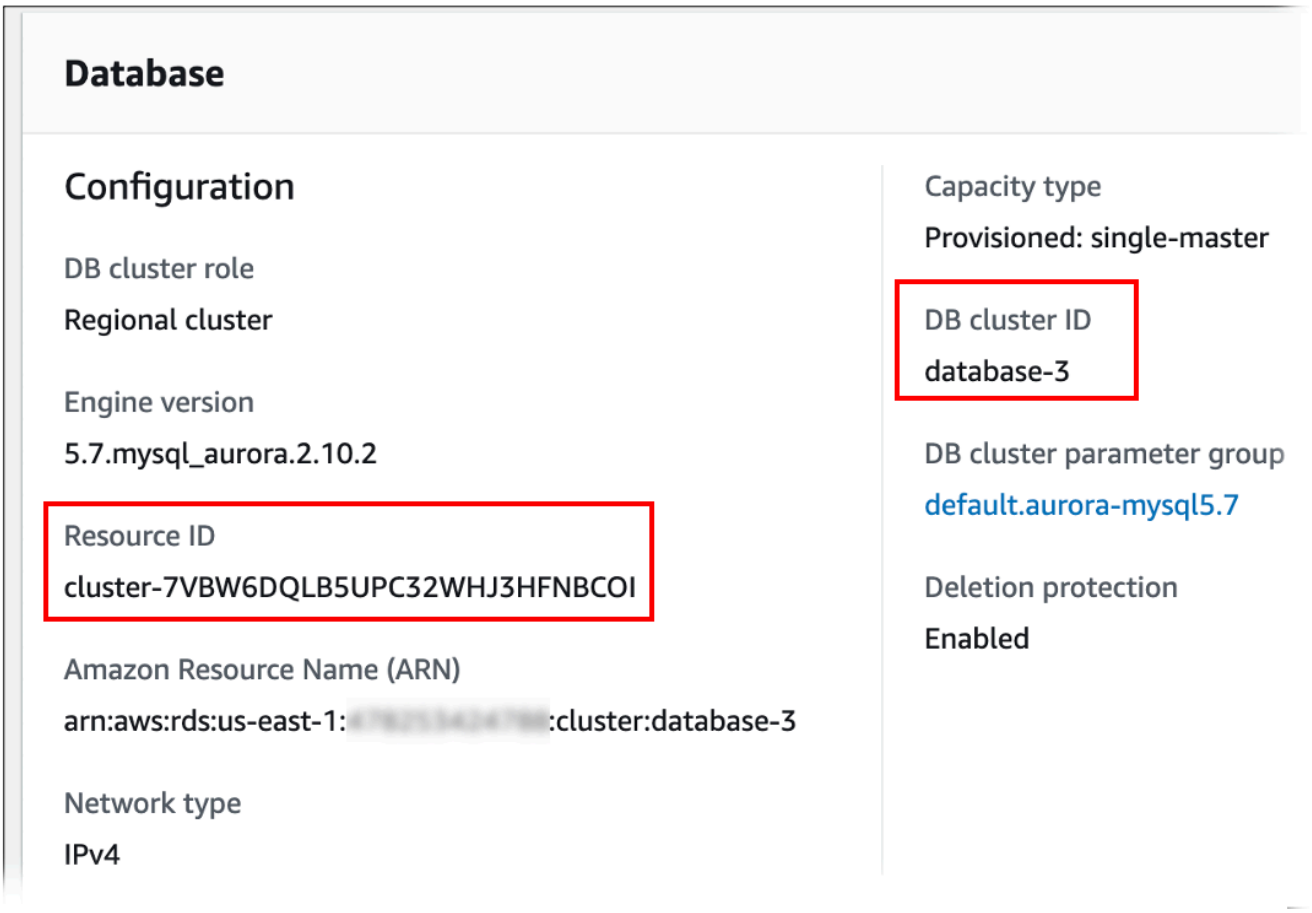
- `rds:DeleteDBCluster`
- `rds:DeleteDBInstance`
- `rds:DeleteDBClusterEndpoint`

Aurora は、お客様に代わってステージング環境のリソースをプロビジョニングおよび変更します。これらのリソースには、内部で定義された命名規則を使用する DB インスタンスが含まれます。したがって、アタッチされた IAM ポリシーには、`my-db-prefix-*` のような部分的なリソース名パターンを含めることはできません。ワイルドカード (*) のみがサポートされています。一般的に、これらのリソースへのアクセスを制御するには、ワイルドカードではなく、リソースタグやその他のサポートされている属性を使用することをおすすめします。詳細については、「Amazon RDS のアクション、リソース、および条件キー」を参照してください。

ブルー/グリーンデプロイの考慮事項

Amazon RDS は、ブルー/グリーンデプロイのリソースを各リソースの `DbiResourceId`および `DbClusterResourceId` で追跡します。このリソース ID は、リソースの AWS リージョン 固有のイミュータブルな識別子です。

リソース ID は、DB クラスター ID とは異なります。



ブルー/グリーンデプロイを切り替えると、リソースの名前 (クラスター ID) は変わりますが、各リソースは同じリソース ID を保持します。例えば、DB クラスター識別子がブルー環境で `mycluster` であったとします。切り替え後、同じ DB クラスターの名前が `mycluster-old1` に変更されたとします。ただし、DB クラスターのリソース ID は切り替え中に変更されません。そのため、グリーンリソースを新しい本稼働用リソースに昇格しても、そのリソース ID は以前に本稼働にあったブルーリソース ID と一致しません。

ブルー/グリーンデプロイに切り替えたら、本稼働用リソースで使用していた統合機能やサービスのリソース ID を新たにプロモートされた本稼働用リソースのものに更新することを検討してください。具体的には、次のような更新を検討してください。

- RDS API とリソース ID を使用してフィルタリングを実行する場合は、切り替え後にフィルタリングに使用されるリソース ID を調整します。
- CloudTrail をリソースの監査に使用する場合は、切り替え後に新しいリソース ID を追跡するように CloudTrail のコンシューマーを調整します。詳細については、「AWS CloudTrail での Amazon Aurora API コールのモニタリング」を参照してください。
- ブルー環境のリソースにデータベースアクティビティストリームを使用する場合は、切り替え後に新しいストリームのデータベースイベントをモニタリングするようにアプリケーションを調整します。詳細については、「データベースアクティビティストリームでサポートされているリージョンと Aurora DB エンジン」を参照してください。
- パフォーマンスインサイト API を使用する場合は、切り替え後に API への呼び出しでリソース ID を調整します。詳細については、「Amazon Aurora での Performance Insights を使用したDB 負荷のモニタリング」を参照してください。

  切り替え後に同じ名前のデータベースをモニタリングできますが、切り替え前のデータは含まれていません。
- IAM ポリシーでリソース ID を使用する場合は、必要に応じて新しく昇格したリソースのリソース ID を追加します。詳細については、「Amazon Aurora での Identity and Access Management」を参照してください。
- DB クラスターに IAM ロールが関連付けられている場合は、スイッチオーバー後にそれらを再度関連付けます。アタッチされたロールはグリーンの環境に自動的にコピーされません。
- IAM データベース認証を使用して DB クラスターを認証する場合は、データベースアクセスに使用する IAM ポリシーの `Resource` 要素の下にブルーとグリーンのデータベースの両方が表示されていることを確認してください。これは、スイッチオーバー後にグリーンのデータベースに接続するために必要です。詳細については、「the section called “IAM データベースアクセス用の IAM ポリシーの作成と使用”」を参照してください。
- ブルー/グリーンデプロイに含まれていた DB クラスターの手動 DB クラスタースナップショットを復元する場合は、スナップショットが取られた時間を調べて、正しい DB クラスタースナップショットを復元します。詳細については、「DB クラスターのスナップショットからの復元」を参照してください。
- Amazon Aurora は、基盤となるブルー環境の Aurora ストレージボリュームをクローニングすることでグリーン環境を作成します。グリーンのクラスターボリュームには、グリーン環境に加えられた増分変更のみが保存されます。ブルー環境の DB クラスターを削除すると、グリーン環境の基盤となる Aurora ストレージボリュームのサイズは、フルサイズまで増大します。詳細については、「the section called “Aurora DB クラスターのボリュームのクローン作成”」を参照してください。

- ブルー/グリーンデプロイのグリーン環境の DB クラスターに DB インスタンスを追加すると、切り替え時にブルー環境の DB インスタンスが新しい DB インスタンスに置き換わることはありません。ただし、新しい DB インスタンスは DB クラスターに残り、新しい本稼働環境の DB インスタンスになります。
- ブルー/グリーンデプロイのグリーン環境で DB クラスターの DB インスタンスを削除すると、ブルー/グリーンデプロイで代わりになる新しい DB インスタンスを作成することはできません。

  削除した DB インスタンスと同じ名前と ARN で新しい DB インスタンスを作成する場合、`DbiResourceId` が異なるため、グリーン環境には含まれません。

  グリーン環境で DB クラスター内の DB インスタンスを削除すると、次のような動作になります。

  - ブルー環境に同じ名前の DB インスタンスが存在する場合、グリーン環境の DB インスタンスに切り替わりません。この DB インスタンスの名前は、DB インスタンス名に `-old`*n* を付加することによって変更されません。
  - ブルー環境の DB インスタンスを参照するアプリケーションは、切り替え後も同じ DB インスタンスを引き続き使用します。

ブルー/グリーンデプロイのベストプラクティス

ブルー/グリーンデプロイのベストプラクティスを以下に示します。

一般的なベストプラクティス

- 切り替える前に、Aurora DB クラスターをグリーン環境で十分にテストしてください。
- グリーン環境のデータベースは読み取り専用のまま維持してください。グリーン環境ではレプリケーションの競合が発生する可能性があるため、書き込み操作の有効化には注意することをお勧めします。また、スイッチオーバー後に本稼働データベースに意図しないデータが発生する可能性もあります。
- ブルー/グリーンデプロイを使用してスキーマの変更を実装する場合は、レプリケーション互換の変更のみを行ってください。

  例えば、ブルーデプロイからグリーンデプロイへのレプリケーションを中断することなく、テーブルの最後に新しい列を追加することができます。ただし、列名の変更やテーブル名の変更などのスキーマの変更は、グリーンデプロイへのレプリケーションを中断させます。

レプリケーションと互換性のある変更の詳細については、MySQL ドキュメントの「ソースとレプリカのテーブル定義が異なるレプリケーション」と PostgreSQL 論理レプリケーションドキュメントの「制限」を参照してください。

- 両方の環境のすべての接続に、クラスターエンドポイント、リーダーエンドポイント、またはカスタムエンドポイントを使用します。静的リストまたは除外リストのあるインスタンスエンドポイントやカスタムエンドポイントは使用しないでください。
- ブルー/グリーンデプロイメントを切り替えるときは、切り替えのベストプラクティスに従ってください。詳細については、「the section called “切り替えのベストプラクティス”」を参照してください。

Aurora PostgreSQL のベストプラクティス

- Aurora PostgreSQL 論理レプリケーション書き込みスルーキャッシュをモニタリングし、必要に応じてキャッシュバッファを調整します。詳細については、「the section called “論理レプリケーションライトスルーキャッシュのモニタリング”」を参照してください。
- データベースが に十分な空きメモリがある場合は、ブルー環境の `logical_decoding_work_mem` DB パラメータの値を増やします。そうすることで、ディスク上でのデコード回数が減り、代わりにメモリを消費します。`FreeableMemory` CloudWatch メトリクスを使用して空きメモリをモニタリングできます。詳細については、「the section called “Aurora の CloudWatch メトリクス”」を参照してください。
- ブルー/グリーンデプロイを作成するときには、すべての PostgreSQL 拡張機能を最新バージョンに更新してください。詳細については、「the section called “PostgreSQL 拡張機能のアップグレード”」を参照してください。
- `aws_s3` 拡張機能を使用している場合は、グリーン環境が作成された後に、必ず IAM ロールを通じてグリーン DB クラスターに Amazon S3 へのアクセスを許可してください。これにより、インポートコマンドとエクスポートコマンドは、スイッチオーバー後も機能し続けることができます。手順については、the section called “Amazon S3 バケットへのアクセスを設定する” を参照してください。
- グリーン環境により上位のエンジンバージョンを指定する場合は、すべてのデータベースに対して `ANALYZE` オペレーションを実行して `pg_statistic` テーブルを更新します。Optimizer の統計情報はメジャーバージョンのアップグレード時に転送されないため、パフォーマンスの問題を回避するためにすべての統計情報を再生成する必要があります。メジャーバージョンアップグレード時のその他のベストプラクティスについては、「the section called “メジャーバージョンのアップグレードを実施する方法”」を参照してください。

- ソースでトリガーを使用してデータを操作している場合は、トリガーを ENABLE REPLICA または ENABLE ALWAYS として設定しないでください。レプリケーションシステムが変更を伝播してトリガーを実行し、重複を引き起こします。
- トランザクションの実行時間が長いと、レプリカの遅延が大きくなる可能性があります。レプリカの遅延を減らすには、以下を検討します。
  - グリーン環境がブルー環境に追いつくまで、遅延が発生する可能性があり長時間実行されるトランザクションを減らします。
  - ブルー/グリーンデプロイを作成する前に、ビジーテーブルで手動バキュームフリーズオペレーションを開始します。
  - PostgreSQL バージョン 12 以降では、大きなテーブルまたはビジーテーブルで index_cleanupパラメータを無効にして、ブルーデータベースの通常のメンテナンスのレートを高くします。
- レプリケーションが遅い場合、送信者と受信者が頻繁に再起動し、同期が遅れる可能性があります。送信者と受信者が再起動しないようにするには、ブルー環境で wal_sender_timeout パラメータを 0 に設定し、グリーン環境で wal_receiver_timeout パラメータを 0 に設定してタイムアウトを無効にします。

ブルー/グリーンデプロイの制限事項

ブルー/グリーンデプロイには、次の制約事項が適用されます。

トピック

- ブルー/グリーンデプロイの一般的な制約事項
- ブルー/グリーンデプロイの PostgreSQL 拡張機能の制約事項
- ブルー/グリーンデプロイの変更の制約事項
- ブルー/グリーンデプロイの PostgreSQL 論理レプリケーションの制約事項

ブルー/グリーンデプロイの一般的な制約事項

ブルー/グリーンデプロイには、次の一般的な制約事項が適用されます。

- Aurora MySQL バージョン 2.08 と 2.09 は、アップグレードのソースバージョンまたはターゲットバージョンとしてはサポートされません。
- ブルー/グリーンデプロイの一部であるクラスターを停止および開始することはできません。

- ブルー/グリーンデプロイでは、AWS Secrets Manager を使用したマスターユーザーのパスワード管理はサポートされていません。
- バックトラックが有効になっている Aurora MySQL ソース DB クラスターからブルー/グリーンデプロイを作成すると、グリーン DB クラスターはバックトラックのサポートなしで作成されます。バックトラックは、ブルー/グリーンデプロイに必須のバイナリログ (binlog) レプリケーションでは機能しないためです。詳細については、「the section called “DB クラスターのバックトラック”」を参照してください。

  ブルー DB クラスターでバックトラックを強制実行しようとすると、ブルー/グリーンデプロイが中断され、スイッチオーバーがブロックされます。
- Aurora MySQL の場合、ソース DB クラスターには `tmp` という名前のデータベースを含めることはできません。この名前のデータベースはグリーン環境にコピーされません。
- Aurora PostgreSQL では、ブルー DB クラスターで `rds.logically_replicate_unlogged_tables` パラメータが 1 に設定されていない限り、ログに記録されていないテーブルはグリーン環境にレプリケートされません。ブルー/グリーンデプロイを作成した後は、ログに記録されていないテーブルで発生する可能性のあるレプリケーションエラーを回避するために、このパラメータ値を変更しないことをお勧めします。
- Aurora PostgreSQL では、ブルー環境の DB クラスターを自己管理の論理ソース (パブリッシャー) またはレプリカ (サブスクライバー) にすることはできません。Aurora MySQL では、ブルー環境の DB クラスターを外部バイナリログレプリカにすることはできません。
- 切り替え中、ブルー環境とグリーン環境では Amazon Redshift とのゼロ ETL 統合はできません。最初に統合を削除してから切り替えて、統合を再作成する必要があります。
- ブルー/グリーンデプロイを作成するときには、グリーン環境でイベントスケジューラー (`event_scheduler` パラメーター) を無効にする必要があります。これにより、グリーン環境でイベントが生成されて不整合が発生するのを防ぐことができます。
- ブルーの DB クラスターで定義されている Aurora Auto Scaling ポリシーはグリーンの環境にコピーされません。
- ブルー/グリーンデプロイは MySQL 用の AWS JDBC ドライバーをサポートしていません。詳細については、GitHub の「既知の制限事項」を参照してください。
- ブルー/グリーンデプロイは、以下の機能ではサポートされていません。
  - Amazon RDS Proxy
  - クロスリージョンリードレプリカ
  - Aurora Serverless v1 DB クラスター
  - Aurora Global Database の一部である DB クラスター。

- Babelfish for Aurora PostgreSQL
- AWS CloudFormation

ブルー/グリーンデプロイの PostgreSQL 拡張機能の制約事項

以下の制約事項が PostgreSQL 拡張機能に適用されます。

- ブルー/グリーンデプロイを作成するときには、ブルー環境で pg_partman 拡張機能を無効にする必要があります。この拡張機能は CREATE TABLE などの DDL オペレーションを実行します。これは、ブルー環境からグリーン環境への論理レプリケーションを中断します。
- ブルー/グリーンデプロイを作成した後は、すべてのグリーンデータベースで pg_cron 拡張機能を無効のままにしておく必要があります。この拡張機能には、スーパーユーザーとして実行されるバックグラウンドワーカーがあり、グリーン環境の読み取り専用設定をバイパスするため、レプリケーションの競合が発生する可能性があります。
- ブルー環境で同じプランが取り込まれた場合、プライマリキーの競合が発生しないように、apg_plan_mgmt 拡張機能の apg_plan_mgmt.capture_plan_baselines パラメータをすべてのグリーンデータベースで off に設定する必要があります。詳細については、「the section called “Aurora PostgreSQL のクエリ計画管理の概要”」を参照してください。

  Aurora Replicas で実行プランをキャプチャする場合は、apg_plan_mgmt.create_replica_plan_capture 関数を呼び出すときにブルー DB クラスターエンドポイントを指定する必要があります。これにより、プランのキャプチャはスイッチオーバー後も引き続き機能します。詳細については、「the section called “レプリカでの Aurora PostgreSQL 実行プランのキャプチャ”」を参照してください。
- ブルー DB クラスターが外部データラッパー (FDW) 拡張機能の外部サーバーとして設定されている場合は、IP アドレスの代わりにクラスターエンドポイント名を使用する必要があります。これにより、スイッチオーバー後も設定は機能し続けます。
- ブルー/グリーンデプロイを作成するときには、ブルー環境で pglogical および pg_active 拡張機能を無効にする必要があります。グリーン環境を新しい本番環境に昇格させたら、拡張機能を再度有効にできます。また、ブルーデータベースは外部インスタンスの論理サブスクライバーにはなれません。
- pgAudit 拡張機能を使用している場合は、ブルー DB インスタンスとグリーン DB インスタンスの両方のカスタム DB パラメータグループの共有ライブラリ (shared_preload_libraries) に残っている必要があります。詳細については、「the section called “pgAudit 拡張機能のセットアップ”」を参照してください。

ブルー/グリーンデプロイの変更の制約事項

ブルー/グリーンデプロイの変更に関する制約事項を次に示します。

- 暗号化されていない DB クラスターを暗号化された DB クラスターに変更することはできません。
- 暗号化された DB クラスターを暗号化されていない DB クラスターに変更することはできません。
- ブルー環境の DB クラスターを、対応するグリーン環境の DB クラスターよりも上位のエンジンバージョンに変更することはできません。
- ブルー環境とグリーン環境のリソースは同じ AWS アカウント にある必要があります。
- ブルー環境に Aurora Auto Scaling ポリシーが含まれている場合、これらのポリシーはグリーンの環境にコピーされません。グリーン環境にはポリシーを手動で再追加する必要があります。

ブルー/グリーンデプロイの PostgreSQL 論理レプリケーションの制約事項

ブルー/グリーンデプロイでは、論理レプリケーションを使用してステージング環境と運用環境の同期を保ちます。PostgreSQL には、論理レプリケーションに関連する特定の制約事項があります。これは、Aurora PostgreSQL DB クラスター のブルー/グリーンデプロイを作成する際の制約となります。

次の表では、Aurora PostgreSQL のブルー/グリーンデプロイメントに適用される論理レプリケーションの制約事項について説明しています。

制限	説明
`CREATE TABLE` や `CREATE SCHEMA` などのデータ定義言語 (DDL) ステートメントは、ブルー環境からグリーン環境にはレプリケートされません。	Aurora がブルーの環境で DDL の変更を検出すると、グリーンデータベースはレプリケーションが低下した状態になります。 ブルー環境での DDL の変更を緑の環境にレプリケートできないことを通知するイベントを受け取ります。ブルー/グリーンデプロイとすべてのグリーンデータベースを削除してから再作成する必要があります。そうしないと、ブルー/グリーンデプロイに切り替えることができません。
シーケンスオブジェクトに対す	スイッチオーバー中、Aurora はグリーン環境のシーケンス値をブルー環境のシーケンス値と一致するようにインクリメントします。シーケンスが数千ある場合、スイッチオーバーが遅れる可能性があります。

制限	説明
る NEXTVAL オペレーションは、ブルー環境とグリーン環境では同期されません。	
ブルー環境での大きなオブジェクトの作成や変更は、グリーン環境にはレプリケートされません。	Aurora が、pg_largeobject システムテーブルに保存されているブルー環境でラージオブジェクトの作成または変更を検出すると、グリーンデータベースはレプリケーションが低下したの状態になります。 Aurora は、ブルー環境での大きなオブジェクトの変化をグリーン環境にレプリケートできないことを通知するイベントを生成します。ブルー/グリーンデプロイとすべてのグリーンデータベースを削除してから再作成する必要があります。そうしないと、ブルー/グリーンデプロイに切り替えることができません。
マテリアライズドビューはグリーン環境では自動的に更新されません。	ブルー環境でマテリアライズドビューを更新しても、グリーン環境では更新されません。スイッチオーバー後、マテリアライズドビューの更新をスケジュールできます。
UPDATE および DELETE オペレーションは、プライマリキーのないテーブルでは許可されません。	ブルー/グリーンデプロイを作成する前に、DB クラスター のすべてのテーブルにプライマリキーがあることを確認してください。

詳細については、PostgreSQL の論理レプリケーションドキュメントの「制限」を参照してください。

ブルー/グリーンデプロイの作成

ブルー/グリーンデプロイを作成するときには、デプロイにコピーする DB クラスターを指定します。選択する DB クラスターは本番 DB クラスターであり、ブルー環境の DB クラスターになりま

す。RDS は、ブルー環境のトポロジを、設定済みの機能とともにステージングエリアにコピーします。この DB クラスターはグリーン環境にコピーされ、RDS はブルー環境の DB クラスターからグリーン環境の DB クラスターへのレプリケーションを設定します。RDS は、DB クラスター内のすべての DB インスタンスもコピーします。

トピック

- ブルー/グリーンデプロイの準備
- ブルー/グリーンデプロイを作成するときの変更を指定
- ブルー/グリーンデプロイの作成

ブルー/グリーンデプロイの準備

Aurora DB クラスターが実行しているエンジンに応じて、ブルー/グリーンデプロイを作成する前に実行する必要がある特定の手順があります。

トピック

- ブルー/グリーンデプロイ用 Aurora MySQL DB クラスターの準備
- ブルー/グリーンデプロイ用 Aurora PostgreSQL DB クラスターの準備

ブルー/グリーンデプロイ用 Aurora MySQL DB クラスターの準備

Aurora MySQL DB クラスターのブルー/グリーンデプロイを作成する前に、バイナリログ (`binlog_format`) がオンになっているカスタム DB クラスターパラメータグループにクラスターを関連付ける必要があります。ブルー環境からグリーン環境へのレプリケーションには、バイナリログが必要です。どのバイナリログ形式でも使用できますが、複製の不整合のリスクを減らすには、`ROW` をお勧めします。カスタム DB クラスターパラメータグループの作成とパラメータの設定については、「the section called “DB クラスターパラメータグループを使用する”」を参照してください。

> **Note**
>
> バイナリログを有効にすると、DB クラスターへの書き込みディスク I/O 操作の回数が増えます。`VolumeWriteIOPs` CloudWatch メトリクスを使用して、IOPS の使用状況をモニタリングできます。

バイナリロギングを有効にした後は、必ず DB クラスターを再起動して変更を有効にしてください。ブルー/グリーンデプロイでは、ライターインスタンスが DB クラスターパラメータグループと同期している必要があり、同期していない場合は作成に失敗します。詳細については、「Aurora クラスター内の DB インスタンスの再起動」を参照してください。

さらに、バイナリログファイルが消去されないように、バイナリログの保持期間を NULL 以外の値に変更することをお勧めします。詳細については、「the section called “設定”」を参照してください。

ブルー/グリーンデプロイ用 Aurora PostgreSQL DB クラスターの準備

Aurora PostgreSQL DB クラスターのブルー/グリーンデプロイを作成する前に、以下を実行するようにしてください。

- クラスターを、論理レプリケーション (rds.logical_replication) が有効になっているカスタム DB クラスターパラメータグループに関連付けます。ブルー環境からグリーン環境へのレプリケーションには、論理レプリケーションが必要です。

  論理レプリケーションを有効にする場合は、max_replication_slots、max_logical_replication_workers、max_worker_processes などの特定のクラスターパラメータも調整する必要があります。論理レプリケーションを有効にしてこれらのパラメータを調整する手順については、「the section called “論理レプリケーションの設定”」を参照してください。

  また、synchronous_commit パラメータが on に設定されていることも確認してください。

  必要なパラメータを設定したら、変更が有効になるように DB クラスターを再起動してください。ブルー/グリーンデプロイでは、ライターインスタンスが DB クラスターパラメータグループと同期している必要があり、同期していない場合は作成に失敗します。詳細については、「Aurora クラスター内の DB インスタンスの再起動」を参照してください。
- DB クラスターがブルー/グリーンデプロイと互換性のあるバージョンの Aurora PostgreSQL を実行していることを確認してください。互換性のあるバージョンの一覧については、「the section called “Aurora PostgreSQL によるブルー/グリーンデプロイ”」を参照してください。
- DB クラスターのすべてのテーブルにプライマリキーがあることを確認します。PostgreSQL の論理レプリケーションでは、プライマリキーのないテーブルに対する UPDATE または DELETE オペレーションは許可されません。
- トリガーを使用する場合は、名前が「rds」で始まる pg_catalog.pg_publication、pg_catalog.pg_subscription、pg_catalog.pg_replicatio オブジェクトの作成、更新、削除を妨げないようにしてください。

ブルー/グリーンデプロイを作成するときの変更を指定

ブルー/グリーンデプロイを作成するときに、グリーン環境の DB クラスターに次の変更を加えることができます。

デプロイ後に、グリーン環境の DB クラスターとその DB インスタンスに他の変更を加えることができます。例えば、データベースにスキーマの変更を加えたり。

DB クラスターの変更については、「Amazon Aurora DB クラスターの変更」を参照してください。

より高いエンジンバージョンを指定する

DB エンジンのアップグレードをテストする場合は、上位のエンジンバージョンを指定できます。スイッチオーバー時に、データベースは指定したメジャーまたはマイナー DB エンジンバージョンにアップグレードされます。

別の DB パラメータグループを指定する

DB クラスターで使用されているものとは異なる DB クラスターパラメータグループを指定します。パラメータの変更がグリーン環境の DB クラスターにどのように影響するかをテストしたり、アップグレードの場合に新しいメジャー DB エンジンバージョンのパラメータグループを指定したりできます。

別の DB クラスターパラメータグループを指定した場合、指定されたパラメータグループがグリーン環境内の DB クラスターに関連付けられます。別の DB クラスターパラメータグループを指定しなかった場合、グリーン環境の DB クラスターはブルー DB クラスターと同じパラメータグループに関連付けられます。

ブルー/グリーンデプロイの作成

ブルー/グリーンデプロイは、AWS Management Console、AWS CLI、または RDS API を使用して作成できます。

コンソール

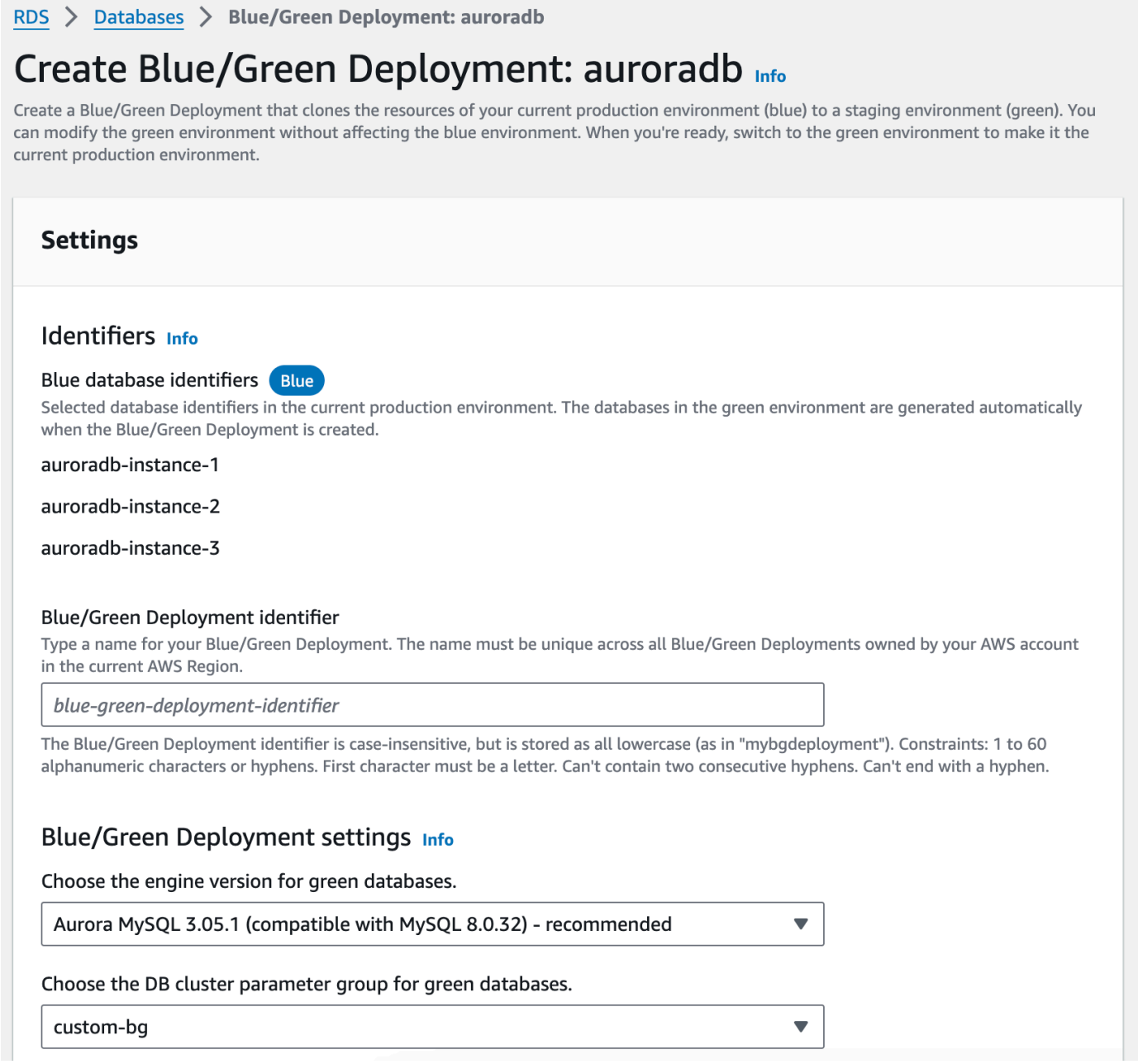
ブルー/グリーンデプロイを作成するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[Databases] (データベース) を選択し、グリーン環境にコピーする DB クラスターを選択します。

3. [アクション] で [ブルー/グリーンデプロイの作成] を選択します。

   Aurora PostgreSQL DB クラスター を選択する場合は、論理レプリケーションの制限を確認して承認してください。詳細については、「the section called “PostgreSQL 論理レプリケーションの制約事項”」を参照してください。

   [Create Blue/Green Deployment] (ブルー/グリーンデプロイの作成) ページが表示されます。



4. ブルーデータベース識別子を確認します。ブルー環境で予期される DB インスタンスに一致することを確認します。一致しない場合は、[Cancel] (キャンセル) を選択します。
5. ブルー/グリーンデプロイ識別子として、ブルー/グリーンデプロイの名前を入力します。

6. (オプション) [Blue/Green Deployment settings] (ブルー/グリーンデプロイ設定) では、グリーン環境の設定を指定します。

   - DB エンジンバージョンのアップグレードをテストするには、DB エンジンのバージョンを選択します。
   - グリーン環境内の DB クラスターに関連付ける DB クラスターパラメータグループを選択します。
   - グリーン環境内の DB インスタンスに関連付ける DB パラメータグループを選択します。

   グリーン環境のデータベースには、デプロイ後に他の変更を加えることができます。
7. [ステージング環境の作成] を選択します。

AWS CLI

AWS CLI を使用してブルー/グリーンデプロイを作成するには、create-blue-green-deployment コマンドを次のオプションを指定して使用します。

- `--blue-green-deployment-name` — ブルー/グリーンデプロイの名前を指定します。
- `--source` — コピーする DB クラスターの ARN を指定します。
- `--target-engine-version` — グリーン環境で DB エンジンのバージョンアップグレードをテストする場合は、エンジンバージョンを指定します。このオプションは、グリーン環境の DB クラスターを指定された DB エンジンバージョンにアップグレードします。

  指定しなかった場合、グリーン環境の DB クラスターは、ブルー環境の DB クラスターと同じエンジンバージョンで作成されます。
- `--target-db-cluster-parameter-group-name` – グリーン環境内の DB クラスターに関連付ける DB クラスターパラメータグループを指定します。
- `--target-db-parameter-group-name` – グリーン環境内の DB インスタンスに関連付ける DB パラメータグループを指定します。

Example ブルー/グリーンデプロイを作成する

Linux、macOS、Unix の場合:

```
aws rds create-blue-green-deployment \
    --blue-green-deployment-name aurora-blue-green-deployment \
```

```
    --source arn:aws:rds:us-east-2:123456789012:cluster:auroradb \
    --target-engine-version 8.0 \
    --target-db-cluster-parameter-group-name mydbclusterparametergroup
```

Windows の場合:

```
aws rds create-blue-green-deployment ^
    --blue-green-deployment-name aurora-blue-green-deployment ^
    --source arn:aws:rds:us-east-2:123456789012:cluster:auroradb ^
    --target-engine-version 8.0 ^
    --target-db-cluster-parameter-group-name mydbclusterparametergroup
```

RDS API

Amazon RDS API を使用してブルー/グリーンデプロイを作成するには、以下のパラメータを指定して `CreateBlueGreenDeployment` オペレーションを使用します。

- `BlueGreenDeploymentName` — ブルー/グリーンデプロイの名前を指定します。
- `Source` — グリーン環境にコピーする DB クラスターの ARN を指定します。
- `TargetEngineVersion` — グリーン環境で DB エンジンのバージョンアップグレードをテストする場合は、エンジンバージョンを指定します。このオプションは、グリーン環境の DB クラスターを指定された DB エンジンバージョンにアップグレードします。

  指定しなかった場合、グリーン環境の DB クラスターは、ブルー環境の DB クラスターと同じエンジンバージョンで作成されます。
- `TargetDBClusterParameterGroupName` – グリーン環境内の DB クラスターに関連付ける DB クラスターパラメータグループを指定します。
- `TargetDBParameterGroupName` – グリーン環境内の DB インスタンスに関連付ける DB パラメータグループを指定します。

ブルー/グリーンデプロイの表示

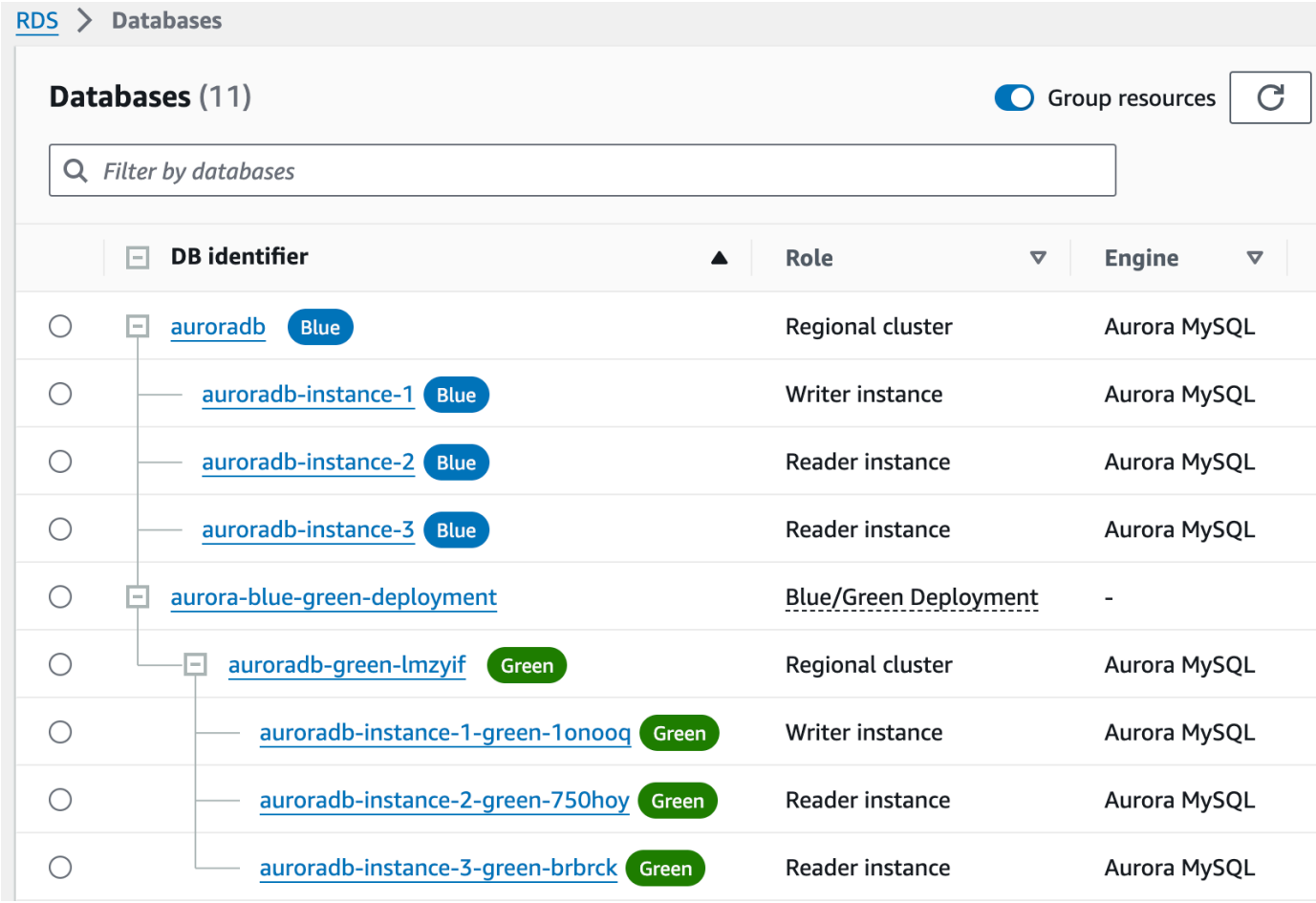
ブルー/グリーンデプロイの詳細は、AWS Management Console、AWS CLI、または RDS API を使用して表示できます。

ブルー/グリーンデプロイについての情報を表示して、イベントをサブスクライブすることもできます。詳細については、「ブルー/グリーンデプロイイベント」を参照してください。

コンソール

ブルー/グリーンデプロイの詳細を表示するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで [Databases] (データベース) を選択し、一覧からブルー/グリーンデプロイを見つけます。

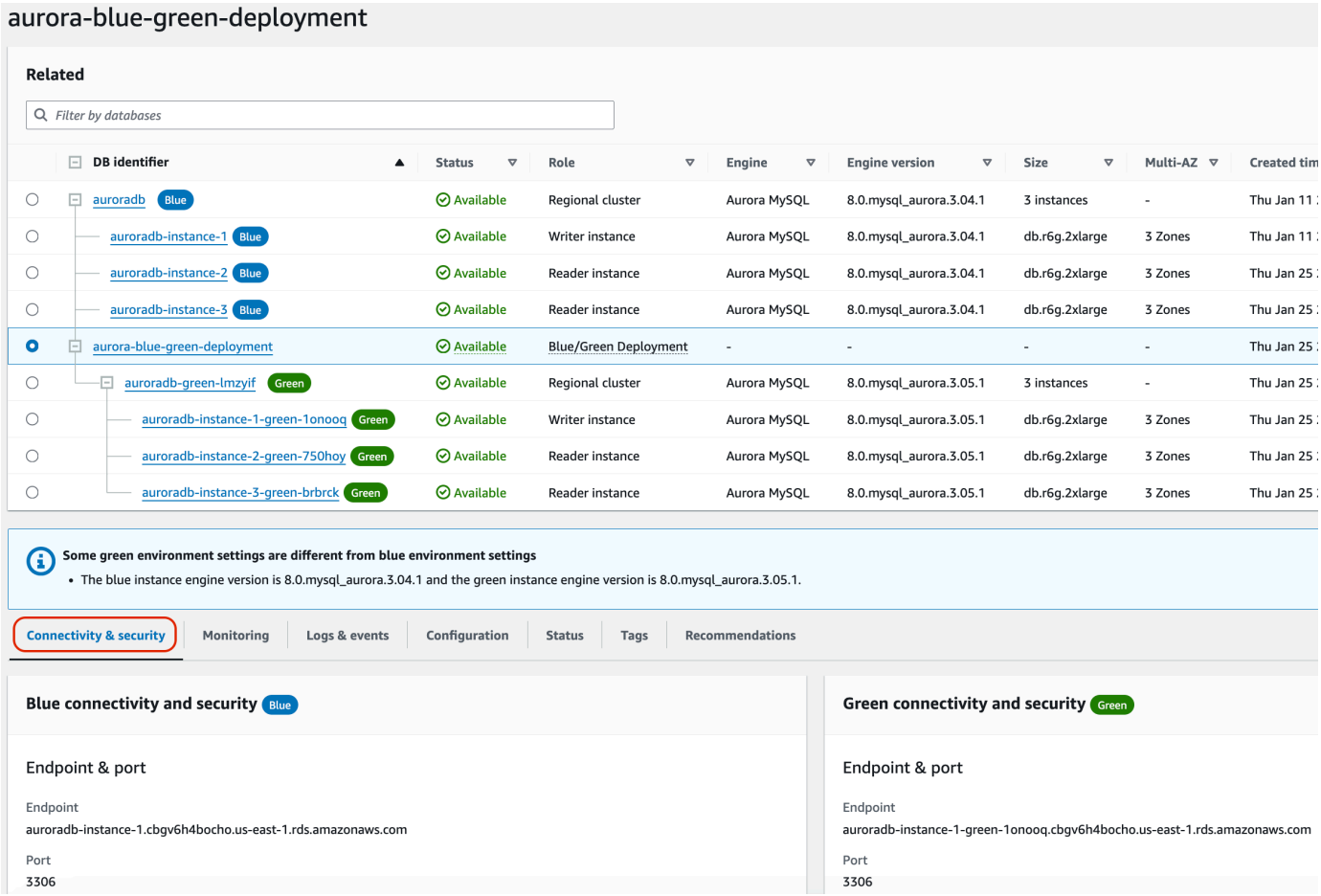


ブルー/グリーンデプロイの [Role] (ロール) 値は、[Blue/Green Deployment] (ブルー/グリーンデプロイ) です。

3. 表示するブルー/グリーンデプロイの名前を選択すると、詳細が表示されます。

各タブには、ブルーデプロイ用のセクションとグリーンデプロイ用のセクションがあります。例えば、[設定] タブでは、グリーン環境で DB エンジンのバージョンをアップグレードしている場合、ブルー環境とグリーン環境で DB エンジンのバージョンが異なる場合があります。

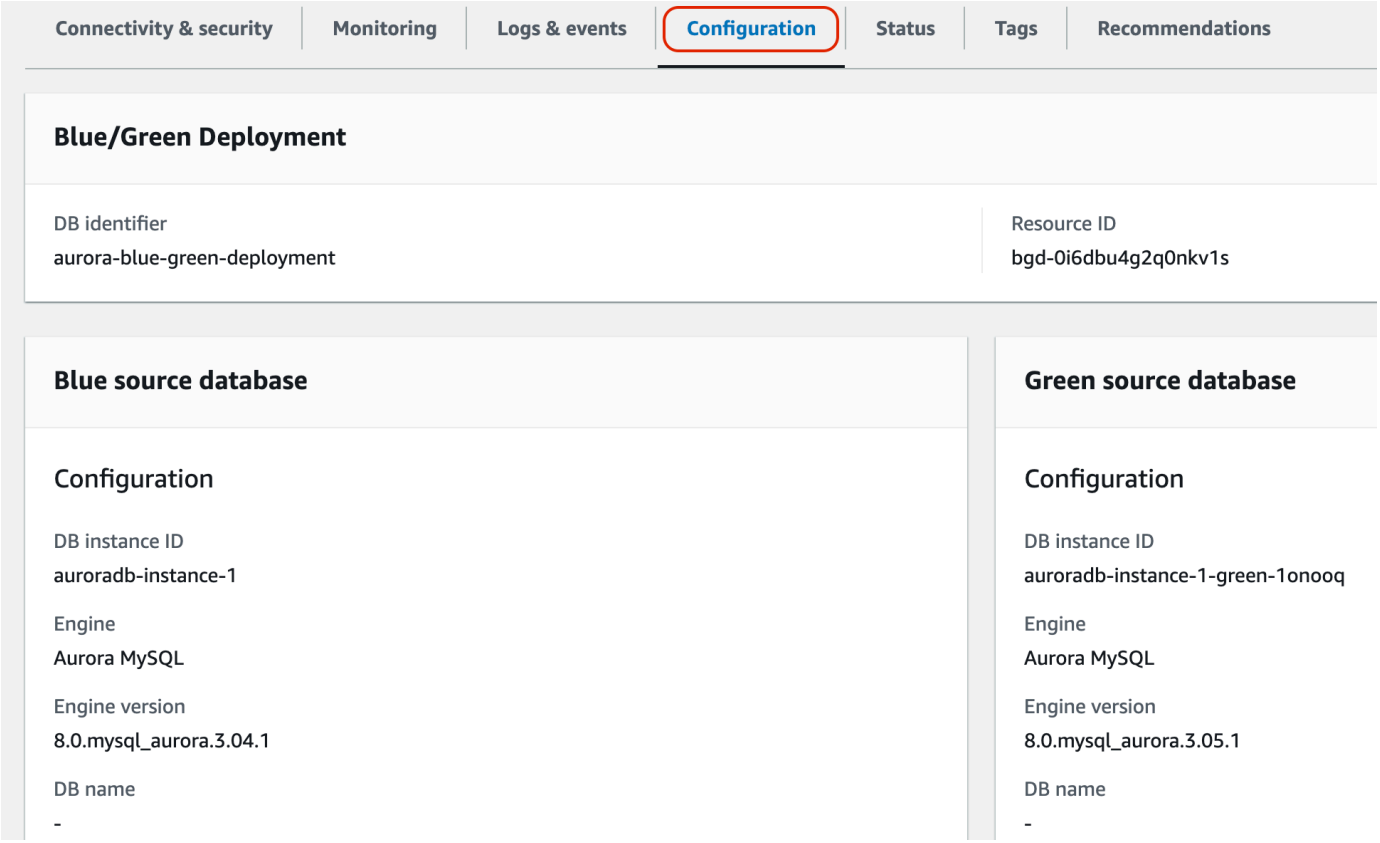
次の画像は、[接続とセキュリティ] タブの例を示しています。



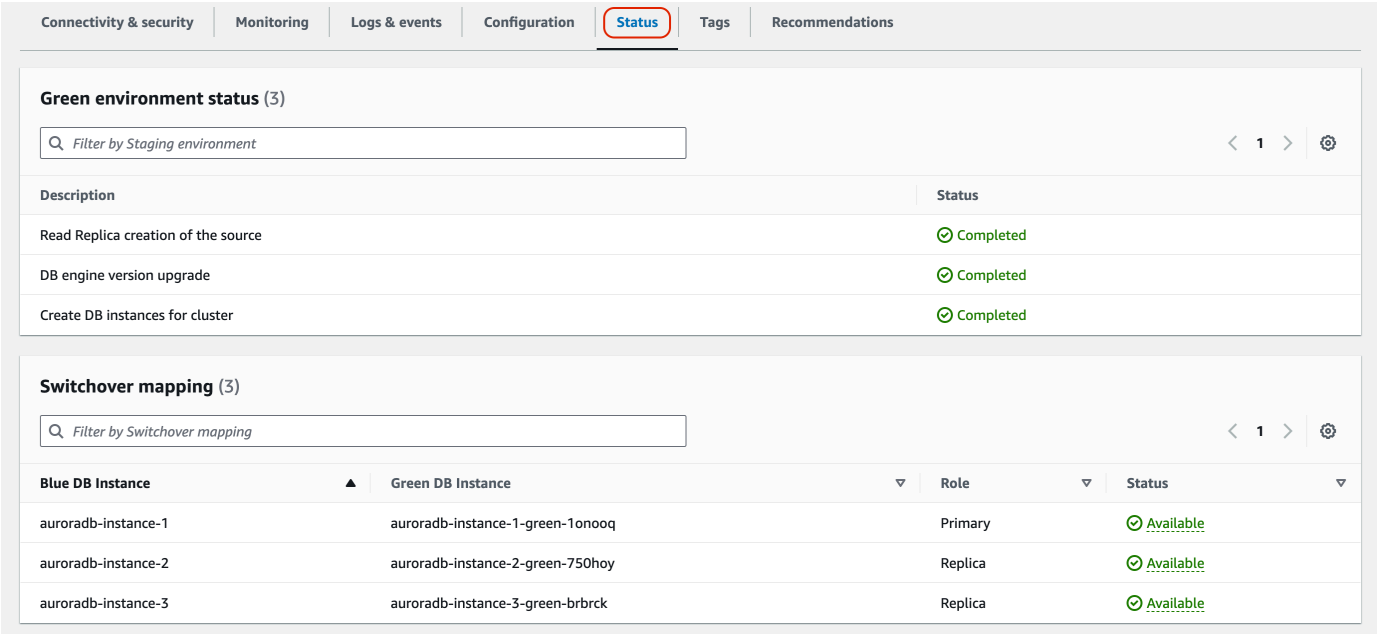
[接続とセキュリティ] タブには、[レプリケーション] というセクションもあります。このセクションには、論理レプリケーションの現在の状態と、ブルー環境とグリーン環境間のレプリカラグが表示されます。レプリケーションの状態が `Replicating` の場合、ブルー/グリーンデプロイは正常にレプリケートされています。

Aurora PostgreSQL ブルー/グリーンデプロイでは、ブルー環境でサポートされていない DDL または大きなオブジェクトを変更すると、レプリケーションの状態が `Replication degraded` に変わることがあります。詳細については、「the section called “PostgreSQL 論理レプリケーションの制約事項”」を参照してください。

次の画像は、[設定] タブの例を示しています。



次の画像は、[ステータス] タブの例を示しています。



AWS CLI

AWS CLI を使用してブルー/グリーンデプロイの詳細を表示するには、describe-blue-green-deployments コマンドを使用します。

Example ブルー/グリーンデプロイの詳細を名前で絞り込んで表示する

describe-blue-green-deployments コマンドを使用すると、--blue-green-deployment-name でフィルタリングできます。次の例は、*my-blue-green-deployment* という名前のブルー/グリーンデプロイの詳細を示しています。

```
aws rds describe-blue-green-deployments --filters Name=blue-green-deployment-
name,Values=my-blue-green-deployment
```

Example 識別子を指定して、ブルー/グリーンデプロイの詳細を表示する

describe-blue-green-deployments コマンドを使用すると、--blue-green-deployment-identifier を指定できます。次の例は、識別子 *bgd-1234567890abcdef* を持つブルー/グリーンデプロイの詳細を示しています。

```
aws rds describe-blue-green-deployments --blue-green-deployment-
identifier bgd-1234567890abcdef
```

RDS API

Amazon RDS API を使用してブルー/グリーンデプロイの詳細を表示するには、DescribeBlueGreenDeployments オペレーションを使用して BlueGreenDeploymentIdentifier を指定します。

ブルー/グリーンデプロイの切り替え

切り替えを行うと、グリーン環境の DB クラスターは DB インスタンスを含めてプロモートされ、本稼働 DB クラスターになります。切り替え前は、本稼働環境のトラフィックはブルー環境のクラスターにルーティングされます。切り替え後、本稼働環境のトラフィックはグリーン環境の DB クラスターにルーティングされます。

トピック

- 切り替えタイムアウト

- 切り替えガードレール
- 切り替えアクション
- 切り替えのベストプラクティス
- 切り替え前に CloudWatch メトリクスを確認する
- スイッチオーバー前のレプリカラグのモニタリング
- ブルー/グリーンデプロイの切り替え
- 切り替え後

切り替えタイムアウト

切り替えのタイムアウト期間は、30 秒から 3,600 秒 (1 時間) まで指定できます。切り替えに指定された期間より長くかかる場合、変更はすべてロールバックされ、どちらの環境にも変更は加えられません。デフォルトのタイムアウト期間は 300 秒 (5 分) です。

切り替えガードレール

切り替えを開始すると、Amazon RDS はいくつかの基本的なチェックを実行して、ブルー環境とグリーン環境が切り替えの準備が整っているかテストします。これらのチェックは切り替えガードレールと呼ばれます。これらの切り替えガードレールは、準備が整っていない環境の切り替えを防ぎます。そのため、予想以上に長いダウンタイムが回避され、切り替えが開始された場合に発生する可能性のあるブルー環境とグリーン環境間のデータ損失を防ぐことができます。

Amazon RDS は、グリーン環境で以下のガードレールチェックを実行します。

- レプリケーションの状態 – グリーン DB クラスターのレプリケーションステータスが正常かどうかをチェックします。グリーン DB クラスターは、ブルー DB クラスターのレプリカです。
- レプリケーションラグ – グリーン DB クラスターのレプリカラグがスイッチオーバーの許容範囲内にあるかどうかをチェックします。許容限度は、指定されたタイムアウト期間に基づきます。レプリカラグは、グリーン DB クラスターがブルー DB クラスターよりどれだけ遅れているかを示します。詳細については、Aurora MySQL および the section called “ レプリケーションのモニタリング” for Aurora PostgreSQL の the section called “リードレプリカ間の遅延の診断と解決” を参照してください
- アクティブな書き込み – グリーン DB クラスターにアクティブな書き込みがないことを確認します。

Amazon RDS は、ブルー環境で以下のガードレールチェックを実行します。

- 外部レプリケーション — Aurora PostgreSQL では、ブルー環境がセルフマネージド論理ソース (パブリッシャー) でもレプリカ (サブスクライバー) でもないことを確認します。その場合は、ブルー環境のすべてのデータベースでセルフマネージドレプリケーションスロットとサブスクリプションを削除し、スイッチオーバーを続行してからそれらを再作成してレプリケーションを再開することをお勧めします。Aurora MySQL の場合は、ブルーデータベースが外部のバイナリログレプリカではないことを確認してください。
- 実行時間の長いアクティブな書き込み — レプリカラグが増える可能性があるため、ブルー DB クラスターに実行時間の長いアクティブな書き込みがないことを確認します。
- 実行時間が長い DDL ステートメント – レプリカラグを増加させる可能性があるため、ブルー DB クラスターに実行時間が長い DDL ステートメントがないことを確認します。
- サポートされていない PostgreSQL の変更 – Aurora PostgreSQL DB クラスター では、ブルー環境で DDL の変更や大きなオブジェクトの追加や変更が行われていないことを確認します。詳細については、「the section called “PostgreSQL 論理レプリケーションの制約事項”」を参照してください。

  Amazon RDS がサポートされていない PostgreSQL の変更を検出すると、レプリケーションの状態が `Replication degraded` に変更され、ブルー/グリーンデプロイではスイッチオーバーができないことが通知されます。スイッチオーバーを続行するには、ブルー/グリーンデプロイとすべてのグリーンデータベースを削除して再作成することをお勧めします。そのためには、[アクション]、[グリーンデータベースで削除] を選択します。

切り替えアクション

ブルー/グリーンデプロイを切り替えると、RDS は次のアクションを実行します。

1. ガードレールチェックを実行して、ブルー環境とグリーン環境を切り替える準備ができているかどうかを確認します。
2. 両方の環境で DB クラスターでの新しい書き込みオペレーションを停止します。
3. 両方の環境で DB インスタンスへの接続を切断し、新しい接続を許可しません。
4. グリーン環境がブルー環境と同期するように、レプリケーションがグリーン環境で追いつくのを待ちます。
5. 両方の環境の DB クラスターと DB インスタンスの名前を変更します。

   RDS は、グリーン環境の DB クラスターと DB インスタンスが、ブルー環境の対応する DB クラスターと DB インスタンスに一致するように名前を変更します。例えば、ブルー環境の DB インスタンスの名前が `mydb` であるとします。また、グリーン環境の対応する DB インスタンスの名

前が mydb-green-abc123 であると仮定します。切り替え時、グリーン環境の DB インスタンスの名前は mydb に変更されます。

RDS は、現在の名前に -old*n* を追加して、ブルー環境の DB クラスターと DB インスタンスの名前を変更します。ここで、*n* は数字です。例えば、ブルー環境の DB インスタンスの名前が mydb であるとします。切り替え後、DB インスタンス名は mydb-old1 になります。

また、RDS はグリーン環境のエンドポイントの名前を、ブルー環境の対応するエンドポイントと一致するように変更するため、アプリケーションを変更する必要はありません。

6. 両方の環境でデータベースへの接続を許可します。
7. 新しい本稼働環境の DB クラスターへの書き込みオペレーションを許可します。

   スイッチオーバーの後、以前の本番 DB クラスターは、読み取りオペレーションのみを許可します。DB クラスターで read_only パラメータを無効にしても、ブルー/グリーンデプロイを削除するまで読み取り専用のままになります。

Amazon EventBridge を使用してスイッチオーバーのステータスをモニタリングできます。詳細については、「the section called “ブルー/グリーンデプロイイベント”」を参照してください。

ブルー環境でタグが設定されている場合、これらのタグは切り替え時に新しい本稼働環境に移動されます。以前の本稼働環境でも、これらのタグは保持されます。タグの詳細については、Amazon RDS リソースのタグ付けを参照してください。

切り替えが開始され、終了する前に何らかの理由で停止した場合、変更はすべてロールバックされ、どちらの環境にも変更は加えられません。

切り替えのベストプラクティス

スイッチオーバーの前に、次のタスクを実行してベストプラクティスに従うことを強くお勧めします。

- グリーン環境でリソースを徹底的にテストします。適切かつ効率的に機能することを確認してください。
- 関連する Amazon CloudWatch メトリクスをモニタリングします。詳細については、「the section called “切り替え前に CloudWatch メトリクスを確認する”」を参照してください。
- 切り替えに最適なタイミングを特定します。

切り替え中は、両方の環境でデータベースからの書き込みが遮断されます。本稼働環境でトラフィックが最も少ない時間を特定します。アクティブな DDL など、トランザクションの実行時間が長い場合、切り替え時間が長くなり、本稼働環境のワークロードのダウンタイムが長くなる可能性があります。

DB クラスターおよび DB インスタンス に多数の接続がある場合は、ブルー/グリーンデプロイを切り替える前に、アプリケーションに必要な最小限の接続数に手動で減らすことを検討してください。これを実現する 1 つの方法は、ブルー/グリーンデプロイのステータスを監視し、ステータスが `SWITCHOVER_IN_PROGRESS` に変わったことを検出すると接続のクリーンアップを開始するスクリプトを作成することです。

- 両方の環境の DB クラスターと DB インスタンスが `Available` 状態にあることを確認します。
- グリーン環境の DB クラスターが正常でレプリケートしていることを確認します。
- ネットワークとクライアントの設定で、DNS キャッシュの存続可能時間 (TTL) が 5 秒を超えないようにしてください。これは Aurora DNS ゾーンのデフォルトです。
  そうしないと、アプリケーションは切り替え後に書き込みトラフィックをブルー環境に送信し続けます。
- Aurora PostgreSQL DB クラスター の場合は、次の操作を行います。
  - スイッチオーバーの前に論理レプリケーションの制約事項を確認し、必要なアクションをすべて実行します。詳細については、「the section called “PostgreSQL 論理レプリケーションの制約事項”」を参照してください。
  - `ANALYZE` 操作を実行して `pg_statistics` テーブルを更新します。これにより、スイッチオーバー後のパフォーマンス上の問題のリスクが軽減されます。

> **Note**
>
> 切り替え中は、切り替えに含まれる DB クラスターを変更することはできません。

切り替え前に CloudWatch メトリクスを確認する

ブルー/グリーンデプロイを切り替える前に、Amazon CloudWatch で次のメトリクスの値を確認することをお勧めします。

- `DatabaseConnections` — このメトリクスを使用して、ブルー/グリーンデプロイのアクティビティレベルを推定し、スイッチオーバー前に、その値がデプロイにとって許容可能なレベルである

ことを確認します。Performance Insights がオンになっている場合、DBLoad は、より正確なメトリクスになります。

- ActiveTransactions — いずれかの DB インスタンスの DB パラメータグループで innodb_monitor_enable が all に設定されている場合、このメトリクスを使用して、切り替えを妨げる可能性のあるアクティブなトランザクションの数が多いかどうかを確認します。

これらのメトリクスの詳細については、「the section called “Aurora の CloudWatch メトリクス”」を参照してください。

スイッチオーバー前のレプリカラグのモニタリング

ブルー/グリーンデプロイを切り替える前に、ダウンタイムを減らすために、グリーンデータベースのレプリカラグがゼロに近いことを確認します。

- Aurora MySQL の場合は、AuroraBinlogReplicaLag CloudWatch メトリクスを使用して、グリーン環境での現在のレプリケーションラグを特定します。
- Aurora PostgreSQL の場合は、次の SQL クエリを使用します。

```
SELECT slot_name,
       confirmed_flush_lsn as flushed,
       pg_current_wal_lsn(),
       (pg_current_wal_lsn() - confirmed_flush_lsn) AS lsn_distance
FROM pg_catalog.pg_replication_slots
WHERE slot_type = 'logical';

slot_name        |    flushed    | pg_current_wal_lsn | lsn_distance
-----------------+---------------+--------------------+------------
logical_replica1 | 47D97/CF32980 | 47D97/CF3BAC8      | 37192
```

confirmed_flush_lsn は、レプリカに送信されたログシーケンス番号 (LSN) の最大値を表します。pg_current_wal_lsn はデータベースの現在の位置を表します。lsn_distance が 0 のとき、レプリカが追いついたことを意味します。

ブルー/グリーンデプロイの切り替え

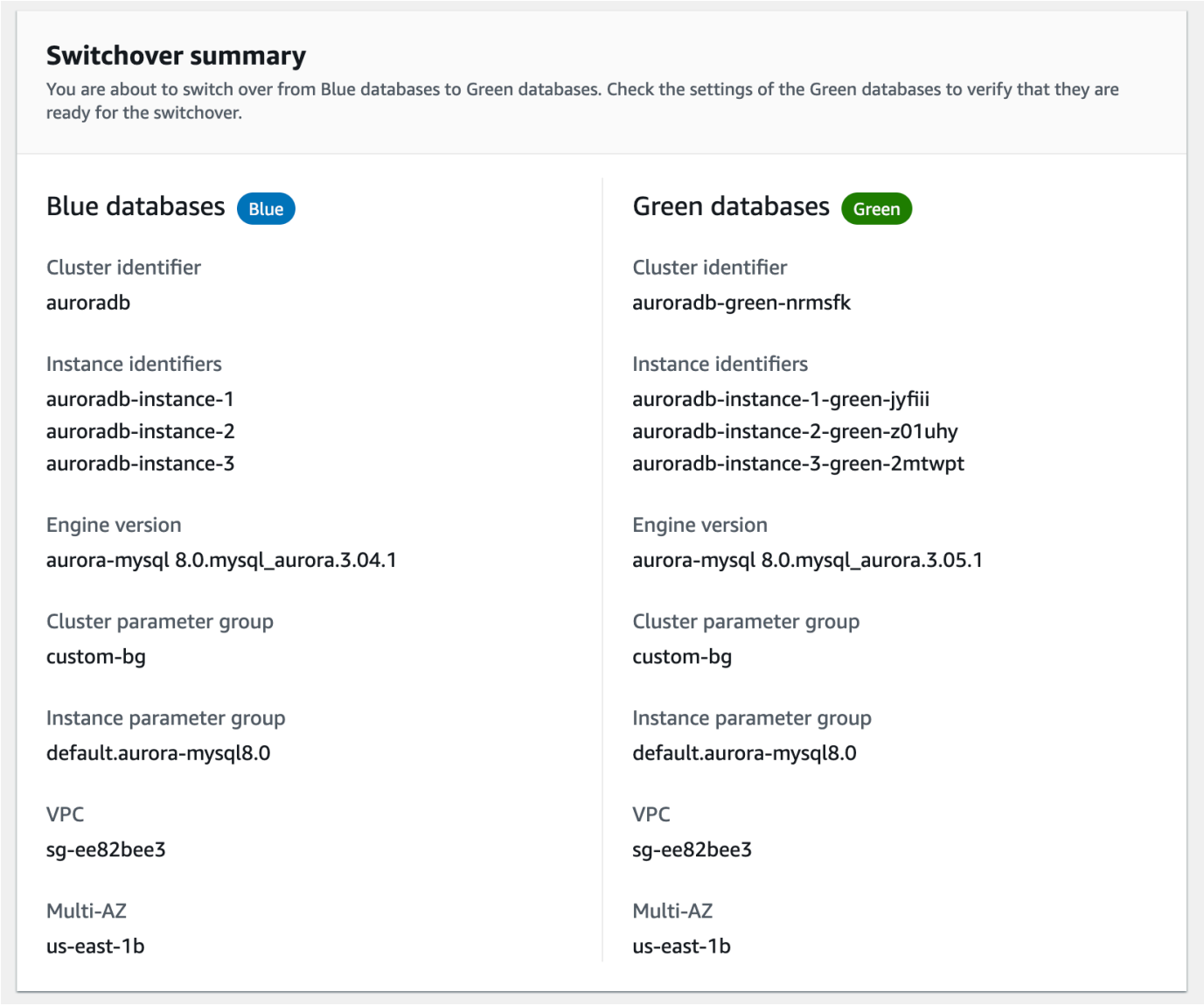
ブルー/グリーンデプロイは、AWS Management Console、AWS CLI、または RDS API を使用して切り替えることができます。

コンソール

ブルー/グリーンデプロイを切り替えるには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[Databases] (データベース) を選択し、切り替えるブルー/グリーンデプロイを選択します。
3. [Actions] (アクション) で、[Switch over] (切り替え) を選択します。

   [Switch over] (切り替え) ページが表示されます。



4. [Switch over] (切り替え) ページで、切り替えの概要を確認します。両方の環境のリソースが期待どおりであることを確認します。一致しない場合は、[Cancel] (キャンセル) を選択します。
5. [タイムアウトの設定] に、スイッチオーバーの制限時間を入力します。
6. クラスターで Aurora PostgreSQL を実行している 場合は、スイッチオーバーの前の推奨事項を確認し、承認してください。詳細については、「the section called “PostgreSQL 論理レプリケーションの制約事項”」を参照してください。
7. [Switch over] (切り替え) を選択します。

AWS CLI

AWS CLI を使用してブルー/グリーンデプロイを切り替えるには、switchover-blue-green-deployment コマンドを次のオプションを指定して使用します。

- `--blue-green-deployment-identifier` — 削除するブルー/グリーンデプロイのリソース ID を指定します。
- `--switchover-timeout` — 切り替えの制限時間を秒単位で指定します。デフォルトは 300 です。

Example ブルー/グリーンデプロイを切り替える

Linux、macOS、Unix の場合:

```
aws rds switchover-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --switchover-timeout 600
```

Windows の場合:

```
aws rds switchover-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --switchover-timeout 600
```

RDS API

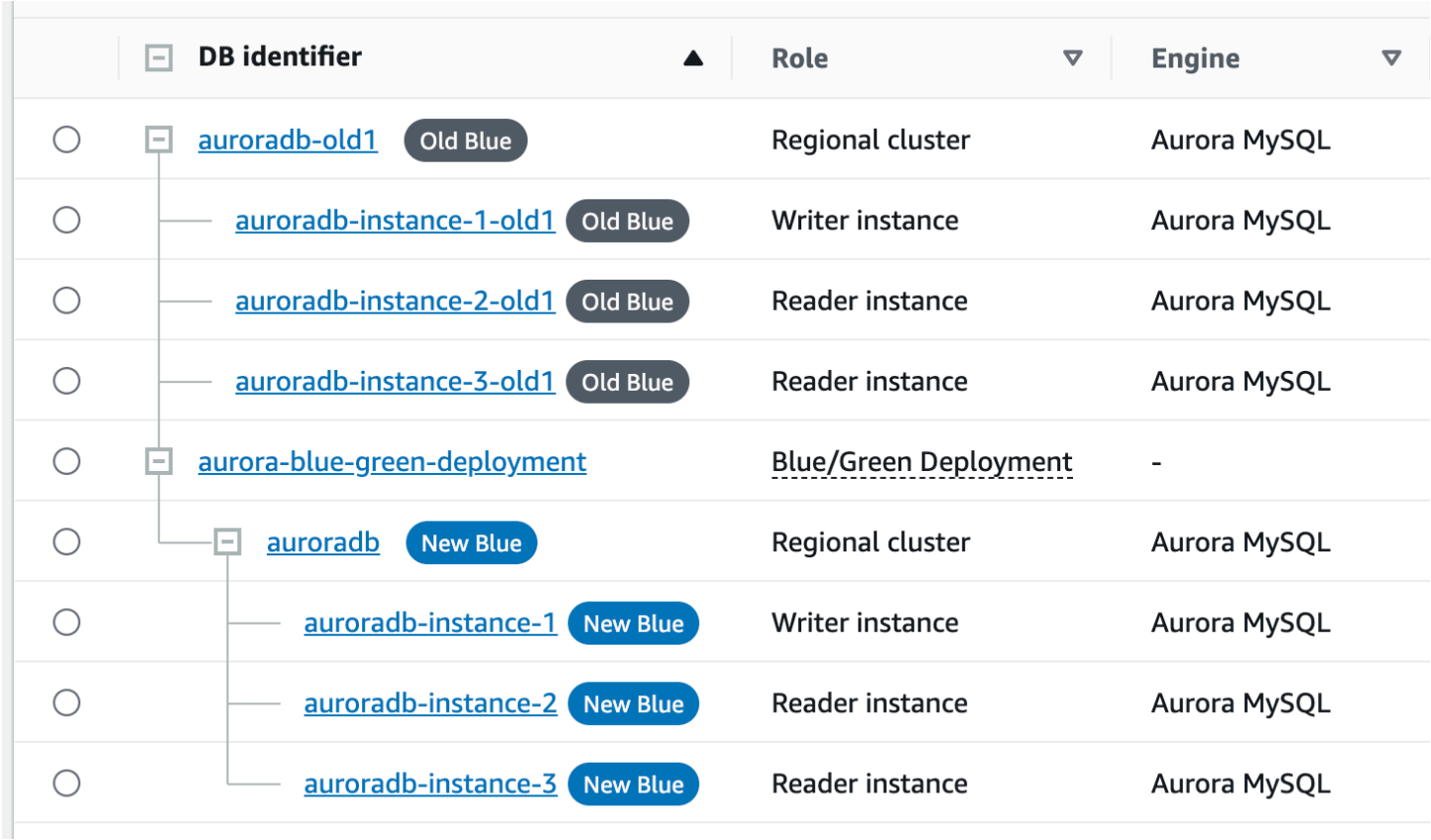
Amazon RDS API を使用してブルー/グリーンデプロイを切り替えるには、SwitchoverBlueGreenDeployment オペレーションを以下のパラメータを指定して使用します。

- `BlueGreenDeploymentIdentifier` — 削除するブルー/グリーンデプロイのリソース ID を指定します。
- `SwitchoverTimeout` — 切り替えの制限時間を秒単位で指定します。デフォルトは 300 です。

切り替え後

切り替え後、以前のブルー環境の DB クラスターと DB インスタンスは保持されます。これらのリソースには標準費用が適用されます。ブルーとグリーンの環境間のレプリケーションとバイナリロギングは停止します。

RDS は、現在のリソース名に `-old`*n* を付加することによって、ブルー環境の DB クラスターと DB インスタンスの名前を変更します。ここで、*n* は数字です。DB クラスターは読み取り専用状態に強制されます。DB クラスターで `read_only` パラメータを無効にしても、ブルー/グリーンデプロイを削除するまで読み取り専用のままになります。



コンシューマーの親ノードの更新

Aurora MySQL ブルー/グリーンデプロイを切り替えた後、スイッチオーバー前にブルー DB クラスターに外部レプリカまたはバイナリログコンシューマーがあった場合は、レプリケーションの継続性を維持するために、スイッチオーバー後に親ノードを更新する必要があります。

スイッチオーバー後、グリーン環境に以前存在していたライター DB インスタンスは、マスターログファイル名とマスターログの位置を含むイベントを発行します。例:

```
aws rds describe-events --output json --source-type db-instance --source-identifier db-
instance-identifier

{
    "Events": [
...
        {
            "SourceIdentifier": "db-instance-identifier",
            "SourceType": "db-instance",
            "Message": "Binary log coordinates in green environment after switchover:
             file mysql-bin-changelog.000003 and position 804",
            "EventCategories": [],
            "Date": "2023-11-10T01:33:41.911Z",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:db-instance-identifier"
        }
    ]
}
```

まず、コンシューマーまたはレプリカが古いブルー環境のすべてのバイナリログを適用していることを確認します。次に、提供されたバイナリログ座標を使用して、コンシューマーでアプリケーションを再開します。例えば、EC2 で MySQL レプリカを実行している場合は、CHANGE MASTER TO コマンドを使用できます。

```
CHANGE MASTER TO MASTER_HOST='{new-writer-endpoint}', MASTER_LOG_FILE='mysql-bin-
changelog.000003', MASTER_LOG_POS=804;
```

ブルー/グリーンデプロイの削除

ブルー/グリーンデプロイは、切り替え前または切り替え後に削除できます。

切り替える前にブルー/グリーンデプロイを削除すると、Amazon RDS はグリーン環境の DB クラスターをオプションで削除します。

- グリーン環境の DB クラスターを削除する場合 (`--delete-target`)、そのクラスターの削除保護が有効になっていないことを確認してください。
- グリーン環境の DB クラスターを削除しなかった場合 (`--no-delete-target`)、そのクラスターは保持されますが、そのクラスターはブルー/グリーンデプロイの一部ではなくなります。レプリケーションは環境間で継続されます。

グリーンデータベースを削除するオプションは、切り替え後はコンソールで使用できなくなります。AWS CLI を使用してブルー/グリーンデプロイを削除するときには、デプロイのステータスが `SWITCHOVER_COMPLETED` の場合、`--delete-target` パラメータを指定できません。

> **Important**
>
> ブルー/グリーンデプロイを削除しても、ブルー環境に影響はありません。

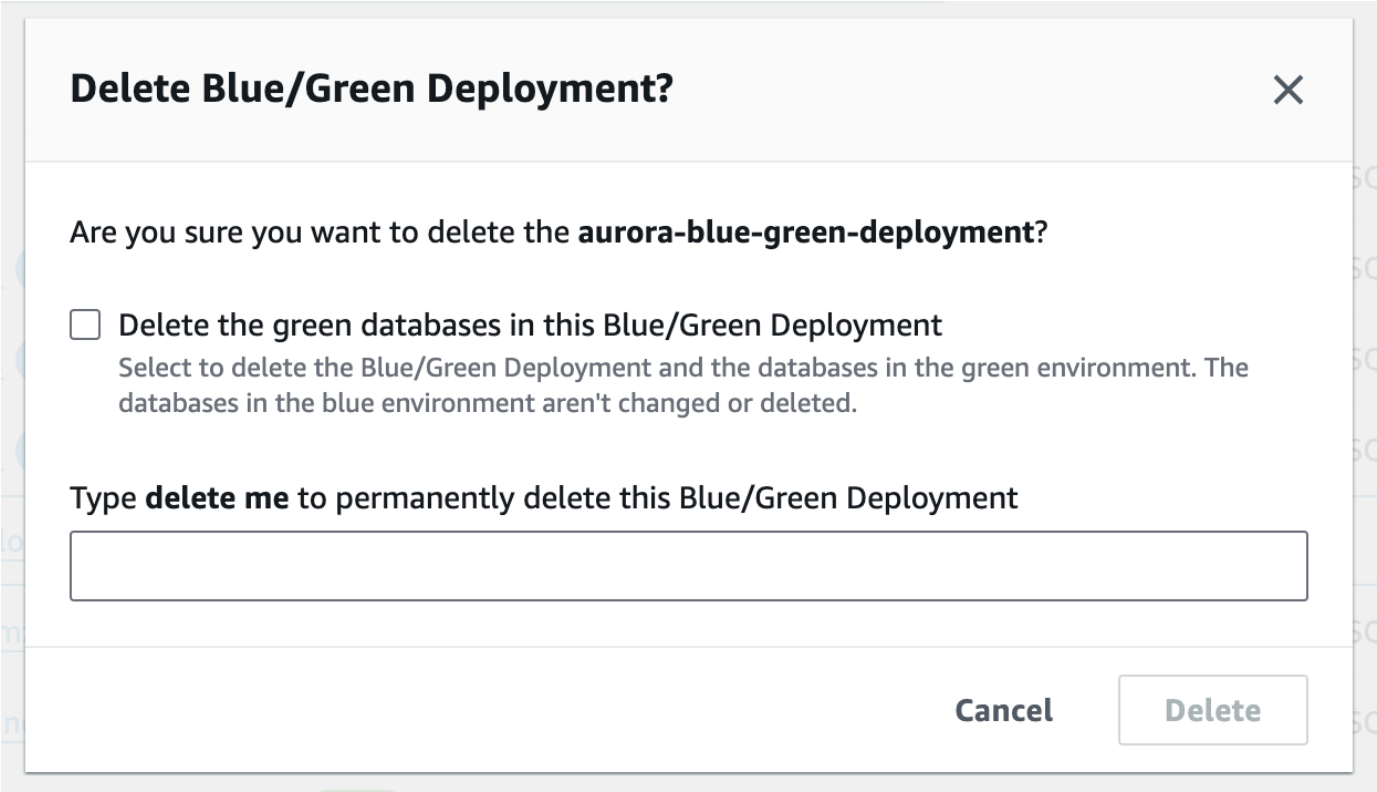
ブルー/グリーンデプロイは、AWS Management Console、AWS CLI、または RDS API を使用して削除できます。

コンソール

ブルー/グリーンデプロイを削除するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Databases] (データベース) を選択し、削除するブルー/グリーンデプロイを選択します。
3. [Actions] (アクション) として、[Delete] (削除) を選択します。

   [Delete Blue/Green Deployment?] (ブルー/グリーンデプロイを削除しますか?) ウィンドウが表示されます。



グリーンデータベースを削除するには、[Delete the green databases in this Blue/Green Deployment] (このブルー/グリーンデプロイのグリーンデータベースを削除) を選択します。

4. ボックスに「**`delete me`**」と入力します。
5. [削除] を選択します。

AWS CLI

AWS CLI を使用してブルー/グリーンデプロイを削除するには、delete-blue-green-deployment コマンドを次のオプションを指定して使用します。

- `--blue-green-deployment-identifier` — 削除するブルー/グリーンデプロイのリソース ID。
- `--delete-target` — グリーン環境の DB クラスターを削除するよう指定します。ブルー/グリーンデプロイのステータスが `SWITCHOVER_COMPLETED` の場合、このオプションは指定できません。
- `--no-delete-target` — グリーン環境の DB クラスターを保持するよう指定します。

Example ブルー/グリーンデプロイとグリーン環境の DB クラスターを削除する

Linux、macOS、Unix の場合:

```
aws rds delete-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --delete-target
```

Windows の場合:

```
aws rds delete-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --delete-target
```

Example ブルー/グリーンデプロイを削除し、グリーン環境の DB クラスターを保持する

Linux、macOS、Unix の場合:

```
aws rds delete-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --no-delete-target
```

Windows の場合:

```
aws rds delete-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --no-delete-target
```

RDS API

Amazon RDS API を使用してブルー/グリーンデプロイを削除するには、以下のパラメータを指定して DeleteBlueGreenDeployment オペレーションを使用します。

- BlueGreenDeploymentIdentifier — 削除するブルー/グリーンデプロイのリソース ID。
- DeleteTarget — TRUE によりグリーン環境の DB を削除するか、FALSE によりクラスターを保持するかを指定します。ブルー/グリーンデプロイのステータスが SWITCHOVER_COMPLETED の場合、TRUE にはできません。

Amazon Aurora DB クラスターのバックアップと復元

これらのトピックでは、Amazon Aurora DB クラスターのバックアップと復元についての情報を提供します。

> **Tip**
>
> Aurora の高可用性機能と自動バックアップ機能により、広範なセットアップを必要とせずにデータを安全に保つことができます。バックアップ戦略を実装する前に、Aurora がデータの複数のコピーを維持する方法と、複数の DB インスタンスおよび AWS リージョン間でそれらのコピーにアクセスする方法について説明します。詳細については、Amazon Aurora の高可用性 を参照してください

トピック

Aurora DB クラスターのバックアップと復元の概要

以下のトピックでは、Aurora バックアップと Aurora DB クラスターを復元する方法について説明します。

目次

バックアップ

Aurora は、クラスターボリュームを自動的にバックアップし、バックアップ保持期間中、復元データを保持します。Aurora の自動バックアップは連続的かつ増分的であるため、バックアップ保持期間内の任意の時点にすばやく復元できます。バックアップデータが書き込まれるときに、データベースサービスのパフォーマンスに影響が出たり、中断が発生したりすることはありません。DB クラスターを作成または変更するときに、バックアップ保持期間 (1 ~ 35 日) を指定できます。Aurora の自動バックアップは Amazon S3 に保存されます。

データ保持期間を超えたバックアップを保持する場合は、クラスターボリュームの中にもデータのスナップショットを作成できます。Aurora DB クラスタースナップショットは期限切れになりません。スナップショットから新しい DB クラスターを作成できます。詳細については、「DB クラスタースナップショットの作成」を参照してください。

> **Note**
>
> - Amazon Aurora DB クラスターの場合、DB クラスターの作成方法に関係なく、デフォルトのバックアップ保持期間は 1 日です。
> - Aurora の自動バックアップを無効にすることはできません。Aurora のバックアップ保持期間は、DB クラスターによって管理されます。

バックアップストレージのコストは、保持する Aurora バックアップおよびスナップショットのデータとその保持期間に応じて異なります。Aurora バックアップおよびスナップショットに伴うストレージの詳細については、「Amazon Aurora バックアップストレージの使用状況を確認する」を参照してください。Aurora バックアップストレージの料金情報については、「Amazon RDS for Aurora の料金」を参照してください。Aurora クラスターを削除した後で、このクラスターに関連するスナップショットを保存すると、Aurora の標準のバックアップストレージ料金が発生します。

AWS Backup を使用する

AWS Backup を使用して、Amazon Aurora DB クラスターのバックアップを管理することができます。

AWS Backup によって管理されるスナップショットは、手動 DB クラスタースナップショットと見なされますが、Aurora の DB クラスタースナップショットクォータにはカウントされません。AWS Backup で作成されたスナップショットには、`awsbackup:job-`*`AWS-Backup-job-number`* という名前が付いています。AWS Backup の詳細については、AWS Backup デベロッパーガイドを参照してください。

また、AWS Backup を使用して、Amazon Aurora DB クラスターの自動バックアップを管理することもできます。DB クラスターが AWS Backup のバックアッププランに関連付けられている場合、そのバックアッププランはポイントインタイムリカバリに使用できます。AWS Backup で管理された自動 (連続) バックアップには、`continuous:cluster-`*`AWS-Backup-job-number`* という名前が付いています。詳細については、「AWS Backup を使用して DB クラスターを指定の時点の状態に復元する」を参照してください。

バックアップウィンドウ

自動バックアップは、優先されるバックアップウィンドウ中に毎日行われます。バックアップウィンドウに割り当てられた時間より長い時間がバックアップに必要な場合、ウィンドウが終了した後

もバックアップが完了するまでバックアップが継続します。バックアップウィンドウは、DB クラスターの週 1 回のメンテナンス時間と重複させることはできません。

Aurora 自動バックアップは継続的かつ漸増的ですが、バックアップウィンドウは、バックアップ保持期間内に保持される毎日のシステムバックアップを作成するために使用されます。保持期間外に保持するには、バックアップをコピーします。

> **Note**
>
> AWS Management Console を使用して DB クラスターを作成する場合、バックアップウィンドウを指定することはできません。ただし、AWS CLI または RDS API を使用して DB クラスターを作成するときは、バックアップウィンドウを指定できます。

DB クラスターの作成時に任意のバックアップウィンドウを指定しない場合、Aurora がデフォルトの 30 分のバックアップウィンドウを割り当てます。この期間は、各 AWS リージョン の 8 時間の時間ブロックからランダムに選択されます。次の表は、デフォルトのバックアップ期間が割り当てられる各 AWS リージョン の時間ブロックを示しています。

リージョン名	リージョン	時間ブロック
米国東部 (オハイオ)	us-east-2	03:00 ~ 11:00 UTC
米国東部 (バージニア北部)	us-east-1	03:00 ~ 11:00 UTC
米国西部 (北カリフォルニア)	us-west-1	06:00 ~ 14:00 UTC
米国西部 (オレゴン)	us-west-2	06:00 ~ 14:00 UTC
アフリカ (ケープタウン)	af-south-1	03:00 ~ 11:00 UTC
アジアパシフィック (香港)	ap-east-1	06:00 ~ 14:00 UTC
アジアパシフィック (ハイデラバード)	ap-south-2	06:30 ~ 14:30 UTC

リージョン名	リージョン	時間ブロック
アジアパシフィック (ジャカルタ)	ap-southeast-3	08:00 ~ 16:00 UTC
アジアパシフィック (メルボルン)	ap-southeast-4	11:00 ~ 19:00 UTC
アジアパシフィック (ムンバイ)	ap-south-1	16:30 ~ 00:30 UTC
アジアパシフィック (大阪)	ap-northeast-3	00:00 ~ 08:00 UTC
アジアパシフィック (ソウル)	ap-northeast-2	13:00 ~ 21:00 UTC
アジアパシフィック (シンガポール)	ap-southeast-1	14:00 ~ 22:00 UTC
アジアパシフィック (シドニー)	ap-southeast-2	12:00 ~ 20:00 UTC
アジアパシフィック (東京)	ap-northeast-1	13:00 ~ 21:00 UTC
カナダ (中部)	ca-central-1	03:00 ~ 11:00 UTC
カナダ西部 (カルガリー)	ca-west-1	18:00 ~ 02:00 UTC
中国 (北京)	cn-north-1	06:00 ~ 14:00 UTC
中国 (寧夏)	cn-northwest-1	06:00 ~ 14:00 UTC
欧州 (フランクフルト)	eu-central-1	20:00 ~ 04:00 UTC
欧州 (アイルランド)	eu-west-1	22:00 ~ 06:00 UTC

リージョン名	リージョン	時間ブロック
欧州 (ロンドン)	eu-west-2	22:00 ~ 06:00 UTC
ヨーロッパ (ミラノ)	eu-south-1	13:00 ~ 21:00 UTC
欧州 (パリ)	eu-west-3	07:29 ~ 14:29 UTC
欧州 (スペイン)	eu-south-2	13:00 ~ 21:00 UTC
欧州 (ストックホルム)	eu-north-1	23:00 ~ 07:00 UTC
欧州 (チューリッヒ)	eu-central-2	13:00 ~ 21:00 UTC
イスラエル (テルアビブ)	il-central-1	03:00 ~ 11:00 UTC
中東 (バーレーン)	me-south-1	06:00 ~ 14:00 UTC
中東 (アラブ首長国連邦)	me-central-1	05:00 ~ 13:00 UTC
南米 (サンパウロ)	sa-east-1	23:00 ~ 07:00 UTC
AWS GovCloud (米国東部)	us-gov-east-1	01:00 ~ 09:00 UTC
AWS GovCloud (米国西部)	us-gov-west-1	06:00 ~ 14:00 UTC

自動バックアップの保持

プロビジョニングされたクラスター、または Aurora Serverless v2 DB クラスターを削除するとき、自動バックアップを保持できます。これにより、クラスターが削除された後でも、バックアップ保持期間内の特定の時点に DB クラスターを復元できます。

保持されている自動バックアップには、DB クラスターからのシステムスナップショットとトランザクションログが含まれています。また、アクティブなクラスターに復元するために必要な DB インスタンスクラスなどの DB クラスタープロパティも含まれます。

AWS Management Console、RDS API、および AWS CLI を使用すると、保持されている自動バックアップを復元または削除できます。

> **Note**
>
> Aurora Serverless v1 DB クラスターの自動バックアップを保持することはできません。

トピック

保持期間

保持されている自動バックアップ内のシステムスナップショットとトランザクションログは、ソース DB クラスターの期限切れと同じ方法で期限切れになります。ソースクラスターの保持期間の設定は、自動バックアップにも適用されます。このクラスター用に作成された新しいスナップショットやログはないため、保持されている自動バックアップは最終的には完全に期限切れになります。保持期間が終了した後も、手動の DB クラスタースナップショットは引き続き保持されますが、自動バックアップはすべて期限切れになります。

コンソール、AWS CLI、または RDS API を使用して、保持されている自動バックアップを削除できます。詳細については、「保持している自動バックアップの削除」を参照してください。

保持されている自動バックアップとは異なり、最終スナップショットには有効期限がありません。自動バックアップを保持しているとしても、最終スナップショットを作成しておくことを強くお勧めします。保持されている自動バックアップは最終的に期限切れになるためです。

保持されたバックアップの表示

保持されている自動バックアップを RDS コンソールに表示するには、ナビゲーションペインで [自動バックアップ] を選択し、[保持] を選択します。保持された自動バックアップに関連付けられている個々のスナップショットを表示するには、ナビゲーションペインで [Snapshots] (スナップショット) を選択します。または、保持されている自動バックアップに関連付けられた個別のスナップ

ショットを記述できます。そのページで、スナップショットのいずれかから DB インスタンスを直接復元できます。

AWS CLI を使用して保持されている自動バックアップを説明するには、次のいずれかのコマンドを使用します。

```
aws rds describe-db-cluster-automated-backups --db-cluster-resource-
id DB_cluster_resource_ID
```

RDS API を使用して保持されている自動バックアップを説明するには、以下のパラメータのいずれかを指定してDescribeDBClusterAutomatedBackupsアクションDbClusterResourceIdを呼び出します。

保持コスト

各 Aurora DB クラスターの Aurora データベースの総ストレージの 100% までのバックアップストレージには追加料金はかかりません。また、DB クラスターを削除した後も自動バックアップを保持する場合、最大 1 日分の追加料金は発生しません。2 日以上保持しているバックアップには料金がかかります。

トランザクションログまたはインスタンスメタデータには追加料金はかかりません。バックアップのその他の料金ルールはすべて、復元可能なクラスターに適用されます。詳細については、「Amazon Aurora 料金」ページを参照してください。

制約事項

保持されている自動バックアップには、次の制限が適用されます。

- 1 つの AWS リージョンで保持できる自動バックアップの最大数は 40 個です。DB クラスターのクォータには含まれません。同時に最大 40 個の DB クラスターを実行し、40 個の DB インスタンスを実行し、40 個の DB クラスターの自動バックアップを保持できます。

  詳細については、「Amazon Aurora のクォータ」を参照してください。
- 保持されている自動バックアップには、パラメータグループまたはオプショングループについての情報は含まれません。
- 削除したクラスターを、削除時の保持期間内のポイントインタイムに復元できます。
- 保持されている自動バックアップは、ソースインスタンスを削除した時点で存在していたシステムバックアップ、トランザクションログ、および DB クラスターのプロパティで構成されているため、変更できません。

保持している自動バックアップの削除

保持された自動バックアップは、不要になったら削除できます。

コンソール

保持されている自動バックアップを削除するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Automated backups (自動バックアップ)] を選択します。
3. [保持] タブを選択します。

4. 削除する保持された自動バックアップを選択します。
5. 「アクション」 で、「削除」 を選択します。
6. 確認ページで、「**`delete me`**」を入力し、[Delete (削除)] を選択します。

AWS CLI

AWS CLI コマンド delete-db-cluster-automated-backup で次のオプションを指定することにより、保持されている自動バックアップを削除できます。

- `--db-cluster-resource-id` – ソース DB クラスターのリソース識別子。

  AWS CLI コマンド describe-db-cluster-automated-backups を実行するこちにより、保持された自動バックアップのソース DB クラスターのリソース識別子を見つけることができます。

Example

この例では、リソース ID `cluster-123ABCEXAMPLE` を持つソース DB クラスターの保持されている自動バックアップを削除します。

Linux、macOS、Unix の場合:

```
aws rds delete-db-cluster-automated-backup \
    --db-cluster-resource-id cluster-123ABCEXAMPLE
```

Windows の場合:

```
aws rds delete-db-cluster-automated-backup ^
    --db-cluster-resource-id cluster-123ABCEXAMPLE
```

RDS API

次のパラメータを指定して Amazon RDS API オペレーション DeleteDBClusterAutomatedBackup を使用することにより、保持されている自動バックアップを削除できます。

- `DbClusterResourceId` – ソース DB クラスターのリソース識別子。

  Amazon RDS API オペレーション DescribeDBClusterAutomatedBackups を使用して、保持されている自動バックアップのソース DB インスタンスのリソース識別子を見つけることができます。

データの復元

Aurora が保持するバックアップデータから、保存した DB クラスターのスナップショットから、または保持されている自動バックアップから、新しい Aurora DB クラスターを作成することで、データを回復できます。バックアップデータから作成された DB クラスターの新しいコピーは、バックアップ保持期間内の任意の時点にすばやく復元できます。バックアップ保持期間中の Aurora 自動バックアップは連続的かつ増分的であるため、復元時間を短縮するためにデータのスナップショットを頻繁に作成する必要がありません。

DB クラスターの最新の復元可能時刻は、DB クラスターを復元できる最近の時点です。これは通常、アクティブな DB クラスターの場合は現在時刻から 5 分以内であり、または保持されている自動バックアップの場合はクラスター削除時刻の 5 分以内です。

最も早い復元可能時刻は、バックアップ保持期間内のどこまで遡ってクラスターボリュームを復元できるかを示します。

DB クラスターの最新の復元可能時刻または最も早い復元可能時刻を判断するには、RDS コンソールで`Latest restorable time` 値または `Earliest restorable time` 値を探します。これらの値の表示については、「保持されたバックアップの表示」を参照してください。

DB クラスターの復元が完了したことは、`Latest restorable time` および `Earliest restorable time` の値を確認することでわかります。復元操作が完了するまで、これらの値は

NULL を返します。`Latest restorable time` または `Earliest restorable time` が NULL を返す場合、バックアップまたは復元操作をリクエストすることはできません。

DB クラスターを指定の時点の状態に復元する方法については、「DB クラスターを指定の時点の状態に復元する」を参照してください。

Aurora 用のデータベースのクローン作成

DB クラスターのスナップショットを復元する代わりに、データベースのクローン作成により Aurora DB クラスターのデータベースのクローンを新しい DB クラスターに作成することもできます。クローンデータベースの初回作成時に使用する追加スペースは最小限です。ソースデータベースまたはクローンデータベースのいずれかでデータが変更された場合に限り、データがコピーされます。同じ DB クラスターから複数のクローンを作成したり、他のクローンから追加のクローンを作成することもできます。詳細については、「Amazon Aurora DB クラスターのボリュームのクローン作成」を参照してください。

バックトラック

Aurora MySQL が、バックアップからデータを復元しないで、DB クラスターを特定の時刻に「巻き戻し」することができるようになりました。詳細については、「Aurora DB クラスターのバックトラック」を参照してください。

Amazon Aurora バックアップストレージの使用状況を確認する

Amazon Aurora では、自動 (連続) バックアップとスナップショットの 2 種類のバックアップを管理します。

自動バックアップストレージ

クラスターの自動 (連続) バックアップでは、指定された保持期間内のデータベースのすべての変更を増分的に保存し、その保持期間内の任意の時点に復元できます。保持期間は 1～35 日の範囲です。自動バックアップは増分的であり、保持期間内の任意の時点に復元する必要があるストレージの量に基づいて課金されます。

Aurora では無料のバックアップ使用量も用意されています。この無料使用量は、(`VolumeBytesUsed` Amazon CloudWatch メトリクスで表される) 最新のクラスターボリュームサイズと同じです。この量は、計算された自動バックアップ使用量から差し引かれます。また、保持期間が 1 日だけの自動バックアップについても料金はかかりません。

例えば、自動バックアップの保持期間が 7 日であり、クラスターを 4 日前の状態に復元する必要があるとします。Aurora は、自動バックアップに保存された増分データを使用して、4 日前の同時刻の時点でのクラスターの状態を再作成します。

自動バックアップでは、クラスターを保持期間内の任意の時点の復元するために必要な情報がすべて保存されます。つまり、新しい情報の書き込みや既存の情報の削除を含め、保持期間中のすべての変更が保存されます。多くの変更が発生するデータベースでは、自動バックアップのサイズは時間の経過とともに大きくなります。データベースに変更が加えられなくなった後は、以前に保存された変更が保持期間を終了するため、自動バックアップのサイズは減少することが予想されます。

自動バックアップの合計請求使用量が、保持期間中の累積クラスターボリュームサイズを超えることはありません。例えば、保持期間が 7 日間で、クラスターボリュームが毎日 100 GB の場合、請求される自動バックアップの使用量は 700 GB (100 GB * 7) を超えることはありません。

スナップショットストレージ

DB クラススナップショットは常にフルバックアップであり、スナップショット作成時点でのクラスターボリュームのサイズと同じサイズです。スナップショットは、ユーザーが手動で作成したものでも、AWS バックアッププランによって自動的に作成されたものでも、手動スナップショットとして扱われます。Aurora は、自動バックアップ保持期間内のすべてのスナップショットについて無制限の無料ストレージを提供します。手動スナップショットが保持期間外になると、毎月 GB 単位で

請求されます。自動化されたシステムスナップショットは、コピーされ、保持期間を過ぎて保持されない限り、課金されることはありません。

Aurora バックアップの一般情報については、「バックアップ」を参照してください。Aurora バックアップストレージの料金情報については、「Amazon Aurora の料金」のページを参照してください。

Aurora バックアップストレージの Amazon CloudWatch メトリクス

Aurora クラスターをモニタリングし、レポートを作成するには、CloudWatch コンソールから Amazon CloudWatch メトリクスを使用できます。CloudWatch メトリクスを使用して、次に示すように、Aurora バックアップに使用されているストレージの量を確認およびモニタリングできます。これらのメトリクスは、Aurora DB クラスターごとに個別に計算されます。

- `BackupRetentionPeriodStorageUsed` は、自動バックアップを保存するために現時点までに使用されたバックアップストレージの量 (バイト単位) を示します。
  - この値は、クラスターボリュームのサイズと、保持期間中に DB クラスターに加えられた変更 (書き込みと更新) の数によって異なります。これは、任意の時点に復元できるようにするには、自動バックアップにクラスターに加えられたすべての変更を保存する必要があるためです。
  - このメトリクスからは、Aurora が提供する無料利用枠のバックアップ使用量は差し引かれません。
  - このメトリクスでは、その日に記録された自動バックアップ使用量について、毎日 1 つのデータポイントが出力されます。
- `SnapshotStorageUsed` - 自動バックアップの保持期間を超えて手動スナップショットを保存するために使用されたバックアップストレージの量 (バイト単位) を表します。
  - 値は、自動バックアップの保持期間を超えて保持するスナップショットの数と、各スナップショットのサイズによって異なります。
  - 各スナップショットのサイズは、そのスナップショットを作成した時点のクラスターボリュームのサイズです。
  - スナップショットはフルバックアップであり、増分バックアップではありません。
  - このメトリクスでは、課金対象のスナップショットごとに 1 日に 1 つのデータポイントが出力されます。1 日のスナップショットの合計使用量を確認するには、このメトリクスの 1 日の合計を求めます。
- `TotalBackupStorageBilled` — 特定のクラスターのすべての請求対象のバックアップ使用量のメトリクスをバイト単位で表します。

`BackupRetentionPeriodStorageUsed + SnapshotStorageUsed - free tier`

- このメトリクスでは、Aurora が提供する無料利用枠のバックアップ使用量を差し引いた `BackupRetentionPeriodStorageUsed` 値について、1 日に 1 つのデータポイントが出力されます。この無料利用枠は、DB クラスターボリュームの最新の記録サイズと同じです。このデータポイントは、自動バックアップの実際の請求使用量を表します。
- このメトリクスでは、すべての `SnapshotStorageUsed` 値について個別の日次データポイントが出力されます。
- 1 日のバックアップの合計使用量を確認するには、このメトリクスの 1 日の合計を求めます。これは、スナップショットの請求使用量と自動バックアップの請求使用量を合計して、バックアップ請求使用量の合計になります。

CloudWatch のメトリクスを使用する方法については、Amazon RDS コンソールでの Aurora メトリクスの使用可否 を参照してください。

バックアップストレージ使用量の計算

自動バックアップの使用量は、バックアップの保持期間内の任意の時点に復元できるようにするために、保存する必要のあるすべての増分レコードを調べて計算されます。

例えば、保持期間が 7 日間の自動バックアップがあるとします。保持期間の直前のクラスターボリュームサイズは 100 GB であり、これは、Aurora が保存する必要のある最小容量です。次に、次の 7 日間に次のアクティビティを行います。ここで、増分レコードサイズとは、データベースの書き込みと更新による変更レコードを格納するのに必要なストレージの量です。

日	増分レコードサイズ (GB)
1	10
2	15
3	25
4	20
5	10
6	25

日	増分レコードサイズ (GB)
7	30
合計	135

このデータから、バックアップの自動バックアップ使用量の計算結果は次のようになります。

```
100 GB (volume size before retention period) + 135 GB (size of incremental records) =
 235 GB total backup usage
```

その後、請求使用量から無料利用枠が差し引かれます。ボリュームの最新サイズが 200 GB であると仮定します。

```
235 GB total backup usage - 200 GB (latest volume size) = 35 GB billed backup usage
```

よくある質問

スナップショットの請求はいつ行われますか?

自動バックアップの保持期間外の (より古い) 手動スナップショットに対して請求されます。

手動スナップショットとは何ですか?

手動スナップショットは、次の条件のいずれかが当てはまるスナップショットです。

- ユーザーによって手動でリクエストされた
- AWS Backup などの自動バックアップサービスによって撮られた
- 保持期間外に保存するために自動システムスナップショットからコピーされた

DB クラスターを削除した場合、手動スナップショットはどうなりますか?

手動スナップショットは、お客様が削除するまで期限切れになりません。

DB クラスターを削除しても、以前に作成した手動スナップショットは引き続き存在します。これらのスナップショットが自動バックアップ保持期間内だったために請求されていなかった場合、現在は対象外となり、使用量に対してすべてフルサイズで請求されるようになります。

バックアップストレージのコストを削減するにはどうしたらいいですか ?

バックアップ使用量に関連するコストを削減するには、いくつかの方法があります。

- 自動バックアップの保持期間外の手動スナップショットを削除します。これには、お客様が撮影したスナップショットだけでなく、AWS Backup プランで撮影した可能性のあるスナップショットも含まれます。AWS Backup プランをチェックして、予定外の保持期間外にスナップショットが保持されていないことを確認してください。
- データベースへの書き込みと更新を評価して、変更する回数を減らすことができるかどうか確認します。自動バックアップでは、保持期間内のすべての増分変更が保存されるため、更新する回数を減らすと、自動バックアップの料金も下がります。
- 自動バックアップの保持期間を短縮することが理にかなっているかどうかを評価します。保持期間を短縮すると、バックアップに増分データが保存される日数が少なくなるため、バックアップ全体のコストを削減できます。ただし、この保持期間を短縮すると、一部のスナップショットが保持期間外になるために請求が開始される可能性もあります。これが適切かどうかを判断する前に、発生する可能性のある余分なスナップショットコストをすべて確認してください。

バックアップストレージはどのように請求されますか?

バックアップストレージは GB /月単位で請求されます。

つまり、バックアップストレージの使用量は、その月の使用量の加重平均として請求されます。30 日間の月の例をいくつか示します。

- 請求されるバックアップ使用量は、月の 30 日間すべてで 100 GB です。料金は以下の通りです。

```
(100 GB * 30) / 30 = 100 GB-month
```

- 請求されるバックアップ使用量は、月の前半の 15 日間は 100 GB、後半の 15 日間は 0 GB です。料金は以下の通りです。

```
(100 GB * 15 + 0 GB * 15) / 30 = 50 GB-month
```

- 請求されるバックアップ使用量は、月の最初の 10 日間は 50 GB、次の 10 日間は 100 GB、最後の 10 日間は 150 GB です。料金は以下の通りです。

```
(50 GB * 10 + 100 GB * 10 + 150 GB * 10) / 30 = 100 GB-month
```

DB クラスターのバックトラック設定はバックアップストレージの使用量にどのように影響しますか?

Aurora DB クラスターのバックトラック設定は、クラスターのバックアップデータのボリュームに影響しません。Amazon では、バックトラックデータのストレージに対して別々に課金しま

す。Aurora バックトラックの料金情報については、「Amazon Aurora の料金」ページを参照してください。

共有スナップショットにはどのようにストレージコストがかかりますか?

別のユーザーとスナップショットを共有している場合でも、そのスナップショットの所有者のままです。ストレージコストはそのスナップショットの所有者に適用されます。所有している共有スナップショットを削除した場合は、誰もアクセスできなくなります。

別のユーザーが所有している共有スナップショットに引き続きアクセスするには、そのスナップショットをコピーします。これにより、新しいスナップショットの所有者になります。コピーされたスナップショットのストレージコストはすべて、お客様のアカウントに適用されます。

スナップショットの共有の詳細については、「DB クラスターのスナップショットの共有」を参照してください。スナップショットのコピーの詳細については、「DB クラスターのスナップショットのコピー」を参照してください。

DB クラススナップショットの作成

Amazon RDS は DB クラスターのストレージボリュームのスナップショットを作成し、個々のデータベースだけではなく、その DB クラスター全体をバックアップします。DB クラスターを作成したら、バックアップする DB クラスターを特定してから、DB クラスターに名前を付けて後で復元できるようにする必要があります。DB クラススナップショットを作成するためにかかる時間は、データベースのサイズによって異なります。スナップショットにはストレージボリューム全体が含まれているため、一時ファイルなどのファイルのサイズも、スナップショットを作成するための時間に影響します。

> **Note**
>
> DB クラスターのスナップショットを撮るには、DB クラスターが `available` 状態である必要があります。

自動バックアップとは異なり、手動スナップショットはバックアップ保持期間の影響を受けません。スナップショットは期限切れになりません。

非常に長期間のバックアップの場合、スナップショットデータを Amazon S3 にエクスポートすることをお勧めします。DB エンジンのメジャーバージョンがサポートされなくなった場合、スナップショットからそのバージョンに復元することはできません。詳細については、「Amazon S3 への DB クラススナップショットデータのエクスポート」を参照してください。

DB クラススナップショットは、AWS Management Console、AWS CLI、または RDS API を使用して作成できます。

コンソール

DB クラススナップショットを作成するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[Snapshots] を選択します。

   手動スナップショットリストが表示されます。
3. [スナップショットの取得] を選択します。

   [Take DB snapshot] (DB スナップショットの取得) ウィンドウが表示されます。

4. [スナップショットのタイプ] で、[DB クラスター] を選択します。
5. スナップショットを作成する [DB クラスター] を選択します。
6. [スナップショット名] を入力します。
7. [スナップショットの取得] を選択します。

   [手動スナップショット] のリストが表示され、新しい DB スナップショットのステータスが `Creating` として表示されます。ステータスが `Available` になると、その作成時間が表示されます。

AWS CLI

AWS CLI を使用して DB クラスタースナップショットを作成するときは、バックアップする DB クラスターを特定してから、DB クラスタースナップショットに名前を付けて後で復元できるようにする必要があります。そのためには、以下のパラメータを指定して AWS CLI の `create-db-cluster-snapshot` コマンドを使用します。

- `--db-cluster-identifier`
- `--db-cluster-snapshot-identifier`

この例では、*`mydbcluster`* という DB クラスターについて *`mydbclustersnapshot`* という名前の DB クラスタースナップショットを作成します。

Example

Linux、macOS、Unix の場合:

```
aws rds create-db-cluster-snapshot \
    --db-cluster-identifier mydbcluster \
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

Windows の場合:

```
aws rds create-db-cluster-snapshot ^
    --db-cluster-identifier mydbcluster ^
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

RDS API

Amazon RDS API を使用して DB クラスタースナップショットを作成するときは、バックアップする DB クラスターを特定してから、DB クラスタースナップショットに名前を付けて後で復元できるようにする必要があります。そのためには、以下のパラメータを指定して Amazon RDS API の `CreateDBClusterSnapshot` コマンドを使用します。

- DBClusterIdentifier
- DBClusterSnapshotIdentifier

DB クラスタースナップショットが使用可能かどうかの確認

DB クラスターのスナップショットが利用可能であることを確認するには、CLI コマンドの `describe-db-cluster-snapshots`、または API アクションの `DescribeDBClusterSnapshots` を使用して、AWS Management Console のクラスターの詳細ページにある [Maintenance & backups] (メンテナンスとバックアップ) タブの [Snapshots] (スナップショット) を表示します。

また、`wait db-cluster-snapshot-available` CLI コマンドを使用して、スナップショットが使用可能になるまで 30 秒ごとに API をポーリングすることもできます。

DB クラスターのスナップショットからの復元

Amazon RDS は DB クラスターのストレージボリュームのスナップショットを作成し、個々のデータベースだけではなく、その DB クラスター全体をバックアップします。DB スナップショットからの復元で、新しい DB クラスターを作成できます。復元の元となる DB クラスタースナップショットの名前を指定し、復元によって作成される新しい DB クラスターの名前を指定します。DB クラスタースナップショットから既存の DB クラスターに復元することはできません。復元すると新しい DB クラスターが作成されます。

> **Important**
>
> スナップショットを廃止済みの DB エンジンバージョンに復元しようとすると、最新のエンジンバージョンへのアップグレードがすぐに始まります。また、バージョンが延長サポートの対象であるか、標準サポートが終了した場合、延長サポート料金が適用される場合があります。詳細については、「Amazon RDS 延長サポートの使用」を参照してください。

ステータスが `available` になると、復元された DB クラスターを使用することができます。

AWS CloudFormation を使用して、DB クラスターのスナップショットから DB クラスターを復元できます。詳細については、AWS CloudFormation ユーザーガイドの AWS::RDS::DBCluster を参照してください。

> **Note**
>
> 手動 DB クラスタースナップショットを共有すると、暗号化されているかいないかに関係なく、権限を持つ AWS アカウントが DB クラスターをコピーしてそこから復元するのではなく、スナップショットから DB クラスターを直接復元できるようになります。詳細については、「DB クラスターのスナップショットの共有」を参照してください。

RDS 延長サポートバージョンを使用した Aurora DB クラスターまたはグローバルクラスターの復元については、「Amazon RDS 延長サポートでの Aurora DB クラスターまたはグローバルクラスターの復元」を参照してください。

パラメータグループに関する考慮事項

復元された DB クラスターを適切なパラメータグループと関連付けることができるように、作成する DB クラススナップショットの DB パラメータグループと DB クラスターは保持しておくことをお勧めします。

異なるパラメータグループを選択しない限り、デフォルトの DB パラメータグループと DB クラスターのパラメータグループが、復元されたクラスターに関連付けられます。デフォルトのパラメータグループでは、カスタムのパラメータ設定を使用できません。

パラメータグループは、DB クラスターを復元する際に指定できます。

DB パラメータグループと DB クラスターのパラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。

セキュリティグループに関する考慮事項

DB クラスターを復元すると、仮想プライベートクラウド (VPC)、DB サブネットグループ、および VPC セキュリティグループはデフォルトのものが、(それらに別のものを選択しない限り) 復元されたインスタンスに関連付けられます。

- Amazon RDS コンソールを使用している場合は、カスタムの VPC セキュリティグループを指定してクラスターに関連付けるか、新しい VPC セキュリティグループを作成できます。
- AWS CLI を使用している場合、`restore-db-cluster-from-snapshot` コマンドで `--vpc-security-group-ids` オプションを指定することにより、カスタムの VPC セキュリティグループを指定して、それをクラスターに関連付けることができます。
- Amazon RDS API を使用している場合、`VpcSecurityGroupIds.VpcSecurityGroupId.N` パラメータを `RestoreDBClusterFromSnapshot` アクションに含むことができます。

復元が完了し、新しい DB クラスターが使用可能になり次第、その DB クラスターを変更して VPC 設定を変更することもできます。詳細については、「Amazon Aurora DB クラスターの変更」を参照してください。

Amazon Aurora に関する考慮事項

Aurora では、DB クラスターのスナップショットを DB クラスターに復元します。

Aurora MySQL と Aurora PostgreSQL では、どちらも DB クラスターのスナップショットを Aurora Serverless DB クラスターに復元することが可能です。詳細については、「Aurora Serverless v1 DB クラスターの復元」を参照してください。

Aurora MySQL では、並行クエリのないクラスターから並行クエリのあるクラスターに DB クラスタースナップショットを復元できます。並行クエリは通常非常に大きいテーブルに使用されるため、スナップショットメカニズムは大量のデータを Aurora MySQL の並行クエリが有効なクラスターに取り込む最も速い方法です。詳細については、「Amazon Aurora MySQL のパラレルクエリの使用」を参照してください。

スナップショットからの復元

DB クラスタースナップショットから DB クラスターを復元するには、AWS Management Console、AWS CLI、または RDS API を使用します。

コンソール

DB クラスターのスナップショットから DB クラスターを復元するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[Snapshots] を選択します。
3. 復元の元にする DB クラスタースナップショットを選択します。
4. [アクション]、[スナップショットを復元] の順に選択します。

   [スナップショットを復元] ページが表示されます。
5. DB クラスターを復元する先の DB エンジンのバージョンを選択します。

   デフォルトでは、スナップショットはソース DB クラスターと同じ DB エンジンバージョン (ある場合) に復元されます。
6. [DB インスタンス識別子] に、復元する DB インスタンスの名前を入力します。
7. DB クラスターのストレージ設定など、その他の設定を指定します。

   各設定の詳細については、「Aurora DB クラスターの設定」を参照してください。
8. [Restore DB Cluster] (DB クラスターを復元) を選択します。

AWS CLI

DB クラスタースナップショットから DB クラスターを復元するには、AWS CLI の restore-db-cluster-from-snapshot コマンドを使用します。

この例では、「mydbclustersnapshot」という以前作成した DB クラスタースナップショットから復元します。mynewdbcluster という名前の新しい DB クラスターに復元します。

DB エンジンバージョンなど、他の設定も指定できます。エンジンバージョンを指定しない場合、DB クラスターはデフォルトのエンジンバージョンに復元されます。

各設定の詳細については、「Aurora DB クラスターの設定」を参照してください。

Example

Linux、macOS、Unix の場合:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mynewdbcluster \
    --snapshot-identifier mydbclustersnapshot \
    --engine aurora-mysql|aurora-postgresql
```

Windows の場合:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mynewdbcluster ^
    --snapshot-identifier mydbclustersnapshot ^
    --engine aurora-mysql|aurora-postgresql
```

DB クラスターが復元された後で、以前の DB クラスターと同じ機能が必要な場合は、DB クラスタースナップショットの作成に使用した DB クラスターによって使用されるセキュリティグループに、復元した DB クラスターを追加する必要があります。

> **Important**
>
> コンソールを使用して DB クラスターを復元する場合、Amazon RDS は自動的に使用する DB クラスターのプライマリ DB インスタンス (ライター) を作成します。AWS CLI を使用して DB クラスターを復元する場合、使用する DB クラスターのプライマリインスタンスを明示的に作成する必要があります。プライマリインスタンスは、DB クラスターで作成される

初期の DB インスタンスです。プライマリ DB インスタンスを作成しない場合、DB クラスターエンドポイントは `creating` ステータスのままです。
DB クラスターのプライマリインスタンスを作成するには、create-db-instance AWS CLI コマンドを呼び出します。`--db-cluster-identifier` オプション値として DB クラスターの名前を含めます。

RDS API

DB クラスターを DB クラスタースナップショットから復元するには、以下のパラメータを指定して RDS API オペレーション RestoreDBClusterFromSnapshot を呼び出します。

- `DBClusterIdentifier`
- `SnapshotIdentifier`

Important

コンソールを使用して DB クラスターを復元する場合、Amazon RDS は自動的に使用する DB クラスターのプライマリ DB インスタンス (ライター) を作成します。RDS API を使用して DB クラスターを復元する場合は、DB クラスターのプライマリインスタンスを明示的に作成する必要があります。プライマリインスタンスは、DB クラスターで作成される初期の DB インスタンスです。プライマリ DB インスタンスを作成しない場合、DB クラスターエンドポイントは `creating` ステータスのままです。
DB クラスターのプライマリインスタンスを作成するには、RDS API オペレーション CreateDBInstance を呼び出します。`DBClusterIdentifier` パラメータの値として DB クラスターの名前を含めます。

DB クラスターのスナップショットのコピー

Amazon Aurora を使用すると、自動バックアップまたは手動 DB クラススナップショットをコピーできます。スナップショットをコピーすると、そのコピーは手動スナップショットになります。自動バックアップまたは手動スナップショットは複数のコピーを作成できますが、各コピーには一意の識別子が必要です。

同じ AWS リージョン 内、AWS リージョン 間で、スナップショットをコピーできます。また共有スナップショットをコピーできます。

DB クラススナップショットを複数のリージョンやアカウントにコピーするには、いくつかのステップを実行する必要があります。これらのコピーアクションごとに別のステップを実行する必要があります。コピーの代わりに、他の AWS アカウントと手動スナップショットを共有することができます。詳細については、「DB クラスターのスナップショットの共有」を参照してください。

> **Note**
>
> Amazon は、Amazon Aurora のバックアップおよびスナップショットデータの保持量と保持期間に基づいて請求を行います。Aurora バックアップおよびスナップショットに伴うストレージの詳細については、「Amazon Aurora バックアップストレージの使用状況を確認する」を参照してください。Aurora ストレージの料金情報については、「Amazon RDS for Aurora の料金」を参照してください。

トピック

制限事項

スナップショットをコピーする際の制約は以下のとおりです。

- 以下の AWS リージョン との間でスナップショットをコピーすることはできません。
  - 中国 (北京)
  - 中国 (寧夏)
- AWS GovCloud (米国東部) と AWS GovCloud (US-West) の間でスナップショットをコピーすることはできます。ただし、これらの AWS GovCloud (US) リージョンと商用 AWS リージョン の間でスナップショットをコピーすることはできません。
- ターゲットスナップショットが使用可能になる前に出典スナップショットを削除すると、スナップショットはコピーされない場合があります。ターゲットスナップショットのステータスが `AVAILABLE` になったことを確認してから、出典スナップショットを削除してください。
- アカウントあたり 1 つのコピー先リージョンに対して最大 5 つのスナップショットコピーリクエストを実行できます。
- 同じソース DB インスタンスに対して複数のスナップショットコピーをリクエストすると、それらは内部的にキューに追加されます。後でリクエストされたコピーは、それ以前のスナップショットコピーが完了するまで開始されません。詳細については、AWS ナレッジセンターの「Why is my EC2 AMI or EBS snapshot creation slow? (EC2 AMI または EBS スナップショットの作成が遅いのはなぜですか?)」を参照してください。
- 関連する AWS リージョン およびデータのコピー量に応じて、リージョン間のスナップショットのコピーは完了するまでに長時間かかることがあります。場合によっては、特定のコピー元 リージョンから多数のクロスリージョンスナップショットコピーのリクエストが発生することがあります。このような場合、Amazon RDS は進行中のいくつかのコピーが完了するまで、そのコピー元リージョンからの新しいクロスリージョンコピーリクエストをキューに入れることがあります。コピーリクエストがキューに入っている間は、そのリクエストに関する進捗情報は表示されません。コピーがスタートされたときに、進捗情報は表示されます。

スナップショット保持期限

Amazon RDS は自動スナップショットをいくつかの状況で削除します。

- 保持期間の終了時。
- DB クラスターの自動スナップショットを無効にした場合。
- DB クラスターを削除した場合。

自動スナップショットをより長期間保持したい場合は、コピーを手動スナップショットとして作成しすると、削除するまで保持されます。デフォルトのストレージ領域を超える場合、手動スナップショットにAmazon RDS ストレージコストが適用される場合があります。

バックアップストレージコストの詳細については、「Amazon RDS の料金」を参照してください。

共有スナップショットのコピー

他の AWS アカウントにより共有されているスナップショットは、コピーすることができます。場合によっては、別の AWS アカウントから共有された暗号化されたスナップショットをコピーすることがあります。このような場合、スナップショットの暗号化に使用された AWS KMS key へのアクセス権が必要になります。

共有 DB クラスタースナップショットは、暗号化されているかどうかにかかわらず、同じ AWS リージョン 内でコピーのみ行えます。詳細については、「暗号化されたスナップショットの共有」を参照してください。

暗号化の処理

KMS キーを使用して暗号化されたスナップショットをコピーできます。暗号化された スナップショットをコピーする場合は、スナップショットのコピーも暗号化する必要があります。同じ AWS リージョン 内で暗号化されているスナップショットをコピーする場合、元のスナップショットと同じ KMS キーを使用してコピーを暗号化できます。または、別の KMS キー を指定することもできます。

リージョン間で、暗号化されたスナップショットをコピーする場合は、送信先の AWS リージョンで有効な KMS キーを指定する必要があります。これは、リージョン固有の KMS キーでも、マルチリージョンのキーでもかまいません。マルチリージョンの KMS キーの詳細については、「AWS KMS でマルチリージョンキーを使用する」を参照してください。

ソーススナップショットはコピープロセス全体で暗号化されたままになります。詳細については、「Amazon Aurora の暗号化された DB クラスターの制限事項」を参照してください。

> **Note**
>
> Amazon Aurora DB クラスタースナップショットの場合、暗号化されていない DB クラスタースナップショットをコピー時に暗号化することはできません。

増分スナップショットコピー

Aurora は、差分スナップショットコピーをサポートしていません。Aurora DB クラススナップショットコピーは常にフルコピーです。フルスナップショットコピーには、DB クラスターを復元するために必要なデータやメタデータすべてが含まれます。

クロスリージョンのスナップショットコピー

AWS リージョン 間で DB クラススナップショットをコピーできます。ただし、クロスリージョンのスナップショットコピーには、特定の制約と考慮事項があります。

関連する AWS リージョン およびデータのコピー量に応じて、リージョン間のスナップショットのコピーは完了するまでに長時間かかることがあります。

場合によっては、特定のコピー元 AWS リージョン からの多数のクロスリージョンスナップショットコピーのリクエストが発生することがあります。このような場合、Amazon RDS は進行中のいくつかのコピーが完了するまで、そのコピー元 AWS リージョン からの新しいクロスリージョンコピーリクエストをキューに入れることがあります。コピーリクエストがキューに入っている間は、そのリクエストに関する進捗情報は表示されません。コピーがスタートされたときに、進捗情報は表示されます。

AWS Backup を使用してクロスリージョンスナップショットをコピーすると、コピーはフルコピーとなり、データ転送料金は増分課金になります。詳細については、「AWS Backup デベロッパーガイド」の「AWS リージョン間でのバックアップコピーの作成」を参照してください。

パラメータグループに関する考慮事項

リージョン間でスナップショットをコピーすると、コピーにはコピー元の DB クラスターで使用されているパラメータグループは含まれません。スナップショットを復元して新しい DB クラスターを作成すると、その DB クラスターは、作成された AWS リージョン のデフォルトのパラメータグループを取得します。新しい DB インスタンスクラスターでコピー元と同じパラメータを使用するには、以下の操作を行います。

1. DB クラスターパラメータグループを作成するには、コピー先の AWS リージョン で元の DB クラスターと同じ設定を使用します。新しい AWS リージョン に既にある場合は、それを使用できます。
2. コピー先の AWS リージョン でスナップショットを復元したら、新しい DB クラスターを変更し、前のステップからの新規または既存のパラメータグループを追加します。

DB クラスターのスナップショットのコピー

DB クラススナップショットをコピーするには、このトピックの手順を使用します。コピー元のデータベースエンジンが Aurora である場合、スナップショットは DB クラススナップショットになります。

AWS アカウントごとに、AWS リージョン 間で同時に 5 つまでの DB クラススナップショットをコピーできます。暗号化されている DB クラススナップショットと暗号化されていない DB クラススナップショットのどちらのコピーもサポートされています。別の AWS リージョン に DB クラススナップショットをコピーする場合には、その AWS リージョン に保持されている手動 DB クラススナップショットを作成します。コピー元の AWS リージョン から DB クラススナップショットをコピーすると、Amazon RDS ではデータ転送料金が発生します。

データ転送料金の詳細については、「Amazon RDS の料金」を参照してください。

DB クラススナップショットのコピーが新しい AWS リージョン に作成されると、その DB クラススナップショットのコピーは、AWS リージョン にある他のすべての DB クラススナップショットと同じように動作します。

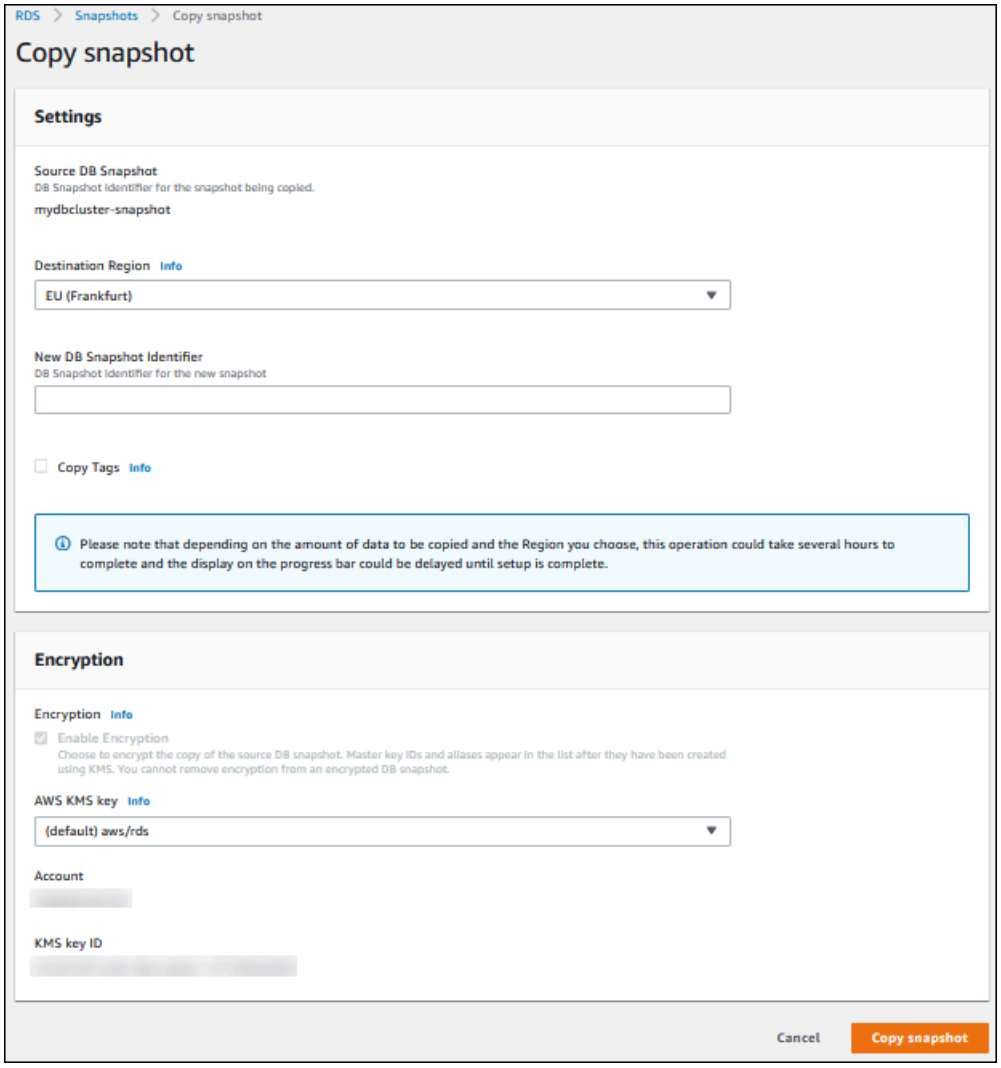
コンソール

この手順は、同じ AWS リージョン 内、またはリージョン間で、暗号化されている、あるいは暗号化されていない DB クラススナップショットをコピーするために使用されます。

コピーオペレーションが進行中にキャンセルするには、DB クラススナップショットが [copying] ステータスの間にターゲット DB クラススナップショットを削除します。

DB クラススナップショットをコピーするには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Snapshots] を選択します。
3. コピーする DB クラススナップショットを選択します。
4. [アクション] で、[スナップショットをコピー] を選択します。[スナップショットをコピー] ページが表示されます。



5. (オプション) 別の AWS リージョン に DB クラスタースナップショットをコピーするには、[Destination Region] (コピー先リージョン) に、その AWS リージョン を選択します。
6. [New DB Snapshot Identifier (新しい DB スナップショットの識別子)] に、DB クラスタースナップショットのコピーの名前を入力します。
7. スナップショットからスナップショットのコピーにタグと値をコピーするには、[Copy Tags] を選択します。
8. [Copy Snapshot] を選択します。

AWS CLI または Amazon RDS API を使用して、暗号化されていない DB クラスタースナップショットをコピーする

AWS CLI または Amazon RDS API を使用して、暗号化されていない DB クラスタースナップショットをコピーするには、以下のセクションの手順を使用します。

コピーオペレーションが進行中にキャンセルするには、DB クラススナップショットが [copying] ステータスの間に `--target-db-cluster-snapshot-identifier` または `TargetDBClusterSnapshotIdentifier` により識別されるターゲット DB クラススナップショットを削除します。

AWS CLI

AWS CLI copy-db-cluster-snapshot コマンドを使用して、DB スナップショットをコピーできます。スナップショットを別の AWS リージョン にコピーする場合、スナップショットのコピー先の AWS リージョン 中でコマンドを実行します。

暗号化されていない DB クラススナップショットのコピーには、以下のオプションが使用されます。

- `--source-db-cluster-snapshot-identifier` - コピーする暗号化された DB クラススナップショットの識別子。別の AWS リージョン にスナップショットをコピーする場合、この識別子は 元の AWS リージョン のための ARN 形式でなければなりません。
- `--target-db-cluster-snapshot-identifier` - DB クラススナップショットの新しいコピーの識別子。

次のコードは、コマンドが実行される AWS リージョン 内にある `myclustersnapshotcopy` という名前の DB クラススナップショット `arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805` のコピーを作成します。コピーが作成されると、元のスナップショットのすべてのタグはコピーされたスナップショットにコピーされます。

Example

Linux、macOS、Unix の場合:

```
aws rds copy-db-cluster-snapshot \
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805 \
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy \
  --copy-tags
```

Windows の場合:

```
aws rds copy-db-cluster-snapshot ^
```

```
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:123456789012:cluster-
snapshot:aurora-cluster1-snapshot-20130805 ^
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy ^
  --copy-tags
```

RDS API

DB クラススナップショットをコピーするには、Amazon RDS API の CopyDBClusterSnapshot オペレーションを使用します。スナップショットを別の AWS リージョン にコピーする場合、スナップショットのコピー先の AWS リージョン でアクションを実行します。

暗号化されていない DB クラススナップショットのコピーには、以下のパラメータが使用されます。

- SourceDBClusterSnapshotIdentifier - コピーする暗号化された DB クラススナップショットの識別子。別の AWS リージョン にスナップショットをコピーする場合、この識別子は元の AWS リージョン のための ARN 形式でなければなりません。
- TargetDBClusterSnapshotIdentifier - DB クラススナップショットの新しいコピーの識別子。

次のコードは、米国西部 (北カリフォルニア) リージョンで arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805 という名前のスナップショット myclustersnapshotcopy のコピーを作成します。コピーが作成されると、元のスナップショットのすべてのタグはコピーされたスナップショットにコピーされます。

Example

```
https://rds.us-west-1.amazonaws.com/
   ?Action=CopyDBClusterSnapshot
   &CopyTags=true
   &SignatureMethod=HmacSHA256
   &SignatureVersion=4
   &SourceDBSnapshotIdentifier=arn%3Aaws%3Ards%3Aus-east-1%3A123456789012%3Acluster-
snapshot%3Aaurora-cluster1-snapshot-20130805
   &TargetDBSnapshotIdentifier=myclustersnapshotcopy
   &Version=2013-09-09
   &X-Amz-Algorithm=AWS4-HMAC-SHA256
   &X-Amz-Credential=AKIADQKE4SARGYLE/20140429/us-west-1/rds/aws4_request
   &X-Amz-Date=20140429T175351Z
   &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
```

```
&X-Amz-Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

AWS CLI または Amazon RDS API を使用して、暗号化されている DB クラスタースナップショットをコピーする

AWS CLI または Amazon RDS API を使用して、暗号化されている DB クラスタースナップショットをコピーするには、以下のセクションの手順を使用します。

コピーオペレーションが進行中にキャンセルするには、DB クラスタースナップショットが [copying] ステータスの間に `--target-db-cluster-snapshot-identifier` または `TargetDBClusterSnapshotIdentifier` により識別されるターゲット DB クラスタースナップショットを削除します。

AWS CLI

AWS CLI copy-db-cluster-snapshot コマンドを使用して、DB スナップショットをコピーできます。スナップショットを別の AWS リージョン にコピーする場合、スナップショットのコピー先の AWS リージョン 中でコマンドを実行します。

暗号化されている DB クラスタースナップショットのコピーには、以下のオプションが使用されます。

- `--source-db-cluster-snapshot-identifier` - コピーする暗号化された DB クラスタースナップショットの識別子。別の AWS リージョン にスナップショットをコピーする場合、この識別子は 元の AWS リージョン のための ARN 形式でなければなりません。
- `--target-db-cluster-snapshot-identifier` - 暗号化された DB クラスタースナップショットの新しいコピーの識別子。
- `--kms-key-id` - DB クラスタースナップショットのコピーを暗号化するのに使用するキーの KMS キー識別子。

  DB クラスタースナップショットが暗号化されており、同じ AWS リージョン 内でスナップショットをコピーして、コピーを暗号化する新しい KMS キーを指定する場合は、オプションでこのオプションを使用できます。それ以外の場合、DB クラスタースナップショットのコピーが出典 DB クラスタースナップショットと同じ KMS キーで暗号化されます。

  DB クラスタースナップショットが暗号化されており、スナップショットを別の AWS リージョンにコピーする場合は、このオプションを使用する必要があります。その場合、送信先 AWS リージョン で KMS キーを指定する必要があります。

次のコード例では、米国西部 (オレゴン) リージョンから US East (N. Virginia) リージョンに暗号化された DB クラスタースナップショットをコピーします。コマンドは US East (N. Virginia) リージョンで呼び出されます。

Example

Linux、macOS、Unix の場合:

```
aws rds copy-db-cluster-snapshot \
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-west-2:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20161115 \
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy \
  --kms-key-id my-us-east-1-key
```

Windows の場合:

```
aws rds copy-db-cluster-snapshot ^
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-west-2:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20161115 ^
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy ^
  --kms-key-id my-us-east-1-key
```

`--source-region` パラメータは、AWS GovCloud (米国東部) と AWS GovCloud (米国西部) リージョン間で暗号化された DB クラスタースナップショットをコピーする場合に必要です。`--source-region` には、ソース DB インスタンスの AWS リージョン を指定します。`source-db-cluster-snapshot-identifier` で指定した AWS リージョン は、`--source-region` として指定した AWS リージョン と一致している必要があります。

`--source-region` を指定しない場合、`--pre-signed-url` の値を指定する必要があります。署名付きの URL は、ソースの AWS リージョン で呼び出される `copy-db-cluster-snapshot` コマンドに対する、署名バージョン 4 で署名されたリクエストを含む URL です。`pre-signed-url` オプションの詳細については、AWS CLI コマンドリファレンスの「copy-db-cluster-snapshot」を参照してください。

RDS API

DB クラスタースナップショットをコピーするには、Amazon RDS API の CopyDBClusterSnapshot オペレーションを使用します。スナップショットを別の AWS リージョン にコピーする場合、スナップショットのコピー先の AWS リージョン でアクションを実行します。

暗号化されている DB クラススナップショットのコピーには、以下のパラメータが使用されます。

- `SourceDBClusterSnapshotIdentifier` - コピーする暗号化された DB クラススナップショットの識別子。別の AWS リージョン にスナップショットをコピーする場合、この識別子は元の AWS リージョン のための ARN 形式でなければなりません。
- `TargetDBClusterSnapshotIdentifier` - 暗号化された DB クラススナップショットの新しいコピーの識別子。
- `KmsKeyId` - DB クラススナップショットのコピーを暗号化するのに使用するキーの KMS キー識別子。

  DB クラススナップショットが暗号化されており、同じ AWS リージョン 内でスナップショットをコピーして、コピーの暗号化に使用する新しい KMS キーを指定する場合は、オプションでこのパラメータを使用できます。それ以外の場合、DB クラススナップショットのコピーが出典 DB クラススナップショットと同じ KMS キーで暗号化されます。

  DB クラススナップショットが暗号化されており、スナップショットを別の AWS リージョンにコピーする場合は、このパラメータを使用する必要があります。その場合、送信先 AWS リージョン で KMS キーを指定する必要があります。
- `PreSignedUrl` - スナップショットを別の AWS リージョン にコピーする場合、`PreSignedUrl` パラメータを指定する必要があります。`PreSignedUrl` の値は、DB クラススナップショットのコピー元のソース AWS リージョン で呼び出される `CopyDBClusterSnapshot` のアクションの署名バージョン 4 で署名された URL でなければなりません。署名付き URL の詳細については、「CopyDBClusterSnapshot」を参照してください。

次のコード例では、米国西部 (オレゴン) リージョンから US East (N. Virginia) リージョンに暗号化された DB クラススナップショットをコピーします。アクションは US East (N. Virginia) リージョンで呼び出されます。

Example

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CopyDBClusterSnapshot
    &KmsKeyId=my-us-east-1-key
    &PreSignedUrl=https%253A%252F%252Frds.us-west-2.amazonaws.com%252F
         %253FAction%253DCopyDBClusterSnapshot
         %2526DestinationRegion%253Dus-east-1
         %2526KmsKeyId%253Dmy-us-east-1-key
```

```
        %2526SourceDBClusterSnapshotIdentifier%253Darn%25253Aaws%25253Ards
%25253Aus-west-2%25253A123456789012%25253Acluster-snapshot%25253Aaurora-cluster1-
snapshot-20161115
        %2526SignatureMethod%253DHmacSHA256
        %2526SignatureVersion%253D4
        %2526Version%253D2014-10-31
        %2526X-Amz-Algorithm%253DAWS4-HMAC-SHA256
        %2526X-Amz-Credential%253DAKIADQKE4SARGYLE%252F20161117%252Fus-west-2%252Frds
%252Faws4_request
        %2526X-Amz-Date%253D20161117T215409Z
        %2526X-Amz-Expires%253D3600
        %2526X-Amz-SignedHeaders%253Dcontent-type%253Bhost%253Buser-agent%253Bx-amz-
content-sha256%253Bx-amz-date
        %2526X-Amz-Signature
%253D255a0f17b4e717d3b67fad163c3ec26573b882c03a65523522cf890a67fca613
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &SourceDBClusterSnapshotIdentifier=arn%3Aaws%3Ards%3Aus-
west-2%3A123456789012%3Acluster-snapshot%3Aaurora-cluster1-snapshot-20161115
    &TargetDBClusterSnapshotIdentifier=myclustersnapshotcopy
    &Version=2014-10-31
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
    &X-Amz-Date=20161117T221704Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=da4f2da66739d2e722c85fcfd225dc27bba7e2b8dbea8d8612434378e52adccf
```

PreSignedUrl パラメータは、AWS GovCloud (米国東部) と AWS GovCloud (米国西部) リージョン間で暗号化された DB クラススナップショットをコピーする場合に必要です。PreSignedUrl の値は、DB クラススナップショットのコピー元のソース AWS リージョン で呼び出される CopyDBClusterSnapshot オペレーションに対して署名バージョン 4 で署名された URL でなければなりません。署名付き URL の詳細については、Amazon RDS API リファレンスにある「CopyDBClusterSnapshot」を参照してください。

署名付き URL を手動ではなく自動的に生成するには、--source-region オプションを使用して AWS CLI copy-db-cluster-snapshot コマンドを使用します。

アカウント間での DB クラススナップショットのコピー

Amazon RDS API の ModifyDBClusterSnapshotAttribute アクションと CopyDBClusterSnapshot アクションを使用することで、他の AWS アカウントが、指定した DB

クラススナップショットをコピーできるようにすることができます。DB クラススナップショットは、同じ AWS リージョン のアカウント間でのみコピーできます。クロスアカウントコピーは、アカウント A がコピー可能なスナップショットを作成した後、アカウント B がそれをコピーするというプロセスになります。

1. アカウント A を使用し、ModifyDBClusterSnapshotAttribute パラメータに **restore** を、AttributeName パラメータにアカウント B の ID を指定して、ValuesToAdd を呼び出します。
2. (スナップショットが暗号化されている場合) アカウント A を使用し、まずアカウント B の ARN を Principal として追加してから kms:CreateGrant アクションを許可し、KMS キーのキーポリシーを更新します。
3. (スナップショットが暗号化されている場合) アカウント B を使用し、ユーザーを選択または作成して、KMS キーを使用して暗号化されている DB クラススナップショットのコピーを許可する IAM ポリシーをそのユーザーにアタッチします。
4. アカウント B を使用して CopyDBClusterSnapshot を呼び出し、SourceDBClusterSnapshotIdentifier パラメータを使用して、コピーする DB クラススナップショットの ARN を指定します (アカウント A の ID を含める必要があります)。

DB クラススナップショットを復元するアクセス許可が付与されているすべての AWS アカウントを一覧表示するには、DescribeDBSnapshotAttributes または DescribeDBClusterSnapshotAttributes API オペレーションを使用します。

AWS アカウントの共有アクセス権限を削除するには、ModifyDBSnapshotAttribute を ModifyDBClusterSnapshotAttribute に設定して、AttributeName パラメータで削除するアカウントの ID を設定し、restore アクションまたは ValuesToRemove アクションを使用します。

暗号化されていない DB クラススナップショットを別のアカウントにコピーする

暗号化されていない DB クラススナップショットを同じ AWS リージョン 内の別のアカウントにコピーするには、以下の手順を使用します。

1. DB クラススナップショットの出典アカウントで、ModifyDBClusterSnapshotAttribute パラメータに **restore** を指定して AttributeName パラメータにターゲットアカウントの ID を指定し、ValuesToAdd を呼び出します。

アカウント 987654321 を使用して次の例を実行すると、2 つの AWS アカウント識別子 123451234512 および 123456789012 が manual-snapshot1 という名前の DB クラスタースナップショットを復元できるようになります。

```
https://rds.us-west-2.amazonaws.com/
 ?Action=ModifyDBClusterSnapshotAttribute
 &AttributeName=restore
 &DBClusterSnapshotIdentifier=manual-snapshot1
 &SignatureMethod=HmacSHA256&SignatureVersion=4
 &ValuesToAdd.member.1=123451234512
 &ValuesToAdd.member.2=123456789012
 &Version=2014-10-31
 &X-Amz-Algorithm=AWS4-HMAC-SHA256
 &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
 &X-Amz-Date=20150922T220515Z
 &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
 &X-Amz-Signature=ef38f1ce3dab4e1dbf113d8d2a265c67d17ece1999ffd36be85714ed36dddbb3
```

2. ターゲットアカウントで、CopyDBClusterSnapshot を呼び出し、SourceDBClusterSnapshotIdentifier パラメータを使用して、コピーする DB クラスタースナップショットの ARN を指定します (出典アカウントの ID を含める必要があります)。

   アカウント 123451234512 を使用して次の例を実行すると、DB クラスターアカウント aurora-cluster1-snapshot-20130805 がアカウント 987654321 からコピーされ、dbclustersnapshot1 という名前の DB クラスタースナップショットが作成されます。

```
https://rds.us-west-2.amazonaws.com/
   ?Action=CopyDBClusterSnapshot
   &CopyTags=true
   &SignatureMethod=HmacSHA256
   &SignatureVersion=4
   &SourceDBClusterSnapshotIdentifier=arn:aws:rds:us-west-2:987654321:cluster-
snapshot:aurora-cluster1-snapshot-20130805
   &TargetDBClusterSnapshotIdentifier=dbclustersnapshot1
   &Version=2013-09-09
   &X-Amz-Algorithm=AWS4-HMAC-SHA256
   &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
   &X-Amz-Date=20140429T175351Z
   &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-
date
```

```
   &X-Amz-
Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

暗号化されている DB クラススナップショットを別のアカウントにコピーする

暗号化されている DB クラススナップショットを同じ AWS リージョン 内の別のアカウントにコピーするには、以下の手順を使用します。

1. DB クラススナップショットの出典アカウントで、ModifyDBClusterSnapshotAttribute パラメータに **restore** を指定して AttributeName パラメータにターゲットアカウントの ID を指定し、ValuesToAdd を呼び出します。

   アカウント 987654321 を使用して次の例を実行すると、2 つの AWS アカウント識別子 123451234512 および 123456789012 が manual-snapshot1 という名前の DB クラススナップショットを復元できるようになります。

   ```
   https://rds.us-west-2.amazonaws.com/
    ?Action=ModifyDBClusterSnapshotAttribute
    &AttributeName=restore
    &DBClusterSnapshotIdentifier=manual-snapshot1
    &SignatureMethod=HmacSHA256&SignatureVersion=4
    &ValuesToAdd.member.1=123451234512
    &ValuesToAdd.member.2=123456789012
    &Version=2014-10-31
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
    &X-Amz-Date=20150922T220515Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=ef38f1ce3dab4e1dbf113d8d2a265c67d17ece1999ffd36be85714ed36dddbb3
   ```

2. DB クラススナップショットのソースアカウントで、暗号化された DB クラススナップショットと同じ AWS リージョン にカスタム KMS キーを作成します。カスタマーマネージドキーの作成中に、ターゲットの AWS アカウント にそのキーへのアクセス権を付与します。詳細については、「カスタマーマネージドキーを作成し、そのキーへのアクセス権を付与する」を参照してください。
3. スナップショットをターゲット AWS アカウント にコピーして共有します。詳細については、「ソースアカウントからスナップショットをコピーして共有する」を参照してください。

4. ターゲットアカウントで、CopyDBClusterSnapshot を呼び出し、SourceDBClusterSnapshotIdentifier パラメータを使用して、コピーする DB クラスタースナップショットの ARN を指定します (出典アカウントの ID を含める必要があります)。

   アカウント 123451234512 を使用して次の例を実行すると、DB クラスターアカウント aurora-cluster1-snapshot-20130805 がアカウント 987654321 からコピーされ、dbclustersnapshot1 という名前の DB クラスタースナップショットが作成されます。

```
https://rds.us-west-2.amazonaws.com/
   ?Action=CopyDBClusterSnapshot
   &CopyTags=true
   &SignatureMethod=HmacSHA256
   &SignatureVersion=4
   &SourceDBClusterSnapshotIdentifier=arn:aws:rds:us-west-2:987654321:cluster-
snapshot:aurora-cluster1-snapshot-20130805
   &TargetDBClusterSnapshotIdentifier=dbclustersnapshot1
   &Version=2013-09-09
   &X-Amz-Algorithm=AWS4-HMAC-SHA256
   &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
   &X-Amz-Date=20140429T175351Z
   &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-
date
   &X-Amz-
Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

DB クラスターのスナップショットの共有

Amazon RDS を使用すると、次の方法で手動 DB クラススナップショットを共有できます。

- 手動 DB クラススナップショットを共有すると、暗号化されているかいないかに関係なく、権限のある AWS アカウントがスナップショットをコピーできるようになります。
- 手動 DB クラススナップショットを共有すると、暗号化されているかいないかに関係なく、権限を持つ AWS アカウントが DB クラスターをコピーしてそこから復元するのではなく、スナップショットから DB クラスターを直接復元できるようになります。

> **Note**
>
> 自動 DB クラススナップショットを共有するには、自動化されたスナップショットをコピーしてそのコピーを共有することで、手動 DB クラススナップショットを作成します。このプロセスは、AWS Backup で生成されたリソースにも適用されます。

スナップショットのコピーの詳細については、「DB クラスターのスナップショットのコピー」を参照してください。DB クラススナップショットから DB インスタンスを復元する方法については、「DB クラスターのスナップショットからの復元」を参照してください。

DB クラススナップショットから DB クラスターを復元する方法の詳細については、「Aurora DB クラスターのバックアップと復元の概要」を参照してください。

手動スナップショットを最大 20 のその他の AWS アカウント と共有することができます。

手動スナップショットを他の AWS アカウント と共有する場合には、次の制限が適用されます。

- AWS Command Line Interface (AWS CLI) または Amazon RDS を使用して共有スナップショットから DB クラスターを復元する際、スナップショット識別子として共有 DB スナップショットの Amazon リソースネーム (ARN) を指定する必要があります。

目次

スナップショットの共有

AWS Management Console、AWS CLI、または RDS API を使用して、DB クラススナップショットを共有できます。

コンソール

Amazon RDS コンソールを使用して、手動 DB クラススナップショットを最大 20 の AWS アカウント と共有することができます。また、コンソールを使用して、手動 スナップショットの 1 つ以上のアカウントとの共有を停止することもできます。

Amazon RDS コンソールを使用して、手動 DB クラススナップショットを共有するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Snapshots] を選択します。
3. 共有する手動スナップショットを選択します。
4. [Actions] (アクション) で、[Share snapshot] (スナップショットの共有) を選択します。
5. [DB snapshot visibility] で次のいずれかのオプションを選択します。
   - ソースが暗号化されていない場合、[パブリック] を選択して、すべての AWS アカウントが DB クラスターをマニュアル DB クラススナップショットから復元できるようにするか、[プライベート] を選択して、指定した AWS アカウント だけが、DB クラスターをマニュアル DB クラススナップショットから復元できるようにします。

> **Warning**
>
> [DB スナップショットの可視性] を [パブリック] に設定すると、すべての AWS アカウント が手動 DB クラススナップショットから DB クラスターを復元し、データへ

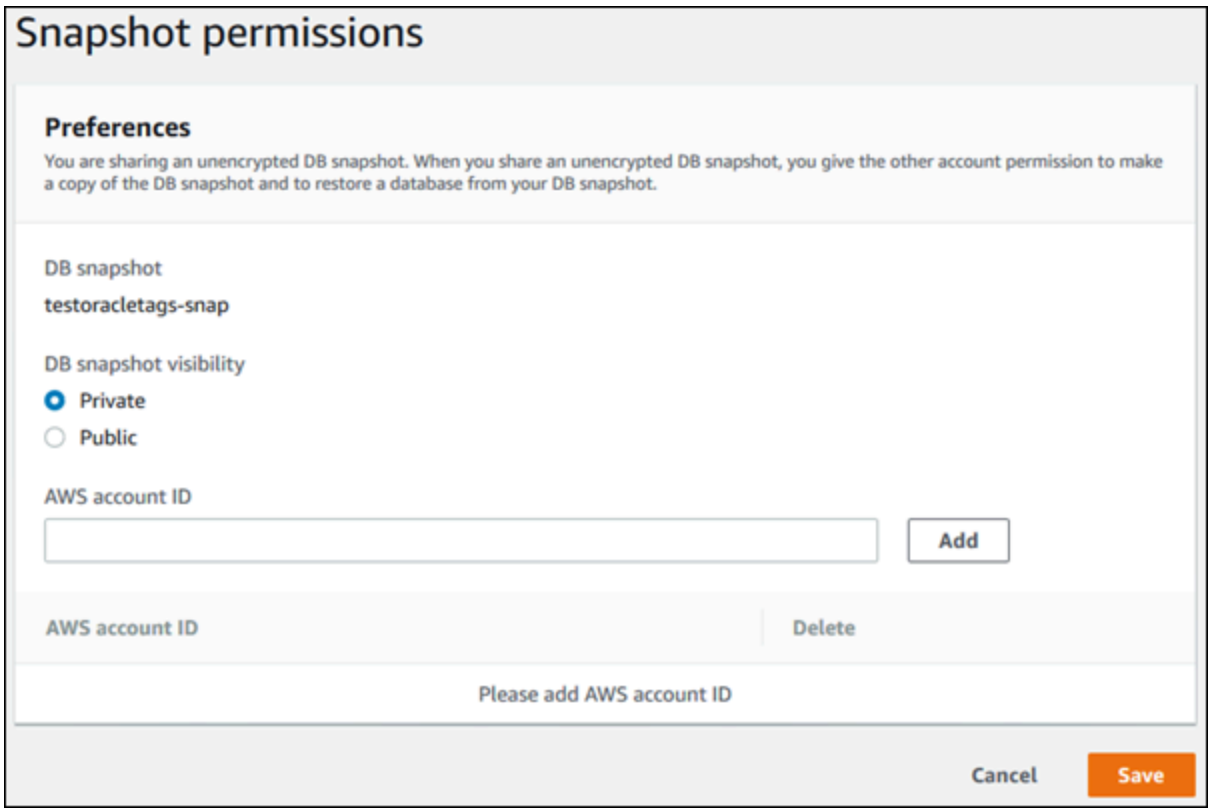
アクセスすることができるようになります。プライベート情報を含む手動 DB クラスタースナップショットは、[Public] として共有しないでください。
詳細については、「公開スナップショットの共有」を参照してください。

- 出典 DB クラスターが暗号化されている場合、暗号化されているスナップショットはパブリックとして共有できないため、[DB snapshot visibility] が [Private] に設定されます。

> **Note**
>
> デフォルトの AWS KMS key で暗号化されたスナップショットは共有できません。この問題を回避する方法については、「暗号化されたスナップショットの共有」を参照してください。

6. [AWS アカウント ID] では、マニュアルスナップショットからの DB クラスターの復元を許可するアカウントの AWS アカウント 識別子を入力してから、[追加] を選択します。この操作を繰り返して、AWS アカウント 識別子を最大 20 AWS アカウント 追加できます。

   アクセス権限が付与されたアカウントのリストに AWS アカウント 識別子を誤って追加した場合には、その AWS アカウント ID の右側にある [削除] を選択すれば、削除することができます。



7. 手動 スナップショットの復元を許可する AWS アカウント 識別子をすべて追加したら、[保存] を選択して変更を保存します。

AWS CLI

DB クラススナップショットを共有するには、`aws rds modify-db-cluster-snapshot-attribute` コマンドを使用します。`--values-to-add` のパラメータを使用して、手動スナップショットの復元が許可されている AWS アカウント のための ID リストを追加します。

Example スナップショットを 1 つのアカウントで共有する

次の例では、AWS アカウント 識別子 123456789012 が `cluster-3-snapshot` という名前の DB クラススナップショットを復元できるようにします。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier cluster-3-snapshot \
--attribute-name restore \
--values-to-add 123456789012
```

Windows の場合:

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier cluster-3-snapshot ^
--attribute-name restore ^
--values-to-add 123456789012
```

Example 複数のアカウントでスナップショットを共有する

次の例では、2 つの AWS アカウント 識別子 111122223333 および 444455556666 が `manual-cluster-snapshot1` という名前の DB クラススナップショットを復元できるようにします。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier manual-cluster-snapshot1 \
--attribute-name restore \
--values-to-add {"111122223333","444455556666"}
```

Windows の場合:

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier manual-cluster-snapshot1 ^
--attribute-name restore ^
--values-to-add "[\"111122223333\",\"444455556666\"]"
```

> **Note**
>
> Windows コマンドプロンプトを使用する場合、JSON コードでは、二重引用符 (") の前にバックスラッシュ (\) を付けてエスケープする必要があります。

スナップショットの復元が有効になっている AWS アカウント を一覧表示するには、`describe-db-cluster-snapshot-attributes` AWS CLI コマンドを使用します。

RDS API

Amazon RDS API を使用することで、手動 DB クラススナップショットを他の AWS アカウント と共有することもできます。そのためには、`ModifyDBClusterSnapshotAttribute` オペレーションを呼び出します。`AttributeName` に `restore` を指定し、`ValuesToAdd` パラメータを使用して、手動 スナップショットの復元が許可されている AWS アカウント の ID のリストを追加します。

手動スナップショットをパブリックにして、すべての AWS アカウント による復元を可能にするには、値 `all` を使用します。ただし、すべての AWS アカウント には利用させたくないプライベート情報を含む手動スナップショットについては、値 `all` を追加しないように注意してください。また、暗号化されているスナップショットでは `all` を指定しないでください。そのようなスナップショットをパブリックにすることはできないためです。

スナップショットを復元することが許可されているすべての AWS アカウント を一覧表示するには、`DescribeDBClusterSnapshotAttributes` API オペレーションを使用します。

公開スナップショットの共有

暗号化されていない手動スナップショットをパブリックとして共有できます。これにより、このスナップショットをすべての AWS アカウント が使用できるようになります。スナップショットを公開として共有する場合には、公開スナップショットにプライベート情報が含まれないように注意してください。

スナップショットがパブリックに共有されると、スナップショットのコピーと DB インスタンスクラスター作成のための AWS アカウント アクセス許可がすべて付与されます。

他のアカウントが所有する公開スナップショットのバックアップストレージについては課金されません。課金されるのは、所有しているスナップショットに対してのみです。

公開スナップショットをコピーする場合は、そのコピーを所有します。スナップショットコピーのバックアップストレージに対しては課金されます。DB クラスターを公開スナップショットから作成する場合、その DB クラスターに対して課金されます。Amazon Aurora の料金情報については、Aurora の料金表ページを参照してください。

削除できるのは、所有している公開スナップショットのみです。共有またはパブリックスナップショットを削除するには、そのスナップショットを所有する AWS アカウント にログインできることを確認してください。

他の AWS アカウント が所有する公開スナップショットの表示

Amazon RDS コンソールの [スナップショット] ページにある [パブリック] タブの特定の AWS リージョン で、他のアカウントが所有する公開スナップショットを表示できます。(自分のアカウントが所有する) スナップショットは、このタブには表示されません。

公開スナップショットを表示するには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Snapshots] を選択します。
3. [Public (公開)] タブを選択します。

   公開スナップショットが表示されます。[Owner (所有者)] 列に、公開スナップショットを所有しているアカウントが表示されます。

   > **Note**
   >
   > ページ設定を変更する必要がある場合、[Public snapshots (公開スナップショット)] リストの右上にあるギヤアイコンを選択して、この列を表示させます。

独自の公開スナップショットの表示

次の AWS CLI コマンド (Unix のみ) を使用して、特定の AWS リージョン の AWS アカウント が所有する公開スナップショットを表示させることができます。

```
aws rds describe-db-cluster-snapshots --snapshot-type public --include-public |
 grep account_number
```

公開スナップショットがある場合、次の例のような出力が返されます。

```
"DBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:myclustersnapshot1",
"DBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:myclustersnapshot2",
```

廃止された DB エンジンバージョンからのパブリックスナップショットの共有

廃止された DB エンジンバージョンからのパブリックスナップショットの復元またはコピーはサポートされていません。サポートされていない既存のパブリックスナップショットを復元またはコピーできるようにするには、次の手順を実行します。

1. スナップショットをプライベートとしてマークします。
2. スナップショットを復元します。
3. 復元した DB クラスターを、サポートされているエンジンバージョンにアップグレードします。
4. スナップショットを作成します。
5. スナップショットをパブリックに再共有します。

暗号化されたスナップショットの共有

「Amazon Aurora リソースの暗号化」で説明しているように、AES-256 暗号化アルゴリズムを使用して暗号化された「保存中」の DB クラスタースナップショットを共有できます。

以下の制限は、暗号化されたスナップショットの共有に適用されます。

- 暗号化されたスナップショットをパブリックとして共有することはできません。
- スナップショットを共有する AWS アカウント のデフォルト KMS キーを使って暗号化されたスナップショットを共有することはできません。

デフォルトの KMS キーの問題を回避するには、以下のタスクを実行します。

1. カスタマーマネージドキーを作成し、そのキーへのアクセス権を付与する.
2. ソースアカウントからスナップショットをコピーして共有する.

3. ターゲットアカウントに共有したスナップショットをコピーします。.

カスタマーマネージドキーを作成し、そのキーへのアクセス権を付与する

まず、暗号化された DB クラススナップショットと同じ AWS リージョン にカスタム KMS キーを作成します。カスタマーマネージドキーの作成中に、別の AWS アカウント にそのキーへのアクセス権を付与します。

カスタマーマネージドキーを作成し、そのキーへのアクセス権を付与するには

1. ソース AWS アカウント から AWS Management Console にサインインします。
2. AWS KMS コンソール (https://console.aws.amazon.com/kms) を開きます。
3. AWS リージョン を変更するには、ページの右上隅にあるリージョンセレクターを使用します。
4. ナビゲーションペインで、[カスタマーマネージドキー] を選択します。
5. [Create key] (キーの作成) を選択します。
6. [キーの設定] ページで、次の操作を行います。
   a. [キータイプ] として、[対称] を選択します。
   b. [キーの使用] で、[暗号化および復号化] を選択します。
   c. [詳細オプション] を展開します。
   d. [キーマテリアルオリジン] として、[KMS] を選択します。
   e. [リージョン] で、[単一リージョンキー] を選択します。
   f. [Next] を選択します。
7. [ラベルを追加] ページで以下のように操作します。
   a. [エイリアス] には、**`share-snapshot`** のように KMS キーの表示名を入力します。
   b. (オプション) KMS キーの説明を入力します。
   c. (オプション) KMS キーにタグを追加します。
   d. [Next] を選択します。
8. [キー管理アクセス許可の定義] ページで、[次へ] をクリックします。
9. [キーの使用アクセス許可の定義] ページで、次の操作を行います。
   a. [その他の AWS アカウント] では、[別の AWS アカウント の追加] を選択します。
   b. アクセスを許可する AWS アカウント の ID を入力します。

複数の AWS アカウント にアクセス権を付与できます。

c. [Next] を選択します。

10. KMS キーを確認し、[終了] を選択します。

ソースアカウントからスナップショットをコピーして共有する

次に、カスタマーマネージドキーを使用して、ソース DB クラススナップショットを新しいスナップショットにコピーします。次に、ターゲット AWS アカウント と共有します。

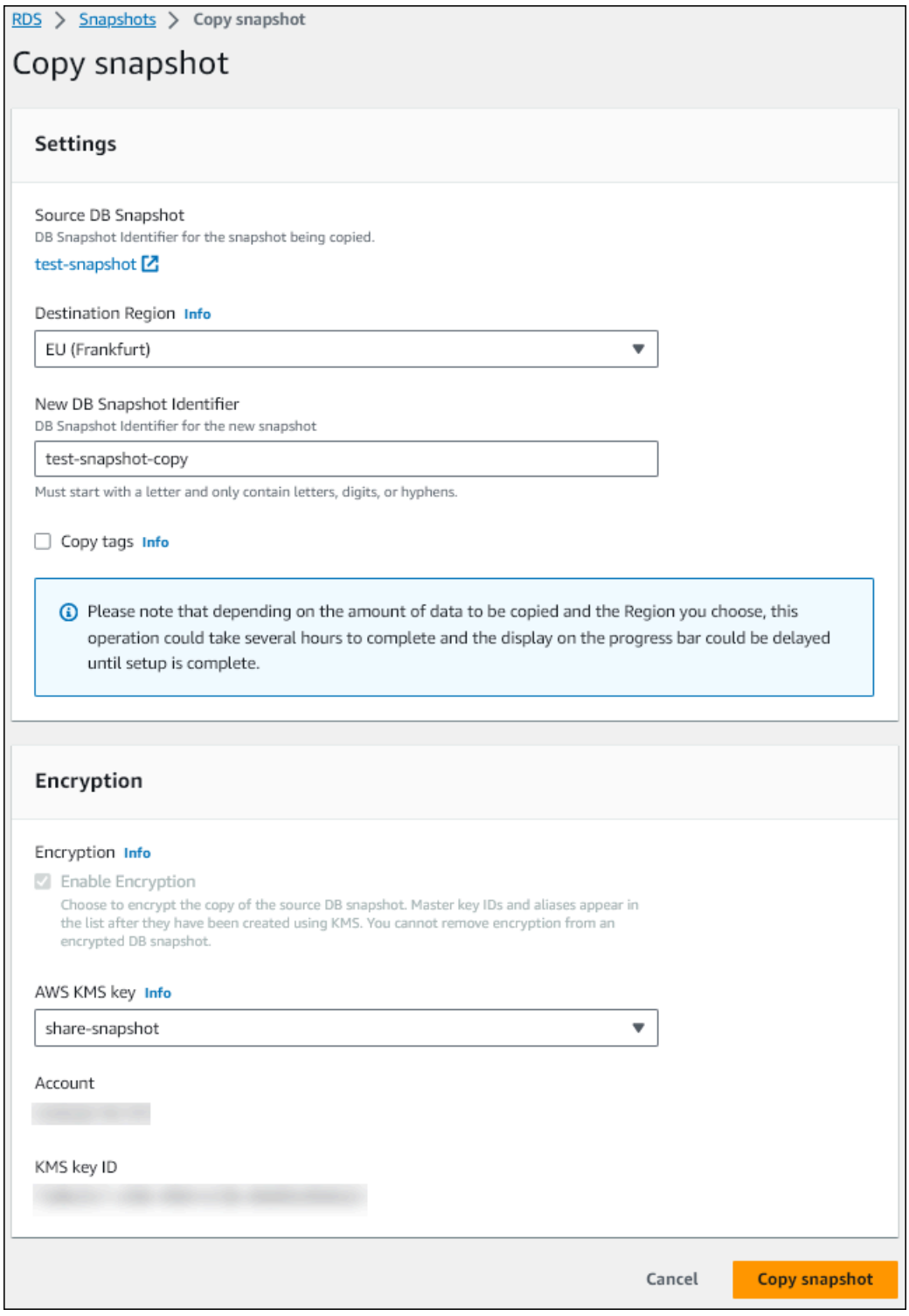
スナップショットをコピーして共有するには

1. ソース AWS アカウント から AWS Management Console にサインインします。
2. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
3. ナビゲーションペインで、[Snapshots] を選択します。
4. コピーする DB クラススナップショットを選択します。
5. [アクション] で、[スナップショットをコピー] を選択します。
6. [スナップショットのコピー] ページで、次の操作を行います。

   a. [送信先リージョン] で、前の手順でカスタマーマネージドキーを作成した AWS リージョンを選択します。

   b. [New DB Snapshot Identifier (新しい DB スナップショットの識別子)] に、DB クラススナップショットのコピーの名前を入力します。

   c. AWS KMS key には、作成したカスタマーマネージドキーを選択します。



d. [スナップショットのコピー] を選択します。

7. スナップショットのコピーが使用可能になったら、それを選択します。
8. [Actions] (アクション) で、[Share snapshot] (スナップショットの共有) を選択します。
9. [スナップショットのアクセス許可] ページで、次の操作を行います。

a. スナップショットのコピーを共有する [AWS アカウント ID] を入力し、[追加] を選択します。

b. [Save] を選択します。

スナップショットが共有されます。

ターゲットアカウントに共有したスナップショットをコピーします。

これで、ターゲット AWS アカウント で共有スナップショットをコピーできます。

共有したスナップショットをコピーするには

1. ターゲット AWS アカウント から AWS Management Console にサインインします。
2. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
3. ナビゲーションペインで、[Snapshots] を選択します。
4. [自分と共有] タブを選択します。
5. 共有スナップショットを選択します。
6. [アクション] で、[スナップショットをコピー] を選択します。
7. 前の手順のようにスナップショットをコピーするための設定を選択しますが、ターゲットアカウントに属する AWS KMS key を使用します。

   [スナップショットのコピー] を選択します。

スナップショット共有の停止

DB クラススナップショットの共有を停止するには、ターゲット AWS アカウント からアクセス許可を削除します。

コンソール

AWS アカウント との手動 DB クラススナップショットの共有を停止するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Snapshots] を選択します。

3. 共有を停止する手動スナップショットを選択します。
4. [Actions] (アクション) を選択してから、[Share Snapshot] (スナップショットの共有) を選択します。
5. AWS アカウント のアクセス許可を削除するには、アクセス権限が付与されたアカウントのリストからそのアカウントの AWS アカウント識別子を選択し、[削除] を選択します。
6. [保存] を選択して変更を保存します。

CLI

リストから AWS アカウント の識別子を削除するには、`--values-to-remove` のパラメータを使用します。

Example スナップショット共有を停止する

次の例は、AWS アカウント ID 444455556666 がスナップショットを復元できないようにします。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier manual-cluster-snapshot1 \
--attribute-name restore \
--values-to-remove 444455556666
```

Windows の場合:

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier manual-cluster-snapshot1 ^
--attribute-name restore ^
--values-to-remove 444455556666
```

RDS API

AWS アカウント でのアクセス許可の共有を削除するには、`AttributeName` を `restore` に設定し、かつ `ValuesToRemove` パラメータを含めて、`ModifyDBClusterSnapshotAttribute` オペレーションを使用します。手動 スナップショットをプライベートとしてマークするには、`all` 属性の値リストから値 `restore` を削除します。

Amazon S3 への DB クラスターデータのエクスポート

ライブの Amazon Aurora DB クラスターから Amazon S3 バケットにデータをエクスポートできます。エクスポートプロセスはバックグラウンドで実行されるため、アクティブな DB クラスターのパフォーマンスには影響しません。

デフォルトでは、DB クラスター内のすべてのデータがエクスポートされます。ただし、特定のデータベース、スキーマ、またはテーブルのセットをエクスポートすることもできます。

Amazon Aurora は DB クラスターをクローンして、クローンからデータを抽出し、そのデータを Amazon S3 バケットに保存します。データは Apache Parquet 形式で一貫して圧縮され、保存されます。個々の Parquet ファイルのサイズは通常、1 ~ 10 MB です。

Aurora MySQL バージョン 2 とバージョン 3 のスナップショットデータをエクスポートすることで得られる高速パフォーマンスは、DB クラスターデータのエクスポートには当てはまりません。詳細については、「Amazon S3 への DB クラスタースナップショットデータのエクスポート」を参照してください。

エクスポートするのがすべてのデータなのか、部分的なデータなのかに関係なく、DB クラスター全体をエクスポートすると課金されます。詳細については、「Amazon Aurora 料金」ページを参照してください。

データをエクスポートすると、Amazon Athena や Amazon Redshift Spectrum などのツールを使用して、エクスポートしたデータを直接分析できます。Athena を使用して Parquet データを読み取る方法の詳細については、Amazon Athena ユーザーガイド の Parquet SerDe を参照してください。Redshift Spectrum を使用して Parquet データを読み取る方法の詳細については、Amazon Redshift Database デベロッパーガイドの「列指向データ形式からの COPY」を参照してください。

機能の可用性とサポートは、各データベースエンジンの特定のバージョンと AWS リージョン によって異なります。S3 への DB クラスターデータのエクスポートのバージョンとリージョンの可用性の詳細については、「Amazon S3 へのクラスターデータエクスポートでサポートされているリージョンと Aurora DB エンジン」を参照してください。

トピック

- DB クラスターエクスポートタスクのキャンセル
- Amazon S3 エクスポートタスクの障害メッセージ
- PostgreSQL のアクセス許可エラーのトラブルシューティング
- ファイル命名規則
- データ変換と保存形式

制限事項

DB クラスターデータの Amazon S3 へのエクスポートには、次の制限があります。

- 同じ DB クラスターに対して複数のエクスポートタスクを同時に実行することはできません。これは、フルエクスポートと部分エクスポートの両方に当てはまります。
- Aurora Serverless v1 DB クラスターは、S3 へのエクスポートをサポートしていません。
- Aurora MySQL および Aurora PostgreSQL は、プロビジョニングされたエンジンモードでのみ S3 へのエクスポートをサポートします。
- S3 へのエクスポートでは、コロン (:) を含む S3 プレフィックスをサポートしていません。
- S3 ファイルパスの次の文字は、エクスポート時にアンダースコア (_) に変換されます。

```
\ ` " (space)
```

- データベース、スキーマ、またはテーブルの名前に次の文字以外の文字が含まれている場合、部分的なエクスポートはサポートされません。ただし、DB クラスター全体をエクスポートすることはできます。
  - ラテン文字 (A-Z)
  - 数字 (0-9)
  - ドル記号 ($)
  - 下線 (_)
- データベーステーブルの列名では、一部の文字と空白文字の使用はサポートされていません。列名に次の文字が含まれるテーブルは、エクスポート時にスキップされます。

```
, ; { } ( ) \n \t = (space)
```

- 名前にスラッシュ (/) が含まれるテーブルは、エクスポート時にスキップされます。
- Aurora PostgreSQL の一時テーブルとログに記録されていないテーブルは、エクスポート中にスキップされます。

- データに BLOB や CLOB などの大きいオブジェクト (500 MB に近いか、それ以上) が含まれている場合、エクスポートは失敗します。
- テーブルに、2 GB に近いか、それ以上のサイズの大きな行が含まれている場合、そのテーブルはエクスポート時にスキップされます。
- 部分エクスポートの場合、`ExportOnly` リストの最大サイズは 200 KB です。
- エクスポートタスクごとに一意の名前を使用することを強くお勧めします。一意のタスク名を使用しない場合、次のエラーメッセージが表示されることがあります。

  ExportTaskAlreadyExistsFault: StartExportTask オペレーションを呼び出すときにエラー (ExportTaskAlreadyExists) が発生しました。ID *xxxxx* のエクスポートタスクは既に存在します。
- 一部のテーブルはスキップされる可能性があるため、エクスポート後にデータの行数とテーブル数を確認することをお勧めします。

DB クラスターデータのエクスポートの概要

次のプロセスを使用して、DB クラスターデータを Amazon S3 バケットにエクスポートします。詳細については、次のセクションを参照してください。

1. データをエクスポートする DB クラスターを特定します。
2. Amazon S3 バケットへのアクセスを設定します。

   バケットとは、Amazon S3 オブジェクトまたはファイルのコンテナです。バケットにアクセスするための情報を指定するには、次のステップに従います。

   a. DB クラスターデータのエクスポート先の S3 バケットを特定します。S3 バケットは DB クラスターと同じ AWS リージョンにある必要があります。詳細については、「エクスポート先の Amazon S3 バケットの特定」を参照してください。
   b. DB クラスターエクスポートタスクによる S3 バケットへのアクセス権を付与する AWS Identity and Access Management (IAM) ロールを作成します。詳細については、「IAM ロールを使用した Amazon S3 バケットへのアクセスの提供」を参照してください。
3. サーバー側の暗号化用の対称暗号化 AWS KMS key を作成します。KMS キーは、エクスポートデータを S3 に書き込むときに、AWS KMS サーバー側の暗号化を設定するために、クラスターエクスポートタスクによって使用されます。

   KMS キーポリシーには、`kms:CreateGrant` と `kms:DescribeKey` の両方のアクセス許可を含める必要があります。Amazon Aurora での KMS キーの使用方法の詳細については、「AWS KMS key 管理」を参照してください。

KMS キーポリシーに deny ステートメントがある場合は、必ず AWS サービスプリンシパル `export.rds.amazonaws.com` を明示的に除外してください。

AWSアカウント内で KMS キーを使用することも、クロスアカウント KMS キーを使用することもできます。詳細については、「クロスアカウント AWS KMS key を使用する」を参照してください。

4. コンソールまたは `start-export-task` CLI コマンドを使用して、DB クラスターを Amazon S3 にエクスポートします。詳細については、「Amazon S3 バケットへの DB クラスターデータのエクスポート」を参照してください。
5. Amazon S3 バケット内のエクスポートされたデータにアクセスするには、Amazon Simple Storage Service ユーザーガイドの「オブジェクトのアップロード、ダウンロード、管理」を参照してください。

Amazon S3 バケットへのアクセスを設定する

Amazon S3 バケットを特定したら、それにアクセスするアクセス許可を DB クラスターエクスポートタスクに与えます。

トピック

- エクスポート先の Amazon S3 バケットの特定
- IAM ロールを使用した Amazon S3 バケットへのアクセスの提供
- クロスアカウント Amazon S3 バケットを使用する

エクスポート先の Amazon S3 バケットの特定

DB クラスターデータをエクスポートする Amazon S3 バケットを特定します。既存の S3 バケットを使用するか、新しい S3 バケットを作成します。

> **Note**
>
> S3 バケットは DB クラスターと同じ AWS リージョンにある必要があります。

Amazon S3 バケットの操作の詳細については、Amazon Simple Storage Service ユーザーガイドで次のトピックを参照してください。

- S3 バケットのプロパティを表示する方法
- Amazon S3 バケットのデフォルト暗号化を有効にする方法
- S3 バケットを作成する方法

IAM ロールを使用した Amazon S3 バケットへのアクセスの提供

DB クラスターデータを Amazon S3 にエクスポートする前に、エクスポートタスクに対して Amazon S3 バケットへの書き込みアクセス許可を付与します。

このアクセス権限を付与するには、バケットへのアクセスを可能にする IAM ポリシーを作成し、次に IAM ロールを作成して、このロールにポリシーをアタッチします。後で IAM ロールを DB クラスターエクスポートタスクに割り当てることができます。

> **Important**
>
> AWS Management Console を使用して DB クラスターをエクスポートする予定がある場合は、DB クラスターをエクスポートするときに IAM ポリシーとロールを自動的に作成することもできます。手順については、Amazon S3 バケットへの DB クラスターデータのエクスポート を参照してください。

Amazon S3 へのアクセス権をタスクに付与するには

1. IAM ポリシーを作成します。このポリシーでバケットおよびオブジェクトへのアクセス許可を与えることにより、DB クラスターエクスポートタスクから Amazon S3 にアクセスできるようにします。

   Amazon Aurora から S3 バケットへのファイル転送を許可するために、以下の必須アクションをポリシーに含めます。

   - `s3:PutObject*`
   - `s3:GetObject*`
   - `s3:ListBucket`
   - `s3:DeleteObject*`
   - `s3:GetBucketLocation`

ポリシーには、S3 バケットとバケット内のオブジェクトを識別するために、以下のリソースを含めます。次のリソースのリストは、Amazon S3 にアクセスするための Amazon リソースネーム (ARN) 形式を示しています。

- `arn:aws:s3:::your-s3-bucket`
- `arn:aws:s3:::your-s3-bucket/*`

Amazon Aurora の IAM ポリシーの作成の詳細については、「IAM データベースアクセス用の IAM ポリシーの作成と使用」を参照してください。IAM ユーザーガイドの「チュートリアル: はじめてのカスタマー管理ポリシーの作成とアタッチ」も参照してください。

以下の AWS CLI コマンドでは、これらのオプションを指定して、ExportPolicy という名前の IAM ポリシーを作成します。このポリシーでは、your-s3-bucket という名前のバケットへのアクセス権が付与されます。

> **Note**
>
> ポリシーを作成したら、ポリシーの ARN を書き留めます。ポリシーを IAM ロールにアタッチする場合、後続のステップで ARN が必要です。

```
aws iam create-policy  --policy-name ExportPolicy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ExportPolicy",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject*",
                "s3:ListBucket",
                "s3:GetObject*",
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::your-s3-bucket",
                "arn:aws:s3:::your-s3-bucket/*"
            ]
```

```
        }
    ]
}'
```

2. IAM ロールを作成して、Aurora がこの IAM ロールがユーザーに代わって Amazon S3 バケットにアクセスすると見なすことができるようにします。詳細については、IAM ユーザーガイドの「IAM ユーザーにアクセス許可を委任するロールの作成」を参照してください。

   以下の例は、AWS CLI コマンドを使用して、`rds-s3-export-role` という名前のロールを作成する例を示しています。

```
aws iam create-role  --role-name rds-s3-export-role  --assume-role-policy-document
 '{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
           "Service": "export.rds.amazonaws.com"
         },
        "Action": "sts:AssumeRole"
      }
    ]
  }'
```

3. 作成した IAM ポリシーを、作成した IAM ロールにアタッチします。

   次の AWS CLI コマンドでは、先ほど作成したポリシーを `rds-s3-export-role` という名前のロールにアタッチします。*`your-policy-arn`* は、以前のステップで書き留めたポリシー ARN に置き換えます。

```
aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-
export-role
```

クロスアカウント Amazon S3 バケットを使用する

S3 バケットは AWS アカウント全体で使用できます。詳細については、「クロスアカウント Amazon S3 バケットを使用する」を参照してください。

Amazon S3 バケットへの DB クラスターデータのエクスポート

1 つの AWS アカウント につき、最大 5 つの DB クラスターエクスポートタスクを同時に実行できます。

> **Note**
>
> データベースのタイプとサイズによっては、DB クラスターデータのエクスポートに時間がかかることがあります。エクスポートタスクは、まず、データベース全体をクローニングおよびスケーリングしてから、Amazon S3 にデータを抽出します。このフェーズでのタスクの進捗状況は、[起動中] と表示されます。タスクが S3 へのデータのエクスポートに切り替わると、進捗状況は [進行中] と表示されます。
> エクスポートが完了するまでにかかる時間は、データベースに格納されているデータによって異なります。例えば、数値のプライマリキーまたはインデックス列が適切に配信されているテーブルは、最も速くエクスポートされます。テーブルにパーティション化に適した列が含まれていない場合や文字列ベースの列にインデックスが 1 つしかない場合、エクスポートには低速なシングルスレッド処理が使用されるため、処理に時間がかかります。

AWS Management Console、AWS CLI、または RDS API を使用して、DB クラスターデータを Amazon S3 にエクスポートできます。

Lambda 関数を使用して DB クラスターデータをエクスポートする場合は、Lambda 関数ポリシーに `kms:DescribeKey` アクションを追加します。詳細については、「AWS Lambda のアクセス許可」を参照してください。

コンソール

[Export to Amazon S3] (Amazon S3 へのエクスポート) コンソールオプションは、Amazon S3 にエクスポートできる DB クラスターに対してのみ表示されます。DB クラスターは、次の理由により、エクスポートに使用できない場合があります。

- DB エンジンが S3 エクスポートでサポートされていない。
- DB クラスターのバージョンが S3 エクスポートでサポートされていない。
- DB クラスターが作成された AWS リージョンで S3 エクスポートがサポートされていない。

DB クラスターデータをエクスポートするには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、データベースを選択します。
3. データをエクスポートする DB クラスターを選択します。
4. [アクション] で、[Amazon S3 にエクスポート] を選択します。

   [EAmazon S3 にエクスポート] ウィンドウが表示されます。
5. [エクスポート識別子] に、エクスポートタスクを識別する名前を入力します。この値は、S3 バケットで作成されるファイルの名前としても使用されます。
6. エクスポートするデータを選択します。

   - [All] (すべて) を選択すると、DB クラスター内のすべてのデータがエクスポートされます。
   - [Partial] (部分的) を選択すると、DB クラスターの特定部分がエクスポートされます。クラスターのどの部分をエクスポートするかを特定するには、[Identifiers] (識別子) に 1 つ以上のデータベース、スキーマ、またはテーブルをスペースで区切って入力します。

     次の形式を使用します。

     ```
     database[.schema][.table] database2[.schema2][.table2] ... databasen[.scheman]
     [.tablen]
     ```

     例:

     ```
     mydatabase mydatabase2.myschema1 mydatabase2.myschema2.mytable1
      mydatabase2.myschema2.mytable2
     ```
7. [S3 バケット] で、エクスポート先のバケットを選択します。

   エクスポートされたデータを S3 バケット内のフォルダパスに割り当てるには、[S3 プレフィックス] にオプションのパスを入力します。
8. IAM ロール の場合は、選択した S3 バケットへの書き込みアクセスを許可するロールを選択するか、新しいロールを作成します。

   - 「IAM ロールを使用した Amazon S3 バケットへのアクセスの提供」のステップに従ってロールを作成した場合は、そのロールを選択します。

- 選択した S3 バケットへの書き込みアクセス権を付与するロールを作成しなかった場合は、[Create a new role] (新しいロールの作成) を選択して、ロールを自動的に作成します。次に、[IAM role name] (IAM ロール名) にロールの名前を入力します。

9. [KMS key] (KMS キー) として、エクスポートされたデータの暗号化に使用するキーの ARN を入力します。
10. [Amazon S3 にエクスポート] を選択します。

AWS CLI

AWS CLI を使用して DB クラスターデータを Amazon S3 にエクスポートするには、以下の必須オプションを指定して start-export-task コマンドを使用します。

- `--export-task-identifier`
- `--source-arn` - DB クラスターの Amazon リソースネーム (ARN)。
- `--s3-bucket-name`
- `--iam-role-arn`
- `--kms-key-id`

以下の例では、エクスポートタスクに *my-cluster-export* という名前が付けられ、データを *my-export-bucket* という名前の S3 バケットにエクスポートします。

Example

Linux、macOS、Unix の場合:

```
aws rds start-export-task \
    --export-task-identifier my-cluster-export \
    --source-arn arn:aws:rds:us-west-2:123456789012:cluster:my-cluster \
    --s3-bucket-name my-export-bucket \
    --iam-role-arn iam-role \
    --kms-key-id my-key
```

Windows の場合:

```
aws rds start-export-task ^
    --export-task-identifier my-DB-cluster-export ^
```

```
    --source-arn arn:aws:rds:us-west-2:123456789012:cluster:my-cluster ^
    --s3-bucket-name my-export-bucket ^
    --iam-role-arn iam-role ^
    --kms-key-id my-key
```

サンプル出力を次に示します。

```
{
    "ExportTaskIdentifier": "my-cluster-export",
    "SourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:my-cluster",
    "S3Bucket": "my-export-bucket",
    "IamRoleArn": "arn:aws:iam:123456789012:role/ExportTest",
    "KmsKeyId": "my-key",
    "Status": "STARTING",
    "PercentProgress": 0,
    "TotalExtractedDataInGB": 0,
}
```

DB クラスターのエクスポート先である S3 バケット内のフォルダパスを指定するには、start-export-task コマンドに `--s3-prefix` オプションを含めます。

RDS API

Amazon RDS API を使用して DB クラスターデータを Amazon S3 にエクスポートするには、以下の必須パラメータを指定して StartExportTask オペレーションを使用します。

- `ExportTaskIdentifier`
- `SourceArn` - DB クラスターの ARN。
- `S3BucketName`
- `IamRoleArn`
- `KmsKeyId`

DB クラスターエクスポートタスクのモニタリング

DB クラスターのエクスポートは、AWS Management Console、AWS CLI、または RDS API を使用してモニタリングできます。

コンソール

DB クラスターのエクスポートをモニタリングするには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで [Exports in Amazon S3] (Amazon S3 にエクスポート) を選択します。

   DB クラスターのエクスポートは [Source type] (ソースタイプ) 列に表示されます。エクスポートのステータスは [Status] (ステータス) 列に表示されます。
3. 特定の DB クラスターのエクスポートに関する詳細情報を表示するには、エクスポートタスクを選択します。

AWS CLI

AWS CLI を使用して DB クラスターのエクスポートをモニタリングするには、describe-export-tasks コマンドを使用します。

次の例は、すべての DB クラスターのエクスポートに関する最新情報を表示する方法を示しています。

Example

```
aws rds describe-export-tasks

{
    "ExportTasks": [
        {
            "Status": "CANCELED",
            "TaskEndTime": "2022-11-01T17:36:46.961Z",
            "S3Prefix": "something",
            "S3Bucket": "examplebucket",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/K7MDENG/
bPxRfiCYEXAMPLEKEY",
            "ExportTaskIdentifier": "anewtest",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "SourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:parameter-groups-
test"
        },
```

```
{
            "Status": "COMPLETE",
            "TaskStartTime": "2022-10-31T20:58:06.998Z",
            "TaskEndTime": "2022-10-31T21:37:28.312Z",
            "WarningMessage": "{\"skippedTables\":[],\"skippedObjectives\":[],\"general
\":[{\"reason\":\"FAILED_TO_EXTRACT_TABLES_LIST_FOR_DATABASE\"}]}",
            "S3Prefix": "",
            "S3Bucket": "examplebucket1",
            "PercentProgress": 100,
            "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "thursday-events-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 263,
            "SourceArn": "arn:aws:rds:us-
west-2:123456789012:cluster:example-1-2019-10-31-06-44"
        },
        {
            "Status": "FAILED",
            "TaskEndTime": "2022-10-31T02:12:36.409Z",
            "FailureCause": "The S3 bucket examplebucket2 isn't located in the current
 AWS Region. Please, review your S3 bucket name and retry the export.",
            "S3Prefix": "",
            "S3Bucket": "examplebucket2",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "wednesday-afternoon-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "SourceArn": "arn:aws:rds:us-
west-2:123456789012:cluster:example-1-2019-10-30-06-45"
        }
    ]
}
```


特定のエクスポートタスクに関する情報を表示するには、--export-task-identifier オプションを describe-export-tasks コマンドに含めます。出力をフィルタリングするには、--Filters オプションを含めます。その他のオプションについては、 describe-export-tasks コマンドを参照してください。

RDS API

Amazon RDS API を使用して DB クラスターのエクスポートに関する情報を表示するには、DescribeExportTasks オペレーションを使用します。

エクスポートワークフローの完了を追跡したり、別のワークフローを開始したりするには、Amazon Simple Notification Service トピックをサブスクライブします。Amazon SNS の詳細については、「Amazon RDS イベント通知の操作」を参照してください。

DB クラスターエクスポートタスクのキャンセル

DB クラスターのエクスポートタスクをキャンセルするには、AWS Management Console、AWS CLI、または RDS API を使用できます。

> **Note**
>
> エクスポートタスクをキャンセルしても、Amazon S3 にエクスポート済みのデータは削除されません。コンソールを使用してデータを削除する方法については、「S3 バケットからオブジェクトを削除する方法」を参照してください。CLI を使用してデータを削除するには、delete-object コマンドを使用します。

コンソール

DB クラスターのエクスポートタスクをキャンセルするには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで [Exports in Amazon S3] (Amazon S3 にエクスポート) を選択します。

   DB クラスターのエクスポートは [Source type] (ソースタイプ) 列に表示されます。エクスポートのステータスは [Status] (ステータス) 列に表示されます。
3. キャンセルするエクスポートタスクを選択します。
4. [キャンセル] を選択します。
5. 確認ページで [エクスポートタスクをキャンセル] を選択します。

AWS CLI

AWS CLI を使用してエクスポートタスクをキャンセルするには、cancel-export-task コマンドを使用します。このコマンドには、--export-task-identifier オプションが必要です。

Example

```
aws rds cancel-export-task --export-task-identifier my-export
{
    "Status": "CANCELING",
    "S3Prefix": "",
    "S3Bucket": "examplebucket",
    "PercentProgress": 0,
    "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "ExportTaskIdentifier": "my-export",
    "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
    "TotalExtractedDataInGB": 0,
    "SourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:export-example-1"
}
```

RDS API

Amazon RDS API を使用してエクスポートタスクをキャンセルするには、ExportTaskIdentifier パラメータを指定して CancelExportTask オペレーションを使用します。

Amazon S3 エクスポートタスクの障害メッセージ

次の表では、Amazon S3 エクスポートタスクに障害が発生したときに返されるメッセージについて説明します。

障害メッセージ	説明
ソース DB クラスターの検索またはアクセスに失敗しました: [クラスター名]	ソース DB クラスターはクローニングできません。
不明な内部エラーが発生しました。	不明なエラー、例外、または障害により、タスクが失敗しました。
エクスポートタスクのメタデータを S3 バケット 「バケット名」 に書き込む	不明なエラー、例外、または障害により、タスクが失敗しました。

障害メッセージ	説明
際に、不明な内部エラーが発生しました。	
RDS エクスポートは、IAM ロール「ロール ARN」を引き受けることができないため、エクスポートタスクのメタデータを書き込めませんでした。	エクスポートタスクは IAM ロールを引き受け、S3 バケットへのメタデータの書き込みが許可されているかどうかを検証します。タスクが IAM ロールを引き受けられない場合は失敗します。
RDS エクスポートで、KMS キー「キー ID」 を持つ IAM ロール 「ロール ARN」 を使用した S3 バケット 「バケット名」へのエクスポートタスクのメタデータの書き込みに失敗しました。エラーコード:「エラーコード」	1 つ以上のアクセス許可が不足しているため、エクスポートタスクが S3 バケットにアクセスできません。この障害メッセージは、次のいずれかのエラーコードを受信したときにレイズされます。 • エラーコードAccessDenied のAWSSecurityTokenServiceException • エラーコードNoSuchBucket 、AccessDenied 、KMS.KMSInvalidStateException 、403 Forbidden 、またはKMS.DisabledException のAmazonS3Exception これらのエラーコードは、IAM ロール、S3 バケット、または KMS キーの設定が間違っていることを示しています。
IAM ロール [ロール ARN]には、S3 バケット [バケット名] で [S3 アクション] を呼び出す権限がありません。許可を確認してエクスポートを再試行してください。	IAM ポリシーの設定が間違っています。S3 バケットに対する特定の S3 アクションのためのアクセス許可がないため、エクスポートタスクが失敗します。
KMS キーチェックに失敗しました。KMS キーの認証情報をチェックして、再試行してください。	KMS キーの認証情報のチェックに失敗しました。

障害メッセージ	説明
S3 の認証情報のチェックに失敗しました。S3 バケットと IAM ポリシーに対するアクセス許可をチェックします。	S3の認証情報のチェックに失敗しました。
S3 バケット「バケット名」が無効です。どちらかが現在の AWS リージョン リージョンにないか、存在しません。S3 バケット名を確認して、エクスポートを再試行してください。	S3 バケットが無効です。
S3 バケット [バケット名] が現在の AWS リージョン リージョンにありません。S3 バケット名を確認して、エクスポートを再試行してください。	S3 バケットが間違った AWS リージョン にあります。

PostgreSQL のアクセス許可エラーのトラブルシューティング

PostgreSQL データベースを Amazon S3 にエクスポートするときに、特定のテーブルがスキップされたことを示す PERMISSIONS_DO_NOT_EXIST エラーが表示される場合があります。このエラーは、通常、DB クラスターの作成時に指定したスーパーユーザーに、これらのテーブルにアクセスするアクセス許可がない場合に発生します。

このエラーを修正するには、次のコマンドを実行します。

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA schema_name TO superuser_name
```

スーパーユーザー権限の詳細については、「マスターユーザーアカウント権限」を参照してください。

ファイル命名規則

特定のテーブルのエクスポートされたデータは、*base_prefix*/*files* の形式で保存されます。そのベースプレフィックスは次のとおりです。

```
export_identifier/database_name/schema_name.table_name/
```

例:

```
export-1234567890123-459/rdststcluster/mycluster.DataInsert_7ADB5D19965123A2/
```

出力ファイルは次の命名規則を使用します。*partition_index* は英数字です。

```
partition_index/part-00000-random_uuid.format-based_extension
```

例:

```
1/part-00000-c5a881bb-58ff-4ee6-1111-b41ecff340a3-c000.gz.parquet
a/part-00000-d7a881cc-88cc-5ab7-2222-c41ecab340a4-c000.gz.parquet
```

ファイルの命名規則は変更されることがあります。したがって、ターゲットテーブルを読み込む場合は、テーブルのベースプレフィックス内のすべてを読み込むことをお勧めします。

データ変換と保存形式

DB クラスターを Amazon S3 バケットにエクスポートすると、Amazon Aurora はデータを Parquet 形式に変換してエクスポートし、保存します。詳細については、「Amazon S3 バケットにエクスポートする際のデータ変換」を参照してください。

Amazon S3 への DB クラスタースナップショットデータのエクスポート

DB クラスタースナップショットデータを Amazon S3 バケットにエクスポートできます。エクスポートプロセスはバックグラウンドで実行されるため、アクティブな DB クラスターのパフォーマンスには影響しません。

DB クラスタースナップショットをエクスポートすると、Amazon Aurora はスナップショットからデータを抽出して Amazon S3 バケットに保存します。手動スナップショットと自動システムスナップショットをエクスポートできます。デフォルトでは、スナップショット内のすべてのデータがエクスポートされます。ただし、特定のデータベース、スキーマ、またはテーブルのセットをエクスポートすることもできます。

データは Apache Parquet 形式で一貫して圧縮され、保存されます。個々の Parquet ファイルのサイズは通常、1 ~ 10 MB です。

データをエクスポートすると、Amazon Athena や Amazon Redshift Spectrum などのツールを使用して、エクスポートしたデータを直接分析できます。Athena を使用して Parquet データを読み取る方法の詳細については、Amazon Athena ユーザーガイド の Parquet SerDe を参照してください。Redshift Spectrum を使用して Parquet データを読み取る方法の詳細については、Amazon Redshift Database デベロッパーガイドの「列指向データ形式からの COPY」を参照してください。

機能の可用性とサポートは、各データベースエンジンの特定のバージョンと AWS リージョン によって異なります。S3 への DB クラスタースナップショットデータのエクスポートのバージョンとリージョンの可用性の詳細については、「Amazon S3 へのスナップショットデータエクスポートでサポートされているリージョンと Aurora DB エンジン」を参照してください。

トピック

- Amazon S3 エクスポートタスクの障害メッセージ
- PostgreSQL のアクセス許可エラーのトラブルシューティング
- ファイル命名規則
- Amazon S3 バケットにエクスポートする際のデータ変換

制限事項

DB スナップショットデータの Amazon S3 へのエクスポートには、次の制限があります。

- 同じ DB クラススナップショットに対して複数のエクスポートタスクを同時に実行することはできません。これは、フルエクスポートと部分エクスポートの両方に当てはまります。
- Aurora Serverless v1 DB クラスターから S3 にスナップショットデータをエクスポートできません。
- S3 へのエクスポートでは、コロン (:) を含む S3 プレフィックスをサポートしていません。
- S3 ファイルパスの次の文字は、エクスポート時にアンダースコア (_) に変換されます。

```
\ ` " (space)
```

- データベース、スキーマ、またはテーブルの名前に次の文字以外の文字が含まれている場合、部分的なエクスポートはサポートされません。ただし、DB スナップショット全体をエクスポートすることはできます。
  - ラテン文字 (A-Z)
  - 数字 (0-9)
  - ドル記号 ($)
  - 下線 (_)
- データベーステーブルの列名では、一部の文字と空白文字の使用はサポートされていません。列名に次の文字が含まれるテーブルは、エクスポート時にスキップされます。

```
, ; { } ( ) \n \t = (space)
```

- 名前にスラッシュ (/) が含まれるテーブルは、エクスポート時にスキップされます。
- Aurora PostgreSQL の一時テーブルとログに記録されていないテーブルは、エクスポート中にスキップされます。
- データに BLOB や CLOB などの大きいオブジェクト (500 MB に近いか、それ以上) が含まれている場合、エクスポートは失敗します。

- テーブルに、2 GB に近いか、それ以上のサイズの大きな行が含まれている場合、そのテーブルはエクスポート時にスキップされます。
- 部分エクスポートの場合、`ExportOnly` リストの最大サイズは 200 KB です。
- エクスポートタスクごとに一意の名前を使用することを強くお勧めします。一意のタスク名を使用しない場合、次のエラーメッセージが表示されることがあります。

  ExportTaskAlreadyExistsFault: StartExportTask オペレーションを呼び出すときにエラー (ExportTaskAlreadyExists) が発生しました。ID *xxxxx* のエクスポートタスクは既に存在します。
- データを S3 にエクスポートしている間はスナップショットを削除できますが、エクスポートタスクが完了するまで、そのスナップショットのストレージコストは引き続き課金されます。
- S3 からエクスポートしたスナップショットデータを新しい DB クラスターに復元することはできません。

スナップショットデータのエクスポートの概要

次のプロセスを使用して、DB スナップショットデータを Amazon S3 バケットにエクスポートします。詳細については、次のセクションを参照してください。

1. エクスポートするスナップショットを特定します。

   既存の自動スナップショットまたは手動スナップショットを使用するか、DB インスタンスの手動スナップショットを作成します。
2. Amazon S3 バケットへのアクセスを設定します。

   バケットとは、Amazon S3 オブジェクトまたはファイルのコンテナです。バケットにアクセスするための情報を指定するには、次のステップに従います。

   a. スナップショットのエクスポート先の S3 バケットを特定します。S3 バケットはスナップショット同じ AWS リージョンに存在する必要があります。詳細については、「エクスポート先の Amazon S3 バケットの特定」を参照してください。

   b. スナップショットエクスポートタスクに対して S3 バケットへのアクセスを許可する AWS Identity and Access Management (IAM) ロールを作成します。詳細については、「IAM ロールを使用した Amazon S3 バケットへのアクセスの提供」を参照してください。
3. サーバー側の暗号化用の対称暗号化 AWS KMS key を作成します。KMS キーは、エクスポートデータを S3 に書き込むときに AWS KMS サーバー側の暗号化を設定するために、スナップショットエクスポートタスクによって使用されます。

KMS キーポリシーには、`kms:CreateGrant` と `kms:DescribeKey` の両方のアクセス許可を含める必要があります。Amazon Aurora での KMS キーの使用方法の詳細については、「AWS KMS key 管理」を参照してください。

KMS キーポリシーに deny ステートメントがある場合は、必ず AWS サービスプリンシパル `export.rds.amazonaws.com` を明示的に除外してください。

AWSアカウント内で KMS キーを使用することも、クロスアカウント KMS キーを使用することもできます。詳細については、「クロスアカウント AWS KMS key を使用する」を参照してください。

4. コンソールまたは `start-export-task` CLI コマンドを使用して、スナップショットを Amazon S3 にエクスポートします。詳細については、「Amazon S3 バケットへのスナップショットのエクスポート」を参照してください。
5. Amazon S3 バケット内のエクスポートされたデータにアクセスするには、Amazon Simple Storage Service ユーザーガイドの「オブジェクトのアップロード、ダウンロード、管理」を参照してください。

Amazon S3 バケットへのアクセスを設定する

Amazon S3 バケットを特定したら、それにアクセスするアクセス許可をスナップショットに与えます。

トピック

- エクスポート先の Amazon S3 バケットの特定
- IAM ロールを使用した Amazon S3 バケットへのアクセスの提供
- クロスアカウント Amazon S3 バケットを使用する
- クロスアカウント AWS KMS key を使用する

エクスポート先の Amazon S3 バケットの特定

DB スナップショットをエクスポートする先の Amazon S3 バケットを特定します。既存の S3 バケットを使用するか、新しい S3 バケットを作成します。

> **Note**
>
> エクスポート先の S3 バケットは、スナップショットと同じ AWS リージョンに存在している必要があります。

Amazon S3 バケットの操作の詳細については、Amazon Simple Storage Service ユーザーガイドで次のトピックを参照してください。

- S3 バケットのプロパティを表示する方法
- Amazon S3 バケットのデフォルト暗号化を有効にする方法
- S3 バケットを作成する方法

IAM ロールを使用した Amazon S3 バケットへのアクセスの提供

DB スナップショットデータを Amazon S3 にエクスポートする前に、スナップショットエクスポートタスクに対して Amazon S3 バケットへの書き込みアクセス権限を付与します。

このアクセス権限を付与するには、バケットへのアクセスを可能にする IAM ポリシーを作成し、次に IAM ロールを作成して、このロールにポリシーをアタッチします。後から IAM ロールをスナップショットエクスポートタスクに割り当てることができます。

> **Important**
>
> AWS Management Console を使用してスナップショットをエクスポートしようとする場合は、スナップショットをエクスポートするときに IAM ポリシーとロールを自動的に作成するように選択できます。手順については、Amazon S3 バケットへのスナップショットのエクスポート を参照してください。

Amazon S3 へのアクセスを DB スナップショットタスクに許可するには

1. IAM ポリシーを作成します。このポリシーでバケットおよびオブジェクトへのアクセス許可を提供することにより、スナップショットエクスポートタスクから Amazon S3 にアクセスできるようにします。

   Amazon Aurora から S3 バケットへのファイル転送を許可するために、以下の必須アクションをポリシーに含めます。

- s3:PutObject*
- s3:GetObject*
- s3:ListBucket
- s3:DeleteObject*
- s3:GetBucketLocation

ポリシーには、S3 バケットとバケット内のオブジェクトを識別するために、以下のリソースを含めます。次のリソースのリストは、Amazon S3 にアクセスするための Amazon リソースネーム (ARN) 形式を示しています。

- arn:aws:s3:::*your-s3-bucket*
- arn:aws:s3:::*your-s3-bucket*/*

Amazon Aurora の IAM ポリシーの作成の詳細については、「IAM データベースアクセス用の IAM ポリシーの作成と使用」を参照してください。IAM ユーザーガイドの「チュートリアル: はじめてのカスタマー管理ポリシーの作成とアタッチ」も参照してください。

以下の AWS CLI コマンドでは、これらのオプションを指定して、ExportPolicy という名前の IAM ポリシーを作成します。このポリシーでは、your-s3-bucket という名前のバケットへのアクセス権が付与されます。

> **Note**
>
> ポリシーを作成したら、ポリシーの ARN を書き留めます。ポリシーを IAM ロールにアタッチする場合、後続のステップで ARN が必要です。

```
aws iam create-policy  --policy-name ExportPolicy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ExportPolicy",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject*",
                "s3:ListBucket",
```

```
                "s3:GetObject*",
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::your-s3-bucket",
                "arn:aws:s3:::your-s3-bucket/*"
            ]
        }
    ]
}'
```

2. IAM ロールを作成して、Aurora がこの IAM ロールがユーザーに代わって Amazon S3 バケットにアクセスすると見なすことができるようにします。詳細については、IAM ユーザーガイドの「IAM ユーザーにアクセス許可を委任するロールの作成」を参照してください。

   以下の例は、AWS CLI コマンドを使用して、rds-s3-export-role という名前のロールを作成する例を示しています。

```
aws iam create-role  --role-name rds-s3-export-role  --assume-role-policy-document
 '{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
           "Service": "export.rds.amazonaws.com"
         },
        "Action": "sts:AssumeRole"
      }
    ]
  }'
```

3. 作成した IAM ポリシーを、作成した IAM ロールにアタッチします。

   次の AWS CLI コマンドでは、先ほど作成したポリシーを rds-s3-export-role という名前のロールにアタッチします。*your-policy-arn* は、以前のステップで書き留めたポリシー ARN に置き換えます。

```
aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-
export-role
```

クロスアカウント Amazon S3 バケットを使用する

Amazon S3 バケットはAWSアカウント全体で使用できます。クロスアカウントバケットを使用するには、S3 エクスポートに使用している IAM ロールへのアクセスを許可するバケットポリシーを追加してください。詳細については、「例 2: クロスアカウントバケット権限を付与するバケット所有者」を参照してください。

- 次の例のように、バケットポリシーをバケットに添付します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:role/Admin"
            },
            "Action": [
                "s3:PutObject*",
                "s3:ListBucket",
                "s3:GetObject*",
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::mycrossaccountbucket",
                "arn:aws:s3:::mycrossaccountbucket/*"
            ]
        }
    ]
}
```

クロスアカウント AWS KMS key を使用する

AWS KMS keyAmazon S3 エクスポートの暗号化にクロスアカウントを使用できます。まずローカルアカウントにキーポリシーを追加し、次に外部アカウントに IAM ポリシーを追加してください。詳細については、「その他のアカウントのユーザーに KMS キーの使用を許可する」を参照してください。

クロスアカウント KMS キーを使用するには

1. ローカルアカウントにキーポリシーを追加します。

   次の例は、外部アカウント 444455556666 のExampleRoleとExampleUserに、ローカルアカウント 123456789012 のアクセス許可を付与します。

```
{
    "Sid": "Allow an external account to use this KMS key",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "arn:aws:iam::444455556666:role/ExampleRole",
            "arn:aws:iam::444455556666:user/ExampleUser"
        ]
    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:CreateGrant",
        "kms:DescribeKey",
        "kms:RetireGrant"
    ],
    "Resource": "*"
}
```

2. 外部アカウントに IAM ポリシーを追加する

   以下の IAM ポリシーの例では、プリンシパルがアカウント 123456789012 の KMS キーを暗号化オペレーションに使用することを許可します。アカウント 444455556666 のExampleRoleとExampleUserにこの許可を与えるには、そのアカウントにポリシーをアタッチします。

```
{
    "Sid": "Allow use of KMS key in account 123456789012",
    "Effect": "Allow",
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
```

```
        "kms:GenerateDataKey*",
        "kms:CreateGrant",
        "kms:DescribeKey",
        "kms:RetireGrant"
    ],
    "Resource": "arn:aws:kms:us-
west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
}
```

Amazon S3 バケットへのスナップショットのエクスポート

1 つの AWS アカウント につき、最大 5 つの DB スナップショットエクスポートタスクを同時に実行できます。

> **Note**
>
> データベースのタイプとサイズによっては、RDS スナップショットのエクスポートに時間がかかることがあります。エクスポートタスクでは、Amazon S3 にデータを抽出する前に、データベース全体を復元およびスケールします。このフェーズでのタスクの進捗状況は、[起動中] と表示されます。タスクが S3 へのデータのエクスポートに切り替わると、進捗状況は [進行中] と表示されます。
> エクスポートが完了するまでにかかる時間は、データベースに格納されているデータによって異なります。例えば、数値のプライマリキーまたはインデックス列が適切に配信されているテーブルは、最も速くエクスポートされます。パーティション化に適した列が含まれていないテーブルや、文字列ベースの列に 1 つのインデックスしかないテーブルは処理に時間がかかります。エクスポートに低速のシングルスレッドプロセスを使用するため、このような長いエクスポート時間となります。

AWS Management Console、AWS CLI、または RDS API を使用して DB スナップショットを Amazon S3 にエクスポートできます。

Lambda 関数を使用してスナップショットをエクスポートする場合は、Lambda 関数ポリシーに `kms:DescribeKey` アクションを追加します。詳細については、「AWS Lambda のアクセス許可」を参照してください。

コンソール

[Amazon S3 へのエクスポート] コンソールオプションは、Amazon S3 にエクスポートできるスナップショットに対してのみ表示されます。スナップショットは、次の理由により、エクスポートに使用できない場合があります。

- DB エンジンが S3 エクスポートでサポートされていない。
- DB インスタンスのバージョンが S3 エクスポートでサポートされていない。
- スナップショットを作成した AWS リージョンで S3 エクスポートがサポートされていない。

DB スナップショットをエクスポートするには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[スナップショット] を選択します。
3. タブから、エクスポートするスナップショットのタイプを選択します。
4. スナップショットのリストで、エクスポートするスナップショットを選択します。
5. [アクション] で、[Amazon S3 にエクスポート] を選択します。

   [EAmazon S3 にエクスポート] ウィンドウが表示されます。
6. [エクスポート識別子] に、エクスポートタスクを識別する名前を入力します。この値は、S3 バケットで作成されるファイルの名前としても使用されます。
7. エクスポートするデータを選択します。

   - [すべて] を選択すると、スナップショット内のすべてのデータがエクスポートされます。
   - [部分的] を選択すると、スナップショットの特定部分がエクスポートされます。スナップショットのどの部分をエクスポートするかを特定するには、[識別子] に 1 つ以上のデータベース、スキーマ、またはテーブルをスペースで区切って入力します。

     次の形式を使用します。

     ```
     database[.schema][.table] database2[.schema2][.table2] ... databasen[.schemaN]
     [.tablen]
     ```

     次に例を示します。

```
mydatabase mydatabase2.myschema1 mydatabase2.myschema2.mytable1
 mydatabase2.myschema2.mytable2
```

8. [S3 バケット] で、エクスポート先のバケットを選択します。

   エクスポートされたデータを S3 バケット内のフォルダパスに割り当てるには、[S3 プレフィックス] にオプションのパスを入力します。

9. IAM ロール の場合は、選択した S3 バケットへの書き込みアクセスを許可するロールを選択するか、新しいロールを作成します。

   - 「IAM ロールを使用した Amazon S3 バケットへのアクセスの提供」のステップに従ってロールを作成した場合は、そのロールを選択します。
   - 選択した S3 バケットへの書き込みアクセス権を付与するロールを作成しなかった場合は、[Create a new role] (新しいロールの作成) を選択して、ロールを自動的に作成します。次に、[IAM role name] (IAM ロール名) にロールの名前を入力します。

10. [AWS KMS key] で、エクスポートされたデータの暗号化に使用するキーの ARN を入力します。

11. [Amazon S3 にエクスポート] を選択します。

AWS CLI

AWS CLI を使用して DB スナップショットを Amazon S3 にエクスポートするには、以下の必須オプションを指定して start-export-task コマンドを使用します。

- `--export-task-identifier`
- `--source-arn`
- `--s3-bucket-name`
- `--iam-role-arn`
- `--kms-key-id`

以下の例では、スナップショットエクスポートタスクは *`my_snapshot_export`* と名前が付けられ、スナップショットを *`my_export_bucket`* という名前の S3 バケットにエクスポートします。

Example

Linux、macOS、Unix の場合:

```
aws rds start-export-task \
    --export-task-identifier my-snapshot-export \
    --source-arn arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name \
    --s3-bucket-name my-export-bucket \
    --iam-role-arn iam-role \
    --kms-key-id my-key
```

Windows の場合:

```
aws rds start-export-task ^
    --export-task-identifier my-snapshot-export ^
    --source-arn arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name ^
    --s3-bucket-name my-export-bucket ^
    --iam-role-arn iam-role ^
    --kms-key-id my-key
```

サンプル出力を次に示します。

```
{
    "Status": "STARTING",
    "IamRoleArn": "iam-role",
    "ExportTime": "2019-08-12T01:23:53.109Z",
    "S3Bucket": "my-export-bucket",
    "PercentProgress": 0,
    "KmsKeyId": "my-key",
    "ExportTaskIdentifier": "my-snapshot-export",
    "TotalExtractedDataInGB": 0,
    "TaskStartTime": "2019-11-13T19:46:00.173Z",
    "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name"
}
```

スナップショットのエクスポート先である S3 バケット内のフォルダパスを指定するには、start-export-task コマンドに `--s3-prefix` オプションを含めます。

RDS API

Amazon RDS API を使用して DB スナップショットを Amazon S3 にエクスポートするには、以下の必須パラメータを指定して StartExportTask オペレーションを使用します。

- `ExportTaskIdentifier`
- `SourceArn`

- `S3BucketName`
- `IamRoleArn`
- `KmsKeyId`

Aurora MySQL のエクスポートパフォーマンス

Aurora MySQL バージョン 2 とバージョン 3 の DB クラスタースナップショットでは、高度なエクスポートメカニズムを使用して、パフォーマンスを向上させ、エクスポート時間を短縮します。このメカニズムには、Aurora 共有ストレージアーキテクチャを活用するための複数のエクスポートスレッドや Aurora MySQL パラレルクエリなどの最適化が含まれています。この最適化は、データセットのサイズと構造に応じて適応的に適用されます。

高速エクスポートプロセスを使用するためにパラレルクエリを有効にする必要はありませんが、このプロセスにはパラレルクエリと同じ制限があります。また、日が `0` または年が `0000` である日付など、一部のデータ値はサポートされていません。詳細については、「Amazon Aurora MySQL のパラレルクエリの使用」を参照してください。

パフォーマンスの最適化を適用すると、Aurora MySQL バージョン 2 と 3 のエクスポート用のかなり大きな (200 GB まで) Parquet ファイルが表示されることもあります。

データ型や値の互換性がないなどの理由で、より高速なエクスポートプロセスを使用できない場合、Aurora はパラレルクエリなしで自動的にシングルスレッドエクスポートモードに切り替わります。使用するプロセスやエクスポートするデータの量に応じて、エクスポートのパフォーマンスは異なります。

スナップショットのエクスポートのモニタリング

DB スナップショットのエクスポートは、AWS Management Console、AWS CLI、または RDS API を使用してモニタリングできます。

コンソール

DB スナップショットのエクスポートをモニタリングするには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで [Exports in Amazon S3] (Amazon S3 にエクスポート) を選択します。

DB スナップショットのエクスポートは [Source type] (ソースタイプ) 列に表示されます。エクスポートのステータスは [Status] (ステータス) 列に表示されます。

3. 特定のスナップショットのエクスポートに関する詳細情報を表示するには、エクスポートタスクを選択します。

AWS CLI

AWS CLI を使用して DB スナップショットのエクスポートをモニタリングするには、describe-export-tasks コマンドを使用します。

次の例は、すべてのスナップショットのエクスポートに関する最新情報を表示する方法を示しています。

Example

```
aws rds describe-export-tasks

{
    "ExportTasks": [
        {
            "Status": "CANCELED",
            "TaskEndTime": "2019-11-01T17:36:46.961Z",
            "S3Prefix": "something",
            "ExportTime": "2019-10-24T20:23:48.364Z",
            "S3Bucket": "examplebucket",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/K7MDENG/
bPxRfiCYEXAMPLEKEY",
            "ExportTaskIdentifier": "anewtest",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "TaskStartTime": "2019-10-25T19:10:58.885Z",
            "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:parameter-
groups-test"
        },
{
            "Status": "COMPLETE",
            "TaskEndTime": "2019-10-31T21:37:28.312Z",
            "WarningMessage": "{\"skippedTables\":[],\"skippedObjectives\":[],\"general
\":[{\"reason\":\"FAILED_TO_EXTRACT_TABLES_LIST_FOR_DATABASE\"}]}",
            "S3Prefix": "",
```

```
            "ExportTime": "2019-10-31T06:44:53.452Z",
            "S3Bucket": "examplebucket1",
            "PercentProgress": 100,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "thursday-events-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 263,
            "TaskStartTime": "2019-10-31T20:58:06.998Z",
            "SourceArn":
 "arn:aws:rds:AWS_Region:123456789012:snapshot:rds:example-1-2019-10-31-06-44"
        },
        {
            "Status": "FAILED",
            "TaskEndTime": "2019-10-31T02:12:36.409Z",
            "FailureCause": "The S3 bucket my-exports isn't located in the current AWS
 Region. Please, review your S3 bucket name and retry the export.",
            "S3Prefix": "",
            "ExportTime": "2019-10-30T06:45:04.526Z",
            "S3Bucket": "examplebucket2",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "wednesday-afternoon-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "TaskStartTime": "2019-10-30T22:43:40.034Z",
            "SourceArn":
 "arn:aws:rds:AWS_Region:123456789012:snapshot:rds:example-1-2019-10-30-06-45"
        }
    ]
}
```

特定のスナップショットのエクスポートに関する情報を表示するには、`--export-task-identifier` コマンドに `describe-export-tasks` オプションを含めます。出力をフィルタリングするには、`--Filters` オプションを含めます。その他のオプションについては、describe-export-tasks コマンドを参照してください。

RDS API

Amazon RDS API を使用して DB スナップショットのエクスポートに関する情報を表示するには、DescribeExportTasks オペレーションを使用します。

エクスポートワークフローの完了を追跡したり、別のワークフローを開始したりするには、Amazon Simple Notification Service トピックをサブスクライブします。Amazon SNS の詳細については、「Amazon RDS イベント通知の操作」を参照してください。

スナップショットのエクスポートタスクのキャンセル

DB スナップショットのエクスポートタスクをキャンセルするには、AWS Management Console、AWS CLI、または RDS API を使用できます。

> **Note**
>
> スナップショットのエクスポートタスクをキャンセルしても、Amazon S3 にエクスポート済みのデータは削除されません。コンソールを使用してデータを削除する方法については、「S3 バケットからオブジェクトを削除する方法」を参照してください。CLI を使用してデータを削除するには、delete-object コマンドを使用します。

コンソール

スナップショットのエクスポートタスクをキャンセルするには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで [Exports in Amazon S3] (Amazon S3 にエクスポート) を選択します。

   DB スナップショットのエクスポートは [Source type] (ソースタイプ) 列に表示されます。エクスポートのステータスは [Status] (ステータス) 列に表示されます。
3. キャンセルするスナップショットのエクスポートタスクを選択します。
4. [キャンセル] を選択します。
5. 確認ページで [エクスポートタスクをキャンセル] を選択します。

AWS CLI

AWS CLI を使用してスナップショットのエクスポートタスクをキャンセルするには、cancel-export-task コマンドを使用します。このコマンドには、`--export-task-identifier` オプションが必要です。

Example

```
aws rds cancel-export-task --export-task-identifier my_export
{
    "Status": "CANCELING",
    "S3Prefix": "",
    "ExportTime": "2019-08-12T01:23:53.109Z",
    "S3Bucket": "examplebucket",
    "PercentProgress": 0,
    "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "ExportTaskIdentifier": "my_export",
    "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
    "TotalExtractedDataInGB": 0,
    "TaskStartTime": "2019-11-13T19:46:00.173Z",
    "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:export-example-1"
}
```

RDS API

Amazon RDS API を使用してスナップショットのエクスポートタスクをキャンセルするには、`ExportTaskIdentifier` パラメータを指定して CancelExportTask オペレーションを使用します。

Amazon S3 エクスポートタスクの障害メッセージ

次の表では、Amazon S3 エクスポートタスクに障害が発生したときに返されるメッセージについて説明します。

障害メッセージ	説明
不明な内部エラーが発生しました。	不明なエラー、例外、または障害により、タスクが失敗しました。
エクスポートタスクのメタデータを S3 バケット 「バケット名」 に書き込む際に、不明な内部エラーが発生しました。	不明なエラー、例外、または障害により、タスクが失敗しました。
RDS エクスポートは、IAM ロール 「ロール ARN」を引き受けることがで	エクスポートタスクは IAM ロールを引き受け、S3 バケットへのメタデータの書き込みが許可されているか

障害メッセージ	説明
きないため、エクスポートタスクのメタデータを書き込めませんでした。	どうかを検証します。タスクが IAM ロールを引き受けられない場合は失敗します。
RDS エクスポートで、KMS キー「キー ID」 を持つ IAM ロール 「ロール ARN」 を使用した S3 バケット 「バケット名」へのエクスポートタスクのメタデータの書き込みに失敗しました。エラーコード:「エラーコード」	1 つ以上のアクセス許可が不足しているため、エクスポートタスクが S3 バケットにアクセスできません。この障害メッセージは、次のいずれかのエラーコードを受信したときにレイズされます。 • エラーコードAccessDenied のAWSSecurityTokenServiceException • エラーコードNoSuchBucket 、AccessDenied 、KMS.KMSInvalidStateException 、403 Forbidden 、またはKMS.DisabledException のAmazonS3Exception これらのエラーコードは、IAM ロール、S3 バケット、または KMS キーの設定が間違っていることを示しています。
IAM ロール [ロール ARN]には、S3 バケット [バケット名] で [S3 アクション] を呼び出す権限がありません。許可を確認してエクスポートを再試行してください。	IAM ポリシーの設定が間違っています。S3 バケットに対する特定の S3 アクションのためのアクセス許可がないため、エクスポートタスクが失敗します。
KMS キーチェックに失敗しました。KMS キーの認証情報をチェックして、再試行してください。	KMS キーの認証情報のチェックに失敗しました。
S3 の認証情報のチェックに失敗しました。S3 バケットと IAM ポリシーに対するアクセス許可をチェックします。	S3 の認証情報のチェックに失敗しました。

障害メッセージ	説明
S3 バケット「バケット名」が無効です。どちらかが現在の AWS リージョンリージョンにないか、存在しません。S3 バケット名を確認して、エクスポートを再試行してください。	S3 バケットが無効です。
S3 バケット [バケット名] が現在の AWS リージョン リージョンにありません。S3 バケット名を確認して、エクスポートを再試行してください。	S3 バケットが間違った AWS リージョン にあります。

PostgreSQL のアクセス許可エラーのトラブルシューティング

PostgreSQL データベースを Amazon S3 にエクスポートするときに、特定のテーブルがスキップされたことを示す PERMISSIONS_DO_NOT_EXIST エラーが表示される場合があります。このエラーは、通常、DB インスタンスの作成時に指定したスーパーユーザーに、これらのテーブルに対するアクセス許可がない場合に発生します。

このエラーを修正するには、次のコマンドを実行します。

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA schema_name TO superuser_name
```

スーパーユーザー権限の詳細については、「マスターユーザーアカウント権限」を参照してください。

ファイル命名規則

特定のテーブルのエクスポートされたデータは、*base_prefix*/*files* の形式で保存されます。そのベースプレフィックスは次のとおりです。

```
export_identifier/database_name/schema_name.table_name/
```

例:

```
export-1234567890123-459/rdststdb/rdststdb.DataInsert_7ADB5D19965123A2/
```

ファイルを名付ける方法には、次の 2 つの規則があります。

- 現在の規則:

```
batch_index/part-partition_index-random_uuid.format-based_extension
```

バッチインデックスは、テーブルから読み込まれたデータのバッチを表すシーケンス番号です。テーブルを小さなチャンクに分割し、並列でエクスポートできない場合は、複数のバッチインデックスになります。テーブルが複数のテーブルにパーティション化されている場合にも同じことが起こります。メインテーブルのテーブルパーティションごとに 1 つずつ、複数のバッチインデックスがあります。

テーブルを小さなチャンクに分割し、並列で読み取ることができる場合は、バッチインデックス 1 フォルダのみになります。

バッチインデックスフォルダ内には、テーブルのデータを含む 1 つまたは複数の Parquet ファイルがあります。Parquet ファイル名のプレフィックスは part-*partition_index* です。テーブルがパーティション化されている場合、パーティションインデックス 00000 で始まる複数のファイルになります。

パーティションインデックスシーケンスにギャップが生じる可能性があります。これは、各パーティションがテーブル内の範囲クエリから取得されるためです。そのパーティションの範囲内にデータがない場合、そのシーケンス番号はスキップされます。

例えば、id 列がテーブルのプライマリキーで、その最小値と最大値が 100 と 1000 であるとします。このテーブルを 9 つのパーティションで エクスポートしようとすると、次のような並列クエリで読み取られます。

```
SELECT * FROM table WHERE id <= 100 AND id < 200
SELECT * FROM table WHERE id <= 200 AND id < 300
```

これにより、part-00000-*random_uuid*.gz.parquet から part-00008-*random_uuid*.gz.parquet までの 9 つのファイルが生成されます。ただし、200 と 350 の間に ID を持つ行がない場合、完了したパーティションの 1 つが空になり、それに対するファイルも作成されません。前の例では、part-00001-*random_uuid*.gz.parquet は作成されません。

- 以前の規則:

```
part-partition_index-random_uuid.format-based_extension
```

これは現在の規則と同じですが、例えば、*batch_index* プレフィックスは除きます。

```
part-00000-c5a881bb-58ff-4ee6-1111-b41ecff340a3-c000.gz.parquet
part-00001-d7a881cc-88cc-5ab7-2222-c41ecab340a4-c000.gz.parquet
part-00002-f5a991ab-59aa-7fa6-3333-d41eccd340a7-c000.gz.parquet
```

ファイルの命名規則は変更されることがあります。したがって、ターゲットテーブルを読み込む場合は、テーブルのベースプレフィックス内のすべてを読み込むことをお勧めします。

Amazon S3 バケットにエクスポートする際のデータ変換

DB スナップショットを Amazon S3 バケットにエクスポートすると、Amazon Aurora はデータを Parquet 形式に変換してエクスポートし、保存します。Parquet の詳細については、Apache Parquet のウェブサイトを参照してください。

Parquet は、すべてのデータを次のプリミティブ型の 1 つとして格納します。

- BOOLEAN
- INT32
- INT64
- INT96
- FLOAT
- DOUBLE
- BYTE_ARRAY - バイナリとも呼ばれる可変長のバイト配列
- FIXED_LEN_BYTE_ARRAY - 値が一定のサイズを持つ場合に使用される固定長のバイト配列

Parquet のデータ型はほとんど存在せず、この形式の読み書きに伴う複雑さが軽減されるようになっています。Parquet は、プリミティブ型を拡張するための論理的な型を提供します。論理的な型は、`LogicalType` メタデータフィールドにデータを持つ注釈として実装されます。論理的な型の注釈は、プリミティブ型の解釈方法を示します。

`STRING` 論理的な型が `BYTE_ARRAY` 型に注釈を付けた場合は、このバイト配列を UTF-8 でエンコードされた文字列として解釈する必要があることを示します。エクスポートタスクが完了する

と、Amazon Aurora は文字列変換が発生したかどうかを通知します。エクスポートされた基になるデータは、常に送信元データと同じです。ただし、UTF-8 のエンコーディングに伴う差異により、Athena などのツールで読み取ると、一部の文字はソースと異なるように表示される場合があります。

詳細については、Parquet ドキュメントの「Parquet Logical Type Definitions」を参照してください。

トピック

- MySQL データ型の Parquet へのマッピング
- PostgreSQL データ型の Parquet へのマッピング

MySQL データ型の Parquet へのマッピング

次の表は、データが変換されて Amazon S3 にエクスポートされる際の MySQL データ型から Parquet データ型へのマッピングを示しています。

出典データ型	Parquet プリミティブ型	論理的な型の注釈	変換に関するメモ
数値データ型			
BIGINT	INT64		
BIGINT UNSIGNED	FIXED_LEN_BYTE_ARRAY(9)	DECIMAL(20,0)	Parquet は符号付き型のみをサポートしているため、マッピングは BIGINT_UNSIGNED 型を格納するために追加のバイト (8 プラス 1) を必要とします。
BIT	BYTE_ARRAY		
DECIMAL	INT32	DECIMAL(p,s)	出典値が 2^{31} 未満の場合は、INT32 として格納されます。

出典データ型	Parquet プリミティブ型	論理的な型の注釈	変換に関するメモ
	INT64	DECIMAL(p,s)	出典値が 2^{31} 以上で 2^{63} 未満の場合は、INT64 として格納されます。
	FIXED_LEN_BYTE_ARRAY(N)	DECIMAL(p,s)	出典値が 2^{63} 以上の場合は、FIXED_LEN_BYTE_ARRAY(N) として格納されます。
	BYTE_ARRAY	STRING	Parquet は、38 を超える 10 進精度をサポートしていません。10 進値は、BYTE_ARRAY 型の文字列に変換され、UTF8 としてエンコードされます。
DOUBLE	DOUBLE		
FLOAT	DOUBLE		
INT	INT32		
INT UNSIGNED	INT64		
MEDIUMINT	INT32		
MEDIUMINT UNSIGNED	INT64		
NUMERIC	INT32	DECIMAL(p,s)	出典値が 2^{31} 未満の場合は、INT32 として格納されます。

出典データ型	Parquet プリミティブ型	論理的な型の注釈	変換に関するメモ
	INT64	DECIMAL(p,s)	出典値が 2^{31} 以上で 2^{63} 未満の場合は、INT64 として格納されます。
	FIXED_LEN_ARRAY(N)	DECIMAL(p,s)	出典値が 2^{63} 以上の場合は、FIXED_LEN_BYTE_ARRAY(N) として格納されます。
	BYTE_ARRAY	STRING	Parquet は、38 を超える数値精度をサポートしていません。この数値は、BYTE_ARRAY 型の文字列に変換され、UTF8 としてエンコードされます。
SMALLINT	INT32		
SMALLINT UNSIGNED	INT32		
TINYINT	INT32		
TINYINT UNSIGNED	INT32		
文字列データ型			
BINARY	BYTE_ARRAY		
BLOB	BYTE_ARRAY		
CHAR	BYTE_ARRAY		

出典データ型	Parquet プリミティブ型	論理的な型の注釈	変換に関するメモ
ENUM	BYTE_ARRAY	STRING	
LINESTRING	BYTE_ARRAY		
LONGBLOB	BYTE_ARRAY		
LONGTEXT	BYTE_ARRAY	STRING	
MEDIUMBLOB	BYTE_ARRAY		
MEDIUMTEXT	BYTE_ARRAY	STRING	
MULTILINESTRING	BYTE_ARRAY		
SET	BYTE_ARRAY	STRING	
TEXT	BYTE_ARRAY	STRING	
TINYBLOB	BYTE_ARRAY		
TINYTEXT	BYTE_ARRAY	STRING	
VARBINARY	BYTE_ARRAY		
VARCHAR	BYTE_ARRAY	STRING	
日付と時刻のデータ型			
DATE	BYTE_ARRAY	STRING	日付は BYTE_ARRAY 型の文字列に変換され、UTF8 としてエンコードされます。
DATETIME	INT64	TIMESTAMP_MICROS	

出典データ型	Parquet プリミティブ型	論理的な型の注釈	変換に関するメモ
TIME	BYTE_ARRAY	STRING	TIME 型は BYTE_ARRAY の文字列に変換され、UTF8 としてエンコードされます。
TIMESTAMP	INT64	TIMESTAMP_MICROS	
YEAR	INT32		
ジオメトリデータ型			
GEOMETRY	BYTE_ARRAY		
GEOMETRYCOLLECTION	BYTE_ARRAY		
MULTIPOINT	BYTE_ARRAY		
MULTIPOLYGON	BYTE_ARRAY		
POINT	BYTE_ARRAY		
POLYGON	BYTE_ARRAY		
JSON データ型			
JSON	BYTE_ARRAY	STRING	

PostgreSQL データ型の Parquet へのマッピング

次の表は、データが変換されて Amazon S3 にエクスポートされる際の PostgreSQL データ型から Parquet データ型へのマッピングを示しています。

PostgreSQL のデータ型	Parquet プリミティブ型	論理的な型の注釈	マッピングに関するメモ
数値データ型			
BIGINT	INT64		
BIGSERIAL	INT64		
DECIMAL	BYTE_ARRAY	STRING	DECIMAL 型は BYTE_ARRAY 型の文字列に変換され、UTF8 としてエンコードされます。 この変換は、データ精度や、数値ではないデータ値 (NaN) に伴う複雑さを回避するためのものです。
DOUBLE PRECISION	DOUBLE		
INTEGER	INT32		
MONEY	BYTE_ARRAY	STRING	
REAL	FLOAT		
SERIAL	INT32		
SMALLINT	INT32	INT_16	
SMALLSERIAL	INT32	INT_16	
文字列および関連データ型			
ARRAY	BYTE_ARRAY	STRING	配列は文字列に変換され、BINARY

PostgreSQL のデータ型	Parquet プリミティブ型	論理的な型の注釈	マッピングに関するメモ
			(UTF8) としてエンコードされます。 この変換は、データ精度、数値ではないデータ値 (NaN)、および時間データ値に伴う複雑さを回避するためのものです。
BIT	BYTE_ARRAY	STRING	
BIT VARYING	BYTE_ARRAY	STRING	
BYTEA	BINARY		
CHAR	BYTE_ARRAY	STRING	
CHAR(N)	BYTE_ARRAY	STRING	
ENUM	BYTE_ARRAY	STRING	
NAME	BYTE_ARRAY	STRING	
TEXT	BYTE_ARRAY	STRING	
TEXT SEARCH	BYTE_ARRAY	STRING	
VARCHAR(N)	BYTE_ARRAY	STRING	
XML	BYTE_ARRAY	STRING	
日付と時刻のデータ型			
DATE	BYTE_ARRAY	STRING	
INTERVAL	BYTE_ARRAY	STRING	

PostgreSQL のデータ型	Parquet プリミティブ型	論理的な型の注釈	マッピングに関するメモ
TIME	BYTE_ARRAY	STRING	
TIME WITH TIME ZONE	BYTE_ARRAY	STRING	
TIMESTAMP	BYTE_ARRAY	STRING	
TIMESTAMP WITH TIME ZONE	BYTE_ARRAY	STRING	
ジオメトリデータ型			
BOX	BYTE_ARRAY	STRING	
CIRCLE	BYTE_ARRAY	STRING	
LINE	BYTE_ARRAY	STRING	
LINESEGMENT	BYTE_ARRAY	STRING	
PATH	BYTE_ARRAY	STRING	
POINT	BYTE_ARRAY	STRING	
POLYGON	BYTE_ARRAY	STRING	
JSON データ型			
JSON	BYTE_ARRAY	STRING	
JSONB	BYTE_ARRAY	STRING	
その他のデータ型			
BOOLEAN	BOOLEAN		
CIDR	BYTE_ARRAY	STRING	ネットワークデータ型

PostgreSQL のデータ型	Parquet プリミティブ型	論理的な型の注釈	マッピングに関するメモ
COMPOSITE	BYTE_ARRAY	STRING	
DOMAIN	BYTE_ARRAY	STRING	
INET	BYTE_ARRAY	STRING	ネットワークデータ型
MACADDR	BYTE_ARRAY	STRING	
OBJECT IDENTIFIER	該当なし		
PG_LSN	BYTE_ARRAY	STRING	
RANGE	BYTE_ARRAY	STRING	
UUID	BYTE_ARRAY	STRING	

DB クラスターを指定の時点の状態に復元する

DB クラスターを特定の時点に復元し、新しい DB クラスターを作成することができます。

DB クラスターを指定した時点に復元する際には、デフォルトの仮想プライベートクラウド (VPC) セキュリティグループを選択します。または、カスタム VPC セキュリティグループを DB クラスターに適用することも可能です。

復元された DB クラスターは、デフォルトの DB クラスターと DB パラメータグループに自動的に関連付けられます。ただし、カスタムパラメータグループに適用するには、復元中に指定します。

Amazon Aurora は、DB クラスターのログレコードを継続的に Amazon S3 にアップロードします。DB クラスターの復元可能な直近の時間を確認するには、AWS CLI の describe-db-clusters コマンドを使用し、DB クラスターの [`LatestRestorableTime`] フィールドに返される値を確認します。

バックアップ保持期間の任意の時点に復元できます。DB クラスターの復元可能な最も早い時間を確認するには、AWS CLI describe-db-clusters コマンドを使用して、DB クラスターの [`EarliestRestorableTime`] フィールドで返される値を確認します。

復元された DB クラスターのバックアップ保持期間は、ソース DB クラスターのバックアップ保持期間と同じです。

> **Note**
>
> このトピックの情報は Amazon Aurora に適用されます。Amazon RDS DB インスタンスの復元については、「DBinstance を指定した時点へ復元する」を参照してください。
> Aurora DB クラスターのバックアップと復元の詳細については、「Aurora DB クラスターのバックアップと復元の概要」を参照してください。
> Aurora MySQL の場合は、プロビジョニング済み DB クラスターを Aurora Serverless DB クラスターに復元できます。詳細については、「Aurora Serverless v1 DB クラスターの復元」を参照してください。
> また、AWS Backup を使用して、Amazon Aurora DB クラスターのバックアップを管理することもできます。DB クラスターが AWS Backup のバックアッププランに関連付けられている場合、そのバックアッププランはポイントインタイムリカバリに使用されます。詳細については、AWS Backup を使用して DB クラスターを指定の時点の状態に復元する を参照してください。

RDS 延長サポートバージョンを使用した Aurora DB クラスターまたはグローバルクラスターの復元については、「Amazon RDS 延長サポートでの Aurora DB クラスターまたはグローバルクラスターの復元」を参照してください。

AWS Management Console、AWS CLI、または RDS API を使用して、DB クラスターを特定の時点に復元できます。

コンソール

DB クラスターを特定の時点に復元するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、「自動バックアップ」を選択します。

   [Current Region] (現在のリージョン) タブに自動バックアップが表示されます。

3. 復元する DB クラスターを選択します。
4. 「アクション」 で、「特定時点への復元」 を選択します。

   [特定時点への復元] ウィンドウが表示されます。
5. 「Latest restorable time」 を選択してできるだけ最新の時点に復元するか、「カスタム」 を選択して時刻を選択します。

   [Custom] (カスタム) を選択した場合は、クラスターを復元する日時を入力します。

   > **Note**
   >
   > 時刻は、協定世界時 (UTC) からのオフセットとしてローカルタイムゾーンで表示されます。例えば、UTC-5 は東部スタンダード時/中部夏時間です。

6. [DB クラスター識別子] として、ターゲットが復元された DB クラスターの名前を入力します。名前は一意である必要があります。

7. 必要に応じて、DB インスタンスクラスや DB ストレージ設定などの他のオプションを選択します。

   各設定の詳細については、「Aurora DB クラスターの設定」を参照してください。

8. [Restore to point in time] (特定時点への復元) を選択します。

AWS CLI

DB クラスターを指定した時点に復元するには、AWS CLI コマンドrestore-db-cluster-to-point-in-time を使用して、新しい DB クラスターを作成します。

他の設定を指定できます。各設定の詳細については、「Aurora DB クラスターの設定」を参照してください。

このオペレーションでは、リソースのタグ付けがサポートされています。--tags オプションを使用すると、ソース DB クラスターのタグは無視され、指定されたタグが使用されます。それ以外の場合は、ソースクラスターの最新のタグが使用されます。

Example

Linux、macOS、Unix の場合:

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier mysourcedbcluster \
    --db-cluster-identifier mytargetdbcluster \
    --restore-to-time 2017-10-14T23:45:00.000Z
```

Windows の場合:

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-identifier mysourcedbcluster ^
    --db-cluster-identifier mytargetdbcluster ^
    --restore-to-time 2017-10-14T23:45:00.000Z
```

> **Important**
>
> コンソールを使用して DB クラスターを特定の時点に復元する場合、Amazon RDS は自動的に使用する DB クラスターのプライマリインスタンス (ライター) を作成します。AWS CLI を使用して DB クラスターを特定の時点に復元する場合は、使用する DB クラスターのプラ

イマリインスタンスを明示的に作成する必要があります。プライマリインスタンスは、DB クラスターで作成される初期の DB インスタンスです。
DB クラスターのプライマリインスタンスを作成するには、AWS CLI コマンド create-db-instance を使用します。`--db-cluster-identifier` オプション値として DB クラスターの名前を含めます。

RDS API

DB クラスターを特定の時間に復元するには、以下のパラメータを指定して Amazon RDS API の `RestoreDBClusterToPointInTime` オペレーションを呼び出します。

- `SourceDBClusterIdentifier`
- `DBClusterIdentifier`
- `RestoreToTime`

Important

コンソールを使用して DB クラスターを特定の時点に復元する場合、Amazon RDS は自動的に使用する DB クラスターのプライマリインスタンス (ライター) を作成します。RDS API を使用して DB クラスターを特定の時点に復元する場合は、DB クラスターのプライマリインスタンスを明示的に作成する必要があります。プライマリインスタンスは、DB クラスターで作成される初期の DB インスタンスです。
DB クラスターのプライマリインスタンスを作成するには、RDS API の CreateDBInstance オペレーションを呼び出します。`DBClusterIdentifier` パラメータの値として DB クラスターの名前を含めます。

保持されている自動バックアップから指定した時点に DB クラスターを復元する

バックアップがソースクラスターの保持期間内であれば、ソース DB クラスターを削除した後、保持されている自動バックアップから DB クラスターを復元できます。このプロセスは、自動バックアップから DB クラスターを復元するのと似ています。

> **Note**
>
> Aurora Serverless v1 クラスターの自動バックアップは保持されないため、この手順を使用して Aurora Serverless v1 DB クラスターを復元することはできません。

コンソール

DB クラスターを特定の時点に復元するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、「自動バックアップ」を選択します。
3. [保持] タブを選択します。

4. 復元する DB クラスターを選択します。
5. 「アクション」 で、「特定時点への復元」 を選択します。

   [特定時点への復元] ウィンドウが表示されます。
6. 「Latest restorable time」 を選択してできるだけ最新の時点に復元するか、「カスタム」 を選択して時刻を選択します。

   [Custom] (カスタム) を選択した場合は、クラスターを復元する日時を入力します。

   > **Note**
   >
   > 時刻は、協定世界時 (UTC) からのオフセットとしてローカルタイムゾーンで表示されます。例えば、UTC-5 は東部スタンダード時/中部夏時間です。
7. [DB クラスター識別子] として、ターゲットが復元された DB クラスターの名前を入力します。名前は一意である必要があります。
8. 必要に応じて、DB インスタンスクラスなどの他のオプションを選択します。

各設定の詳細については、「Aurora DB クラスターの設定」を参照してください。

9. [Restore to point in time] (特定時点への復元) を選択します。

AWS CLI

DB クラスターを指定した時点に復元するには、AWS CLI コマンドrestore-db-cluster-to-point-in-time を使用して、新しい DB クラスターを作成します。

他の設定を指定できます。各設定の詳細については、「Aurora DB クラスターの設定」を参照してください。

このオペレーションでは、リソースのタグ付けがサポートされています。--tags オプションを使用すると、ソース DB クラスターのタグは無視され、指定されたタグが使用されます。それ以外の場合は、ソースクラスターの最新のタグが使用されます。

Example

Linux、macOS、Unix の場合:

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-resource-id cluster-123ABCEXAMPLE \
    --db-cluster-identifier mytargetdbcluster \
    --restore-to-time 2017-10-14T23:45:00.000Z
```

Windows の場合:

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-resource-id cluster-123ABCEXAMPLE ^
    --db-cluster-identifier mytargetdbcluster ^
    --restore-to-time 2017-10-14T23:45:00.000Z
```

Important

コンソールを使用して DB クラスターを特定の時点に復元する場合、Amazon RDS は自動的に使用する DB クラスターのプライマリインスタンス (ライター) を作成します。AWS CLI を使用して DB クラスターを特定の時点に復元する場合は、使用する DB クラスターのプライマリインスタンスを明示的に作成する必要があります。プライマリインスタンスは、DB クラスターで作成される初期の DB インスタンスです。

DB クラスターのプライマリインスタンスを作成するには、AWS CLI コマンド create-db-instance を使用します。--db-cluster-identifier オプション値として DB クラスターの名前を含めます。

RDS API

DB クラスターを特定の時間に復元するには、以下のパラメータを指定して Amazon RDS API の RestoreDBClusterToPointInTime オペレーションを呼び出します。

- SourceDbClusterResourceId
- DBClusterIdentifier
- RestoreToTime

Important

コンソールを使用して DB クラスターを特定の時点に復元する場合、Amazon RDS は自動的に使用する DB クラスターのプライマリインスタンス (ライター) を作成します。RDS API を使用して DB クラスターを特定の時点に復元する場合は、DB クラスターのプライマリインスタンスを明示的に作成する必要があります。プライマリインスタンスは、DB クラスターで作成される初期の DB インスタンスです。
DB クラスターのプライマリインスタンスを作成するには、RDS API の CreateDBInstance オペレーションを呼び出します。DBClusterIdentifier パラメータの値として DB クラスターの名前を含めます。

AWS Backup を使用して DB クラスターを指定の時点の状態に復元する

AWS Backup を使用して自動バックアップを管理し、指定した時刻に復元できます。そのためには、AWS Backup でバックアッププランを作成し、DB クラスターをリソースとして割り当てます。次に、バックアップルールで PITR の継続的バックアップを有効にします。バックアッププランとバックアップルールについて詳しくは、「AWS Backup デベロッパーガイド」を参照してください。

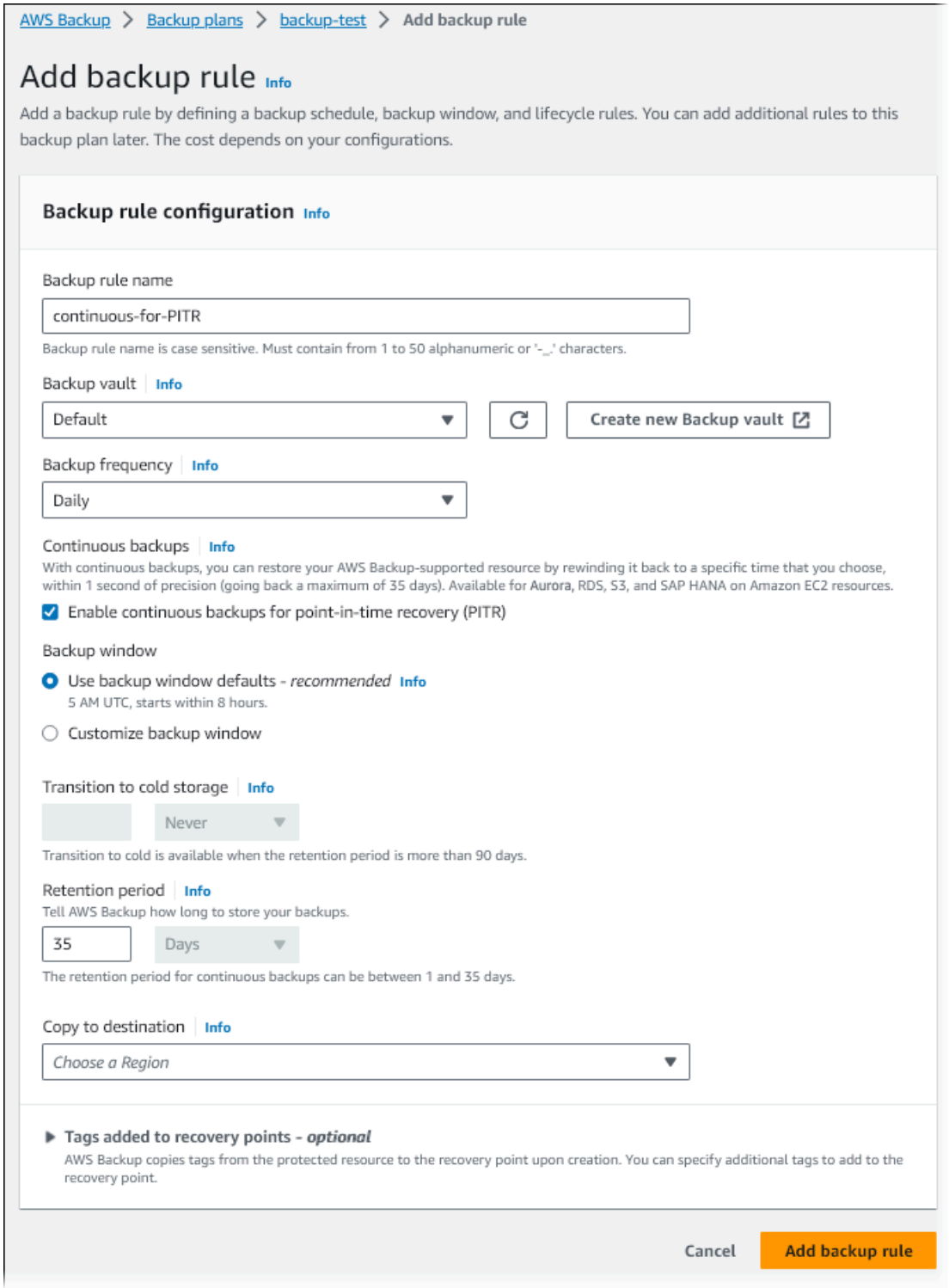
AWS Backup での継続的バックアップの有効化

バックアップルールで継続的バックアップを有効にします。

PITR の継続的なバックアップを有効にするには

1. AWS Management Console にサインインして、AWS Backup コンソール (https://console.aws.amazon.com/backup) を開きます。
2. ナビゲーションペインで、[バックアッププラン] を選択します。
3. [バックアッププラン名] で、DB クラスターのバックアップに使用するバックアッププランを選択します。
4. [バックアップルール] セクションで、[バックアップルールの追加] を選択します。

   [バックアップルールを追加] ページが表示されます。
5. [ポイントインタイムリカバリ (PITR) の継続的バックアップを有効にする] チェックボックスを選択します。



6. 必要に応じて他の設定を選択し、[バックアップルールの追加] を選択します。

AWS Backup での継続的なバックアップからの復元

バックアップボールトから指定した時刻に復元します。

コンソール

AWS Management Console を使用して、DB クラスターを特定の時点に復元旧できます。

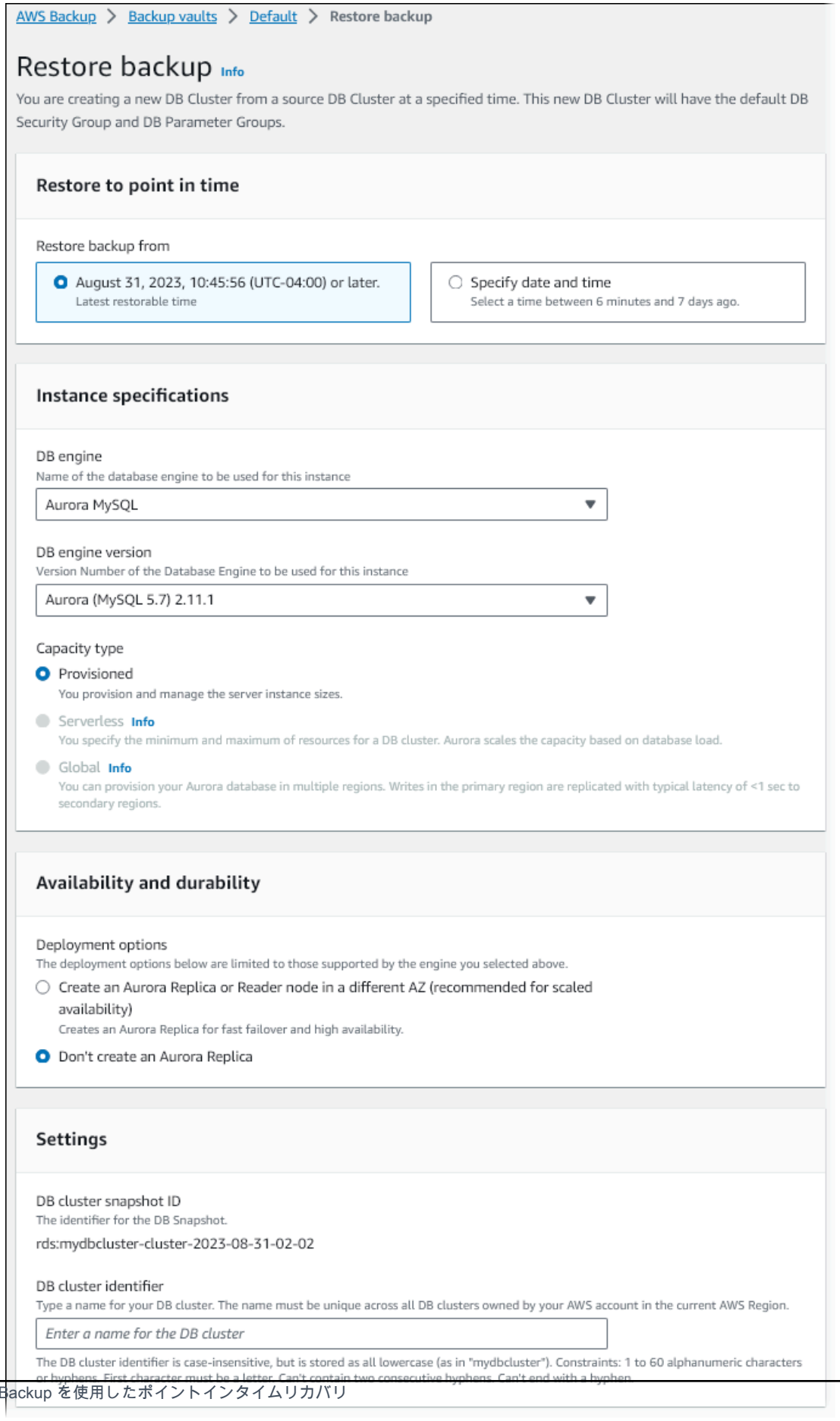
AWS Backup で継続的なバックアップから復元するには

1. AWS Management Console にサインインして、AWS Backup コンソール (https://console.aws.amazon.com/backup) を開きます。
2. ナビゲーションペインで、[バックアップボールト] を選択します。
3. たとえば、継続的バックアップを含むバックアップボールト (例: デフォルト) を選択します。

   バックアップボールトの詳細ページが表示されます。
4. [復旧ポイント] で、自動バックアップの復旧ポイントを選択します。

   バックアップタイプは [連続] で、名前は `continuous:cluster-`*`AWS-Backup-job-number`* です。
5. [アクション] で、[復旧] を選択します。

   [バックアップを復元] ページが表示されます。


AWS Backup > Backup vaults > Default > Restore backup
Restore backup Info
You are creating a new DB Cluster from a source DB Cluster at a specified time. This new DB Cluster will have the default DB Security Group and DB Parameter Groups.
Restore to point in time
Restore backup from
August 31, 2023, 10:45:56 (UTC-04:00) or later.
Latest restorable time
Specify date and time
Select a time between 6 minutes and 7 days ago.
Instance specifications
DB engine
Name of the database engine to be used for this instance
Aurora MySQL
DB engine version
Version Number of the Database Engine to be used for this instance
Aurora (MySQL 5.7) 2.11.1
Capacity type
Provisioned
You provision and manage the server instance sizes.
Serverless Info
You specify the minimum and maximum of resources for a DB cluster. Aurora scales the capacity based on database load.
Global Info
You can provision your Aurora database in multiple regions. Writes in the primary region are replicated with typical latency of <1 sec to secondary regions.
Availability and durability
Deployment options
The deployment options below are limited to those supported by the engine you selected above.
Create an Aurora Replica or Reader node in a different AZ (recommended for scaled availability)
Creates an Aurora Replica for fast failover and high availability.
Don't create an Aurora Replica
Settings
DB cluster snapshot ID
The identifier for the DB Snapshot.
rds:mydbcluster-cluster-2023-08-31-02-02
DB cluster identifier
Type a name for your DB cluster. The name must be unique across all DB clusters owned by your AWS account in the current AWS Region.
Enter a name for the DB cluster
The DB cluster identifier is case-insensitive, but is stored as all lowercase (as in "mydbcluster"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

6. [特定時点への復元] では、[日付と時刻を指定] を選択して、特定の時点に復元します。
7. DB クラスターの復元に必要な他の設定を選択し、[バックアップの復元] を選択します。

   [ジョブ] ページに、[復元ジョブ] ペインが表示されます。ページ上部のメッセージには、復元ジョブに関する情報が表示されます。

DB クラスターが復元されたら、プライマリ (ライター) DB インスタンスを追加する必要があります。DB クラスターのプライマリインスタンスを作成するには、AWS CLI コマンド create-db-instance を使用します。`--db-cluster-identifier` パラメータの値として DB クラスターの名前を含めます。

CLI

start-restore-job AWS CLI コマンドを使用して、DB クラスターを指定した時刻に復元します。以下のパラメータは必須です。

- `--recovery-point-arn` — 復元する元の復旧ポイントの Amazon リソースネーム (ARN)。
- `--resource-type` — `Aurora` を使用します。
- `--iam-role-arn` — AWS Backup オペレーションを使用する IAM ロールの ARN。
- `--metadata` — DB クラスターの復元に使用するメタデータ。以下のパラメータは必須です。
  - `DBClusterIdentifier`
  - `Engine`
  - `RestoreToTime`、または `UseLatestRestorableTime`

次の例は、DB クラスターを指定した時刻に復元する方法を示しています。

```
aws backup start-restore-job \
--recovery-point-arn arn:aws:backup:eu-central-1:123456789012:recovery-
point:continuous:cluster-itsreallyjustanexample1234567890-487278c2 \
--resource-type Aurora \
--iam-role-arn arn:aws:iam::123456789012:role/service-role/AWSBackupDefaultServiceRole
 \
--metadata '{"DBClusterIdentifier":"backup-pitr-test","Engine":"aurora-
mysql","RestoreToTime":"2023-09-01T17:00:00.000Z"}'
```

次の例は、DB クラスターを最新の復元可能な時刻に復元する方法を示しています。

```
aws backup start-restore-job \
```

```
--recovery-point-arn arn:aws:backup:eu-central-1:123456789012:recovery-
point:continuous:cluster-itsreallyjustanexample1234567890-487278c2 \
--resource-type Aurora \
--iam-role-arn arn:aws:iam::123456789012:role/service-role/AWSBackupDefaultServiceRole
 \
--metadata '{"DBClusterIdentifier":"backup-pitr-latest","Engine":"aurora-
mysql","UseLatestRestorableTime":"true"}'
```

DB クラスターが復元されたら、プライマリ (ライター) DB インスタンスを追加する必要があります。DB クラスターのプライマリインスタンスを作成するには、AWS CLI コマンド create-db-instance を使用します。`--db-cluster-identifier` パラメータの値として DB クラスターの名前を含めます。

DB クラスターのスナップショットの削除

Amazon RDS で管理されている DB のクラススナップショットは、不要になったら削除することができます。

> **Note**
>
> AWS Backup で管理されているバックアップを削除するには、AWS Backup コンソールを使用します。AWS Backup の詳細については、「AWS Backup デベロッパーガイド」を参照してください。

DB クラスターのスナップショットの削除

DB クラススナップショットを削除するには、コンソール、AWS CLI、または RDS API を使用します。

共有またはパブリックのスナップショットを削除するには、そのスナップショットを所有する AWS アカウントにサインインする必要があります。

コンソール

DB クラススナップショットを削除するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Snapshots] を選択します。
3. 削除する DB クラススナップショットを選択します。
4. 「アクション」 で、「スナップショットの削除」 を選択します。
5. 確認ページで、[削除] を選択します。

AWS CLI

DB クラススナップショットを削除するには、AWS CLI の delete-db-cluster-snapshot コマンドを使用します。

DB クラススナップショットを削除するには、以下のオプションを使用します。

- `--db-cluster-snapshot-identifier` - DB クラススナップショットの識別子。

Example

次のコードは、DB クラススナップショット `mydbclustersnapshot` を削除します。

Linux、macOS、Unix の場合:

```
aws rds delete-db-cluster-snapshot \
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

Windows の場合:

```
aws rds delete-db-cluster-snapshot ^
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

RDS API

DB クラススナップショットを削除するには、Amazon RDS API オペレーションの DeleteDBClusterSnapshot を使用します。

DB クラススナップショットを削除するには、以下のパラメータを使用します。

- `DBClusterSnapshotIdentifier` - DB クラススナップショットの識別子。

チュートリアル: DB クラスターのスナップショットから Amazon Aurora DB クラスターを復元する

Amazon Aurora を使用した作業でよくあるシナリオとして、ときどき DB インスタンスで作業をするが常に必要なわけではない、ということがあります。例えば、四半期ごとにのみ実行するレポートのデータを保持するために DB クラスターを使用する場合があります。このようなシナリオで経費を節約する 1 つの方法として、レポートが完了した後、DB クラスターの DB クラスタースナップショットを作成することが挙げられます。その後、DB クラスターを削除し、新しいデータをアップロードして次の四半期にレポートを実行する必要があるときに復元します。

DB クラスターを復元するときは、復元の元となる DB クラスタースナップショットの名前を指定します。次に、復元オペレーションから作成される新しい DB クラスターの名前を指定します。スナップショットから DB クラスターを復元する方法の詳細については、「DB クラスターのスナップショットからの復元」を参照してください。

このチュートリアルでは、復元された DB クラスターを Aurora MySQL バージョン 2 (MySQL 5.7 と互換) から Aurora MySQL バージョン 3 (MySQL 8.0 と互換) にアップグレードします。

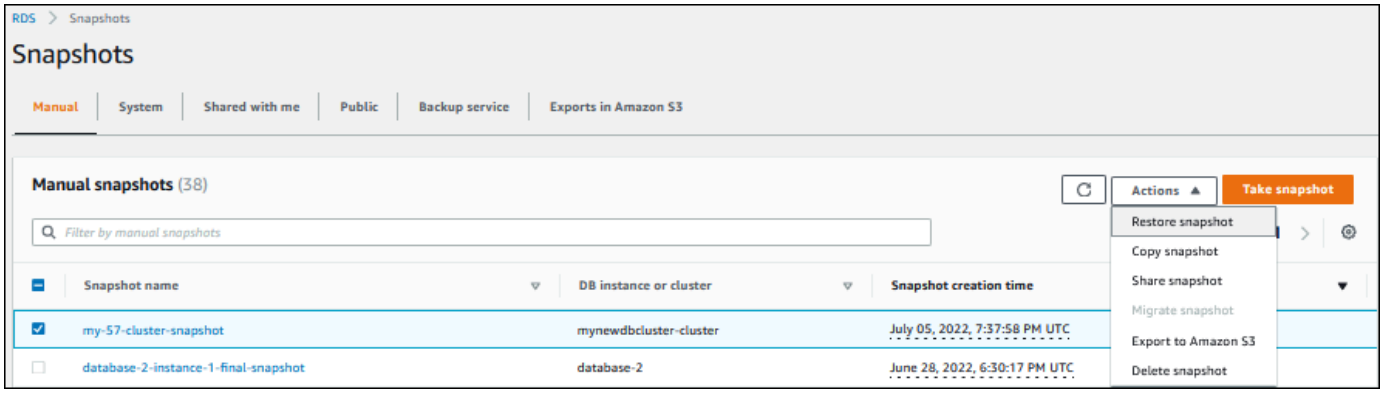
Amazon RDS コンソールを使用して DB クラスターのスナップショットから DB クラスターを復元する

AWS Management Console を使用してスナップショットから DB クラスターを復元すると、プライマリ (ライター) DB インスタンスも作成されます。

> **Note**
>
> プライマリ DB インスタンスは、作成中はリーダーインスタンスとして表示されますが、作成後はライターインスタンスになります。

DB クラスターのスナップショットから DB クラスターを復元するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Snapshots] を選択します。
3. 復元の元にする DB クラスタースナップショットを選択します。
4. [アクション]、[スナップショットを復元] の順に選択します。

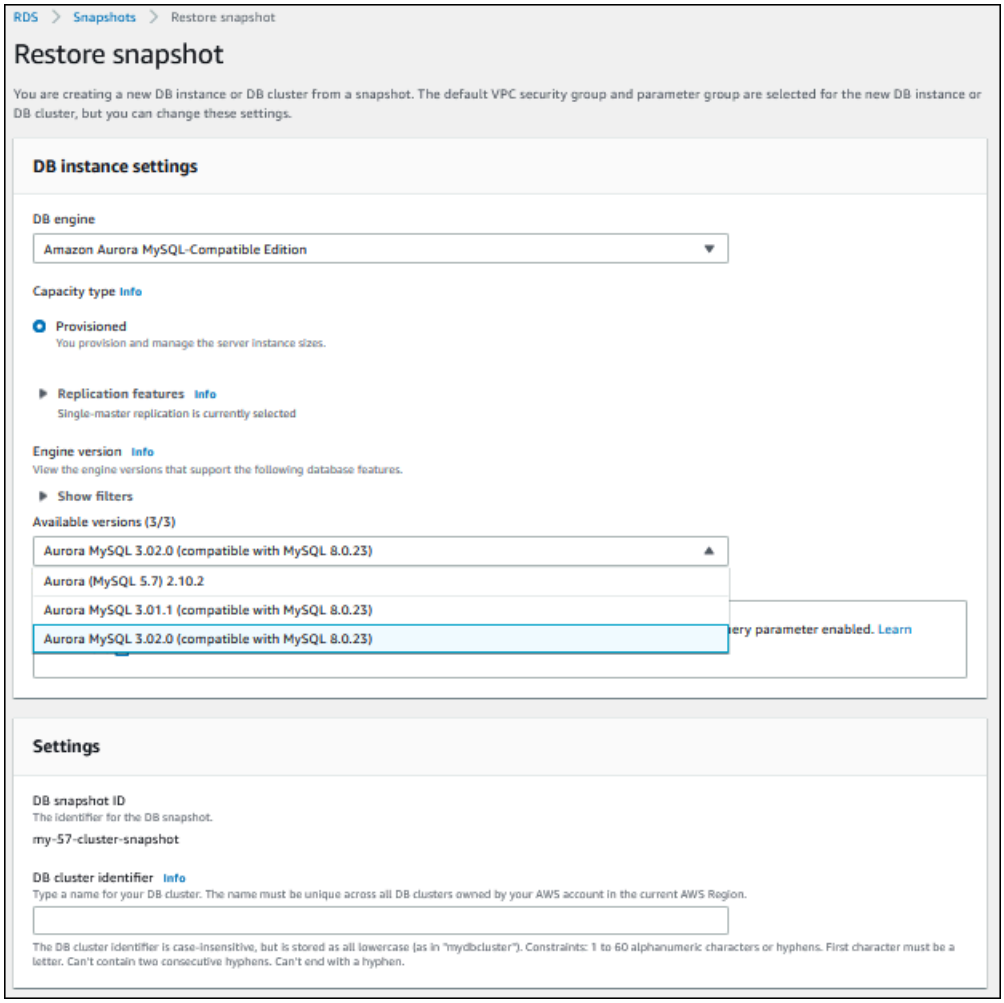


[Restore snapshot (スナップショットの復元)] ページが表示されます。

5. [DB instance settings] (DB インスタンスの設定) で、次のようにします。

   a. DB エンジンのデフォルト設定を使用します。

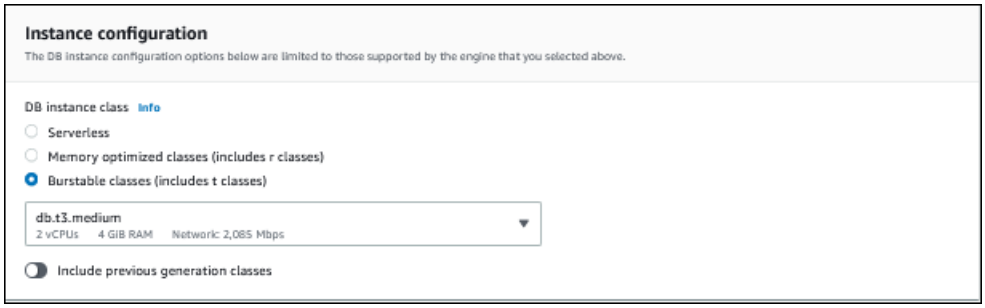
   b. 使用可能なバージョンについては、Aurora MySQL 3.02.0 (MySQL 8.0.23 と互換) など、MySQL—8.0 と互換性があるバージョンを選択します。



6. [Settings] (設定) の [DB cluster identifier] (DB クラスター識別子) に、復元する DB クラスターに使用する名前を入力します (例: **`my-80-cluster`**)。
7. [Connectivity] (接続) では、次のデフォルトの設定を使用します。
   - Virtual Private Cloud (VPC)
   - DB サブネットグループ
   - パブリックアクセス
   - VPC セキュリティグループ (ファイアウォール)
8. DB インスタンスクラスを選択します。

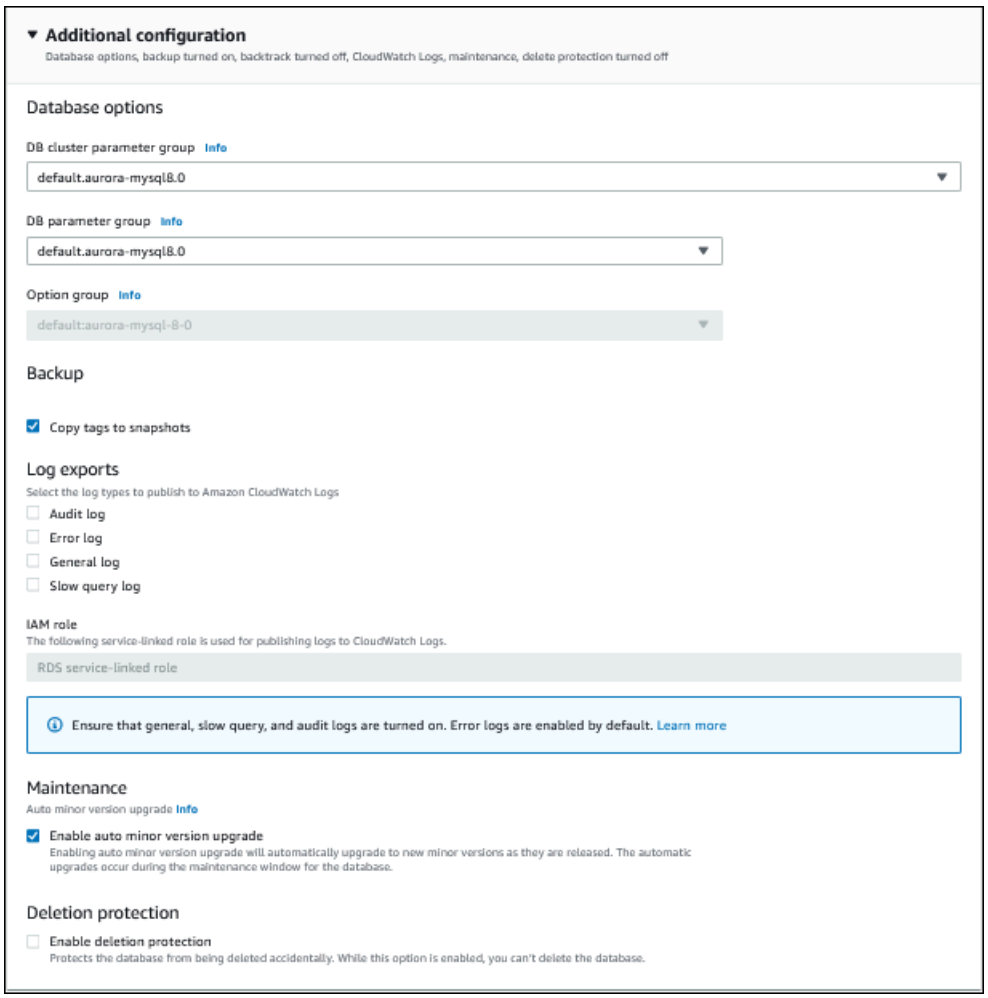
   このチュートリアルでは、[Burstable classes (includes t classes)] (バースト可能クラス (t クラスを含む)) を選択してから、[db.t3.medium] を選択します。

> **Note**
>
> T DB インスタンスクラスは、開発およびテストサーバー、またはその他の本稼働以外のサーバーにのみ使用することをお勧めします。T インスタンスクラスの詳細については、「DB インスタンスクラスタイプ」を参照してください。



9. [Database authentication] (データベース認証) では、デフォルトの設定を使用します。
10. [Encryption] (暗号化) では、デフォルトの設定を使用します。

    スナップショットのソース DB クラスターが暗号化されている場合、復元される DB クラスターも暗号化されます。暗号化を解除することはできません。
11. ページの下部の [Additional configuration] (追加設定) を開きます。



12. 以下の選択を行います。

    a. このチュートリアルでは、DB クラスターパラメーターグループのデフォルト値を使用します。

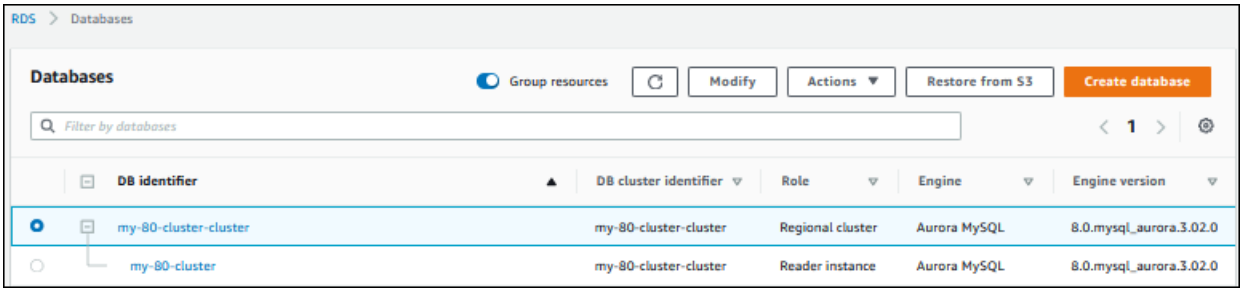
    b. このチュートリアルでは、DB パラメーターグループのデフォルト値を使用します。

    c. [Log exports] (ログのエクスポート) で、すべてのチェックボックスをオンにします。

    d. [Deletion protection] (削除保護) で、[Enable deletion protection] (削除保護の有効化) チェックボックスをオンにします。

13. DB インスタンスの復元 を選択します。

[Databases] (データベース) ページには、ステータスが `Creating` である復元された DB クラスターが表示されます。



プライマリ DB インスタンスは、作成中はリーダーインスタンスとして表示されますが、作成後はライターインスタンスになります。

AWS CLI を使用して DB クラスターのスナップショットから DB クラスターを復元する

AWS CLI を使用してスナップショットから DB クラスターを復元するには、次の 2 つのステップを実行します。

1. restore-db-cluster-from-snapshot コマンドを使用した DB クラスターの復元
2. create-db-instance コマンドを使用した プライマリ (ライター) DB インスタンスの作成

DB クラスターの復元

`restore-db-cluster-from-snapshot` コマンドを使用します。以下のような必須オプションがあります。

- `--db-cluster-identifier` - 復元された DB クラスターの名前。
- `--snapshot-identifier` - 復元の元となるスナップショットの名前。
- `--engine` - 復元された DB クラスターのデータベースエンジン。復元するクラスターは、ソース DB クラスターのデータベースエンジンと互換性がある必要があります。

  選択肢は次のとおりです。

  - `aurora-mysql` – Aurora MySQL 5.7 および 8.0 互換。
  - `aurora-postgresql` – Aurora PostgreSQL 互換。

  この例では、`aurora-mysql` を使用します。
- `--engine-version` - 復元された DB クラスターのバージョン。この例では、MySQL-8.0 と互換性があるバージョンを使用します。

次の例では、my-57-cluster-snapshot という名前の DB クラスターのスナップショットから my-new-80-cluster という名前のAurora MySQL 8.0 と互換性がある DB クラスターを復元します。

DB クラスターを復元するには

- 以下のいずれかのコマンドを使用します。

  Linux、macOS、Unix の場合:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier my-new-80-cluster \
    --snapshot-identifier my-57-cluster-snapshot \
    --engine aurora-mysql \
    --engine-version 8.0.mysql_aurora.3.02.0
```

  Windows の場合:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier my-new-80-cluster ^
    --snapshot-identifier my-57-cluster-snapshot ^
    --engine aurora-mysql ^
    --engine-version 8.0.mysql_aurora.3.02.0
```

出力は以下のようになります。

```
{
    "DBCluster": {
        "AllocatedStorage": 1,
        "AvailabilityZones": [
            "eu-central-1b",
            "eu-central-1c",
            "eu-central-1a"
        ],
        "BackupRetentionPeriod": 14,
        "DatabaseName": "",
        "DBClusterIdentifier": "my-new-80-cluster",
        "DBClusterParameterGroup": "default.aurora-mysql8.0",
        "DBSubnetGroup": "default",
        "Status": "creating",
```

```
        "Endpoint": "my-new-80-cluster.cluster-############.eu-
central-1.rds.amazonaws.com",
        "ReaderEndpoint": "my-new-80-cluster.cluster-ro-############.eu-
central-1.rds.amazonaws.com",
        "MultiAZ": false,
        "Engine": "aurora-mysql",
        "EngineVersion": "8.0.mysql_aurora.3.02.0",
        "Port": 3306,
        "MasterUsername": "admin",
        "PreferredBackupWindow": "01:55-02:25",
        "PreferredMaintenanceWindow": "thu:21:14-thu:21:44",
        "ReadReplicaIdentifiers": [],
        "DBClusterMembers": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-########",
                "Status": "active"
            }
        ],
        "HostedZoneId": "Z1RLNU0EXAMPLE",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:eu-central-1:123456789012:key/#######-5ccc-49cc-8aaa-
############",
        "DbClusterResourceId": "cluster-ZZ12345678ITSJUSTANEXAMPLE",
        "DBClusterArn": "arn:aws:rds:eu-central-1:123456789012:cluster:my-new-80-
cluster",
        "AssociatedRoles": [],
        "IAMDatabaseAuthenticationEnabled": false,
        "ClusterCreateTime": "2022-07-05T20:45:42.171000+00:00",
        "EngineMode": "provisioned",
        "DeletionProtection": false,
        "HttpEndpointEnabled": false,
        "CopyTagsToSnapshot": false,
        "CrossAccountClone": false,
        "DomainMemberships": [],
        "TagList": []
    }
}
```

プライマリ (ライター) DB インスタンスの作成

プライマリ (ライター) DB インスタンスを作成するには、`create-db-instance` コマンドを使用します。以下のような必須オプションがあります。

- `--db-cluster-identifier` - 復元された DB クラスターの名前。
- `--db-instance-identifier` - プライマリ DB インスタンスの名前。
- `--db-instance-class` - プライマリ DB インスタンスのインスタンスクラス。この例では、`db.t3.medium` を使用します。

> **Note**
>
> T DB インスタンスクラスは、開発およびテストサーバー、またはその他の本稼働以外のサーバーにのみ使用することをお勧めします。T インスタンスクラスの詳細については、「DB インスタンスクラスタイプ」を参照してください。

- `--engine`— プライマリ DB インスタンスのデータベースエンジン。データベースエンジンは、復元された DB クラスターが使用するエンジンと同じである必要があります。

  選択肢は次のとおりです。

  - `aurora-mysql` – Aurora MySQL 5.7 および 8.0 互換。
  - `aurora-postgresql` – Aurora PostgreSQL 互換。

  この例では、`aurora-mysql` を使用します。

次の例では、`my-new-80-cluster` という名前の復元された Aurora MySQL 8.0 互換 DB クラスターで `my-new-80-cluster-instance` という名前のプライマリ (ライター) DB インスタンスを作成します。

プライマリ DB インスタンスを作成するには

- 以下のいずれかのコマンドを使用します。

  Linux、macOS、Unix の場合:

```
aws rds create-db-instance \
    --db-cluster-identifier my-new-80-cluster \
    --db-instance-identifier my-new-80-cluster-instance \
    --db-instance-class db.t3.medium \
    --engine aurora-mysql
```

  Windows の場合:

```
aws rds create-db-instance ^
    --db-cluster-identifier my-new-80-cluster ^
    --db-instance-identifier my-new-80-cluster-instance ^
    --db-instance-class db.t3.medium ^
    --engine aurora-mysql
```

出力は以下のようになります。

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "my-new-80-cluster-instance",
        "DBInstanceClass": "db.t3.medium",
        "Engine": "aurora-mysql",
        "DBInstanceStatus": "creating",
        "MasterUsername": "admin",
        "AllocatedStorage": 1,
        "PreferredBackupWindow": "01:55-02:25",
        "BackupRetentionPeriod": 14,
        "DBSecurityGroups": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-########",
                "Status": "active"
            }
        ],
        "DBParameterGroups": [
            {
                "DBParameterGroupName": "default.aurora-mysql8.0",
                "ParameterApplyStatus": "in-sync"
            }
        ],
        "DBSubnetGroup": {
            "DBSubnetGroupName": "default",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-2305ca49",
            "SubnetGroupStatus": "Complete",
            "Subnets": [
                {
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "eu-central-1a"
```

```
                    },
                    "SubnetOutpost": {},
                    "SubnetStatus": "Active"
                },
                {
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "eu-central-1b"
                    },
                    "SubnetOutpost": {},
                    "SubnetStatus": "Active"
                },
                {
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "eu-central-1c"
                    },
                    "SubnetOutpost": {},
                    "SubnetStatus": "Active"
                }
            ]
        },
        "PreferredMaintenanceWindow": "sat:02:41-sat:03:11",
        "PendingModifiedValues": {},
        "MultiAZ": false,
        "EngineVersion": "8.0.mysql_aurora.3.02.0",
        "AutoMinorVersionUpgrade": true,
        "ReadReplicaDBInstanceIdentifiers": [],
        "LicenseModel": "general-public-license",
        "OptionGroupMemberships": [
            {
                "OptionGroupName": "default:aurora-mysql-8-0",
                "Status": "in-sync"
            }
        ],
        "PubliclyAccessible": false,
        "StorageType": "aurora",
        "DbInstancePort": 0,
        "DBClusterIdentifier": "my-new-80-cluster",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:eu-central-1:534026745191:key/#######-5ccc-49cc-8aaa-
############",
        "DbiResourceId": "db-5C6UT5PU0YETANOTHEREXAMPLE",
        "CACertificateIdentifier": "rds-ca-2019",
```

```
        "DomainMemberships": [],
        "CopyTagsToSnapshot": false,
        "MonitoringInterval": 0,
        "PromotionTier": 1,
        "DBInstanceArn": "arn:aws:rds:eu-central-1:123456789012:db:my-new-80-cluster-
instance",
        "IAMDatabaseAuthenticationEnabled": false,
        "PerformanceInsightsEnabled": false,
        "DeletionProtection": false,
        "AssociatedRoles": [],
        "TagList": []
    }
}
```

Amazon Aurora クラスターでのメトリクスのモニタリング

Amazon Aurora は、レプリケートされたデータベースサーバーのクラスターを使用します。通常、Aurora クラスターをモニタリングするには、複数の DB インスタンスの状態を確認する必要があります。インスタンスには、主に書き込みオペレーションか、読み取り専用オペレーション、またはこれらの組み合わせを処理する特殊なロールがあります。また、レプリケーションラグを測定することによって、クラスターの全体的な状態をモニタリングします。これは、1 つの DB インスタンスによって加えられた変更が他のインスタンスで使用可能になるまでの時間です。

トピック

Amazon Aurora のメトリクスのモニタリングの概要

モニタリングは、Amazon Aurora と AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。マルチポイント障害をより簡単にデバッグするには、AWS ソリューションのすべての部分からモニタリングデータを収集することをお勧めします。

トピック

モニタリング計画

Amazon Aurora のモニタリングをスタートする前に、モニタリングプランを作成します。この計画で、以下の質問に答えるようにします。

- どのような目的でモニタリングしますか?
- どのリソースをモニタリングしますか?
- どのくらいの頻度でこれらのリソースをモニタリングしますか?
- どのモニタリングツールを使用しますか?
- 誰がモニタリングタスクを実行しますか?
- 問題が発生したときに誰に通知しますか?

パフォーマンスのベースライン

モニタリング目標を達成するには、ベースラインを確立する必要があります。これを行うには、Amazon Aurora 環境で負荷条件と時期をさまざまに変えてパフォーマンスを測定します。次のようなメトリクスをモニタリングできます。

- ネットワークスループット
- クライアント接続
- 読み取り、書き込み、メタデータのいずれかのオペレーションの I/O
- DB インスタンスのバーストクレジットバランス

Amazon Aurora の履歴パフォーマンスデータを保存することをお勧めします。保存したデータを使用して、現在のパフォーマンスを過去の傾向と比較できます。また、正常なパフォーマンスパターンを異常から区別し、問題に対処するための方法を考案することもできます。

パフォーマンスガイドライン

一般的に、パフォーマンスメトリクスの許容値は、ベースラインに対してアプリケーションの現在の動作によって異なります。ベースラインからの一貫した差異またはトレンドになっている差異を調べます。多くの場合、次のメトリクスがパフォーマンスの問題の原因を示しています。

- CPU または RAM の高消費量 - CPU または RAM の消費量が大きい値になっていても、それは妥当である場合があります。ただし、アプリケーションの目標 (スループット、同時実行数など) に沿った想定値であることが前提です。
- ディスクスペースの消費量 - 使用されているディスクスペースが一貫して合計ディスクスペースの 85% 以上である場合は、ディスクスペースの消費量を調べます。インスタンスからデータを削除するか、別のシステムにデータをアーカイブして、スペースを解放できるかどうかを確認します。
- ネットワークトラフィック - ネットワークトラフィックについてシステム管理者に問い合わせて、ドメインネットワークとインターネット接続に対する想定スループットを把握します。スループットが一貫して想定よりも低い場合は、ネットワークトラフィックを調べます。
- データベース接続数 - ユーザー接続数が多いことが、インスタンスのパフォーマンスが下がっていること、応答時間が長くなっていることに関連しているとわかった場合、データベース接続数を制限することを検討します。DB インスタンスの最適なユーザー接続数は、インスタンスのクラスと実行中のオペレーションの複雑さによって異なります。データベース接続数を確認するには、`User Connections` パラメータが 0 (無制限) 以外の値に設定されているパラメータグループと DB インスタンスを関連付けます。既存のパラメータグループを使用するか、新しいパラメータグループを作成できます。詳細については、「「パラメータグループを使用する」」を参照してください。
- IOPS メトリクス - IOPS メトリクスの想定値はディスクの仕様とサーバーの設定によって異なるため、ベースラインを使用して一般的な値を把握します。値とベースラインとの差が一貫しているかどうかを調べます。最適な IOPS パフォーマンスを得るには、読み取りおよび書き込みオペレーションが最小限になるように、一般的な作業セットがメモリに収まることを確認してください。

確立したベースラインをパフォーマンスが下回ると、場合によって、ワークロードに対してデータベースの可用性を最適化するために変更を加える必要があります。例えば、DB インスタンスのイン

スタンスクラスの変更が必要になる場合があります。または、クライアントで使用できる DB インスタンスとリードレプリカの数の変更が必要になる場合があります。

モニタリングツール

モニタリングは、Amazon Aurora およびその他の AWS ソリューションの信頼性、可用性、およびパフォーマンスを維持する上で重要な部分です。AWS には、Amazon Aurora を監視したり、問題が発生したときに報告したり、必要に応じて自動アクションを実行したりするためのモニタリングツールが用意されています。

トピック

- 自動モニタリングツール
- 手動モニタリングツール

自動モニタリングツール

モニタリングタスクをできるだけ自動化することをお勧めします。

トピック

- Amazon Aurora クラスターステータスと推奨事項
- Amazon Aurora の Amazon CloudWatch メトリクス
- Amazon RDS Performance Insights とオペレーティングシステムのモニタリング
- 統合サービス

Amazon Aurora クラスターステータスと推奨事項

以下の自動化されたツールを使用して、Amazon Aurora をモニタリングし、問題が発生したときにレポートできます。

- Amazon Aurora クラスターステータス – Amazon RDS コンソール、AWS CLI、または RDS API を使用して、クラスターの現在のステータスに関する詳細を表示します。
- Amazon Aurora 推奨事項 — DB インスタンス、DB クラスター、、DB クラスターパラメータグループなどのデータベースリソースに関する推奨事項が自動的に表示されます。詳細については、「Amazon Aurora の推奨事項の表示とこれらに対する対応」を参照してください。

Amazon Aurora の Amazon CloudWatch メトリクス

Amazon Aurora が Amazon CloudWatch と統合し、追加のモニタリング機能が利用できるようになりました。

- Amazon CloudWatch - このサービスは AWS で実行されている AWS リソースやアプリケーションをリアルタイムにモニタリングします。次の Amazon CloudWatch 機能を Amazon Aurora で使用できます。
  - Amazon CloudWatch メトリクス - Amazon Aurora は、アクティブな各データベースのメトリクスを 1 分ごとに CloudWatch に自動送信します。CloudWatch の Amazon RDS メトリクスに対する追加料金は発生しません。詳細については、Amazon Aurora の Amazon CloudWatch メトリクスを参照してください。
  - Amazon CloudWatch アラーム - 特定の期間にわたって 1 つの Amazon Aurora メトリクスをモニタリングできます。そのため、設定したしきい値に関連するメトリクスの値に基づいて、1 つ以上のアクションを実行できます。

Amazon RDS Performance Insights とオペレーティングシステムのモニタリング

Amazon Aurora のパフォーマンスをモニタリングするには、次の自動化されたツールを使用します。

- Amazon RDS Performance Insights – データベース負荷を評価し、いつどこに措置を講じたらよいかを判断します。詳しくは、「Amazon Aurora での Performance Insights を使用したDB 負荷のモニタリング」を参照してください。
- Amazon RDS 拡張モニタリング – オペレーティングシステムのメトリクスをリアルタイムで参照します。詳しくは、「拡張モニタリングを使用した OS メトリクスのモニタリング」を参照してください。

統合サービス

次の AWS のサービスが Amazon Aurora と統合されます。

- Amazon EventBridge は、アプリケーションをさまざまなソースのデータに簡単に接続できるようにするサーバーレスイベントバスサービスです。詳しくは、「Amazon Aurora イベントのモニタリング」を参照してください。

- Amazon CloudWatch Logs を使用すると、Amazon Aurora インスタンス、CloudTrail、およびその他のソースからのログファイルをモニタリングして保存し、それらにアクセスすることができます。詳しくは、「Amazon Aurora ログファイルのモニタリング」を参照してください。
- AWS CloudTrail は、AWS アカウント により、またはそのアカウントに代わって行われた API コールおよび関連イベントを取得し、指定した Amazon S3 バケットにログファイルを配信します。詳しくは、「AWS CloudTrail での Amazon Aurora API コールのモニタリング」を参照してください。
- データベースアクティビティストリーミングは、Amazon Aurora の一機能であり、 DB クラスター内のアクティビティのストリーミングをほぼリアルタイムで提供します。詳しくは、「データベースアクティビティストリームを使用した Amazon Aurora のモニタリング」を参照してください。
- DevOps Guru for RDS は Amazon DevOps Guru の一機能であり、Amazon Aurora データベースの Performance Insights メトリクスに機械学習を適用します。詳しくは、「Amazon DevOps Guru for Amazon RDS でパフォーマンスの異常を分析する」を参照してください。

手動モニタリングツール

CloudWatch アラームがカバーしない項目については、手動でモニタリングする必要があります。Amazon RDS、CloudWatch、AWS Trusted Advisor などの AWS コンソールダッシュボードには、AWS 環境の状態が一目でわかるビューが表示されます。また、DB インスタンスのログファイルを確認することをお勧めします。

- Amazon RDS コンソールから、リソースに関する以下の項目をモニタリングできます。
  - DB インスタンスへの接続の数
  - DB インスタンスへの読み書きオペレーションの量
  - DB インスタンスが現在使用しているストレージの量
  - DB インスタンスに使用されているメモリと CPU の量
  - DB インスタンスとの間で送受信されるネットワークトラフィックの量
- Trusted Advisor ダッシュボードから、以下のコスト最適化、セキュリティ、対障害性、パフォーマンス向上のチェックを確認できます。
  - Amazon RDS アイドル DB インスタンス
  - Amazon RDS セキュリティグループのアクセスリスク
  - Amazon RDS バックアップ
  - Amazon RDS Multi-AZ

  - Aurora DB インスタンスのアクセシビリティ

  これらのチェックの詳細については、「Trusted Advisor のベストプラクティス (チェック)」を参照してください。
- CloudWatch ホームページには、次の内容が表示されます。
  - 現在のアラームとステータス
  - アラームとリソースのグラフ
  - サービスのヘルスステータス

  また、CloudWatch を使用して、次のことが可能です。
  - 重視するサービスをモニタリングするためのカスタマイズしたダッシュボードを作成する。
  - メトリクスデータをグラフ化して、問題をトラブルシューティングして、傾向を確認する。
  - AWS リソースのすべてのメトリクスを検索およびブラウズする。
  - 問題があることを通知するアラームを作成/編集する。

クラスターのステータスの表示

Amazon RDS コンソールを使用すると、DB クラスターのステータスにすばやくアクセスできます。

トピック

- Amazon Aurora DB クラスターの表示
- DB クラスターステータスの表示
- Aurora クラスター内の DB インスタンスのステータスの表示

Amazon Aurora DB クラスターの表示

DB クラスター内の Amazon Aurora DB クラスターと DB インスタンスの情報を表示するためのさまざまなオプションがあります。

- Amazon RDS コンソールで DB クラスターおよび DB インスタンスを表示するには、ナビゲーションペインから [データベース] を選択します。
- AWS Command Line Interface (AWS CLI) を使用して DB クラスターと DB インスタンスの情報を取得できます。
- Amazon RDS API を使用して DB クラスターと DB インスタンスの情報を取得できます。

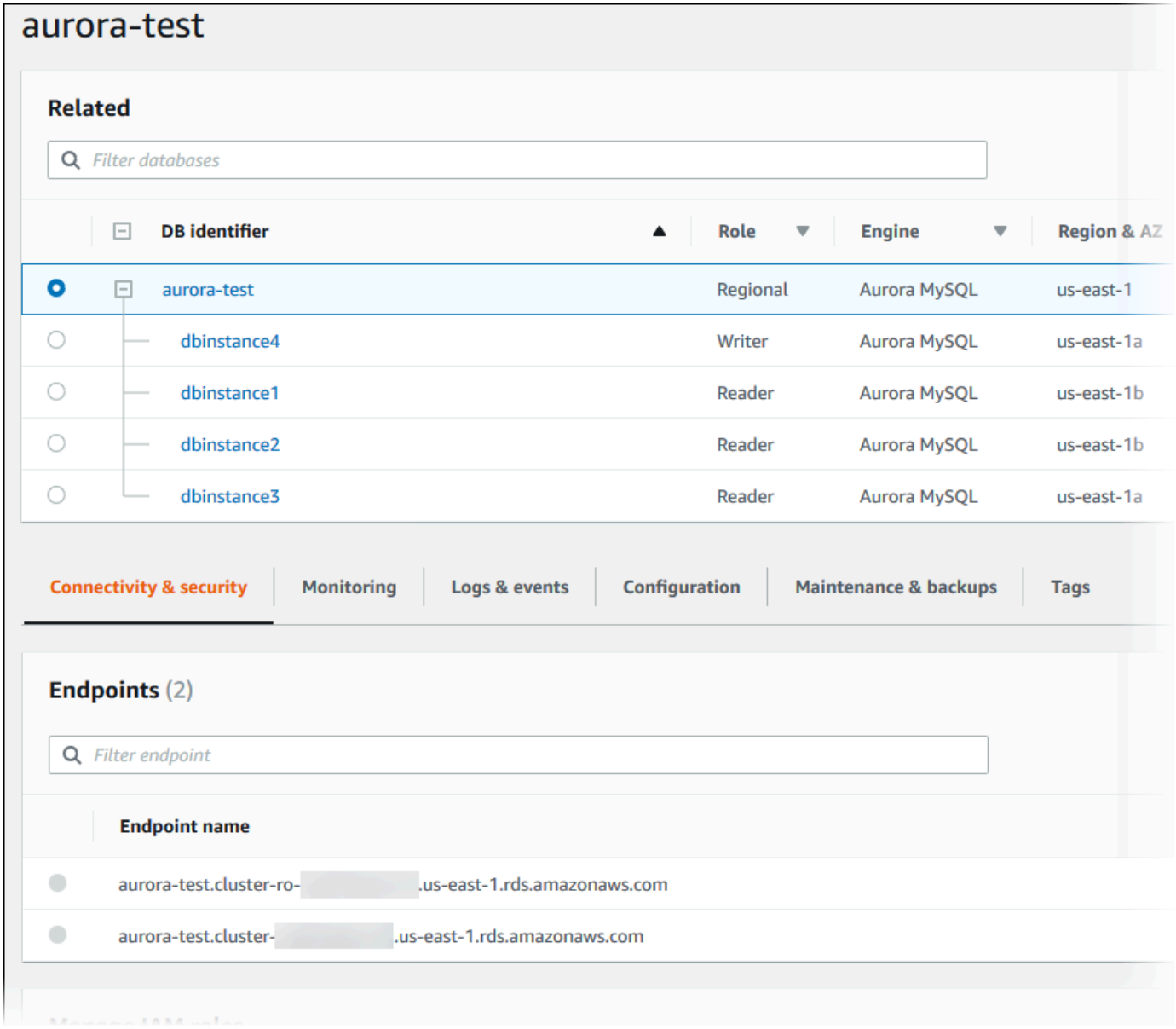
コンソール

Amazon RDS コンソールで、DB クラスターに関する詳細を表示するには、コンソールのナビゲーションペインから [データベース] を選択します。また、Amazon Aurora DB クラスターのメンバーである DB インスタンスの詳細について確認できます。

Amazon RDS コンソールで DB クラスターを表示または変更するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、データベースを選択します。
3. リストから表示したい Aurora DB クラスターの名前を選択します。

   例えば、次のイメージは `aurora-test` という DB クラスターの詳細ページを示しています。DB クラスターには、[DB identifier (DB 識別子)] リストに表示される 4 つの DB インスタンスがあります。ライター DB インスタンス `dbinstance4` は、DB クラスターのプライマリ DB インスタンスです。



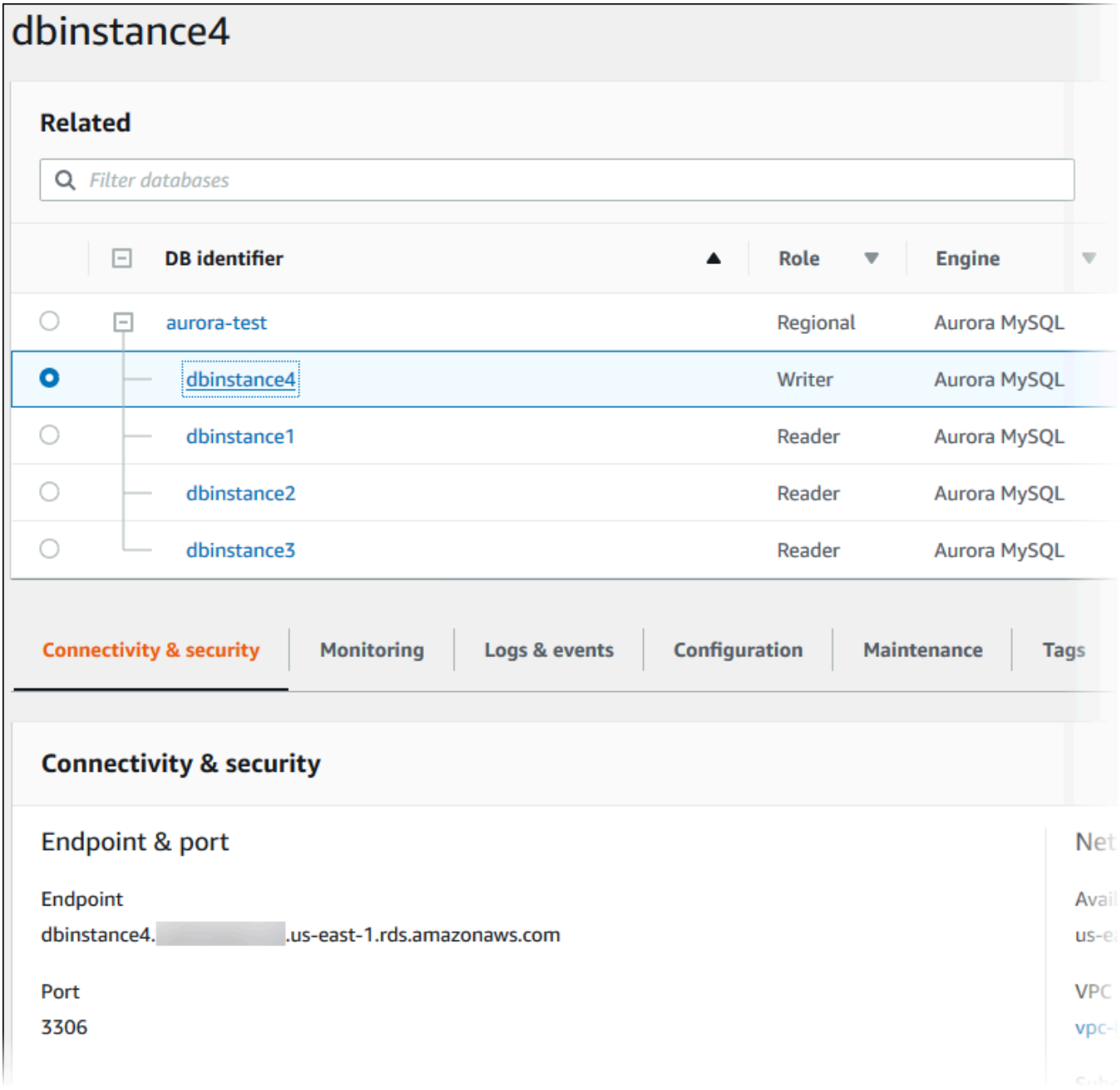
4. DB クラスターを変更するには、リストから DB クラスターを選択して [Modify] (変更) を選択します。

Amazon RDS コンソールで DB クラスターの DB インスタンスを表示または変更するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、データベースを選択します。
3. 次のいずれかを行います。

- DB インスタンスを表示するには、リストから Aurora DB クラスターのメンバーであるインスタンスを選択します。

  例えば、dbinstance4 DB インスタンス識別子を選択すると、次の図に示されているように、コンソールで dbinstance4 DB インスタンスの詳細ページが表示されます。



- DB インスタンスを変更するには、リストから DB インスタンスを選択し、[Modify] (変更) を選択します。DB クラスターの変更の詳細については、「Amazon Aurora DB クラスターの変更」を参照してください。

AWS CLI

AWS CLI を使用して DB クラスターの情報を表示するには、describe-db-clusters コマンドを使用します。例えば、次の AWS CLI コマンドは、設定されている us-east-1 アカウントの変更 AWS リージョンにあるすべての DB クラスターの DB クラスター情報を表示します。

```
aws rds describe-db-clusters --region us-east-1
```

お客様の AWS CLI に JSON 出力が設定されている場合、このコマンドは次の出力を返します。

```
{
    "DBClusters": [
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            "Endpoint": "sample-cluster1.cluster-123456789012.us-
east-1.rds.amazonaws.com"
            "AllocatedStorage": 1,
            "DBClusterIdentifier": "sample-cluster1",
            "MasterUsername": "mymasteruser",
            "EarliestRestorableTime": "2023-03-30T03:35:42.563Z",
            "DBClusterMembers": [
                {
                    "IsClusterWriter": false,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "sample-replica"
                },
                {
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "sample-primary"
                }
            ],
            "Port": 3306,
            "PreferredBackupWindow": "03:34-04:04",
            "VpcSecurityGroups": [
                {
                    "Status": "active",
                    "VpcSecurityGroupId": "sg-ddb65fec"
                }
            ],
            "DBSubnetGroup": "default",
            "StorageEncrypted": false,
```

```
            "DatabaseName": "sample",
            "EngineVersion": "5.7.mysql_aurora.2.11.0",
            "DBClusterParameterGroup": "default.aurora-mysql5.7",
            "BackupRetentionPeriod": 1,
            "AvailabilityZones": [
                "us-east-1b",
                "us-east-1c",
                "us-east-1d"
            ],
            "LatestRestorableTime": "2023-03-31T20:06:08.903Z",
            "PreferredMaintenanceWindow": "wed:08:15-wed:08:45"
        },
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            "Endpoint": "aurora-sample.cluster-123456789012.us-
east-1.rds.amazonaws.com",
            "AllocatedStorage": 1,
            "DBClusterIdentifier": "aurora-sample-cluster",
            "MasterUsername": "mymasteruser",
            "EarliestRestorableTime": "2023-03-30T10:21:34.826Z",
            "DBClusterMembers": [
                {
                    "IsClusterWriter": false,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "aurora-replica-sample"
                },
                {
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "aurora-sample"
                }
            ],
            "Port": 3306,
            "PreferredBackupWindow": "10:20-10:50",
            "VpcSecurityGroups": [
                {
                    "Status": "active",
                    "VpcSecurityGroupId": "sg-55da224b"
                }
            ],
            "DBSubnetGroup": "default",
            "StorageEncrypted": false,
            "DatabaseName": "sample",
```

```
            "EngineVersion": "5.7.mysql_aurora.2.11.0",
            "DBClusterParameterGroup": "default.aurora-mysql5.7",
            "BackupRetentionPeriod": 1,
            "AvailabilityZones": [
                "us-east-1b",
                "us-east-1c",
                "us-east-1d"
            ],
            "LatestRestorableTime": "2023-03-31T20:00:11.491Z",
            "PreferredMaintenanceWindow": "sun:03:53-sun:04:23"
        }
    ]
}
```

RDS API

Amazon RDS API を使用して DB クラスターの情報を表示するには、DescribeDBClusters オペレーションを使用します。

DB クラスターステータスの表示

DB クラスターのステータスは、その正常性を示します。Amazon RDS コンソール、AWS CLI、または API を使用して、DB クラスターとクラスターインスタンスのステータスを表示できます。

> **Note**
>
> Aurora では、メンテナンスのステータスと呼ばれる別のステータスも使用します。これは、Amazon RDS コンソールの [Maintenance] 列に表示されます。この値は、DB クラスターに適用する必要があるメンテナンスパッチのステータスを示します。メンテナンスのステータスは、DB クラスターのステータスから独立しています。メンテナンスのステータスの詳細については、「DB クラスターのアップデートを適用する」を参照してください。

DB クラスターの考えられるステータス値を以下の表に示します。

DB クラスターのステータス	請求される	説明
[使用可能]	請求される	DB クラスターは正常で、使用可能です。Aurora サーバーレスクラスターが使用可能で一時停止している場合は、ストレージに対してのみ料金が発生します。
バックアップ中	請求される	DB クラスターは現在バックアップ中です。
バックトラック中	請求される	DB クラスターは現在バックトラック中です。このステータスは Aurora MySQL にのみ適用されます。
クローン失敗	課金されない	DB クラスターのクローンに失敗しました。
[作成中]	課金されない	DB クラスターを作成しています。作成中は DB クラスターにアクセスできません。
[Deleting] (削除中)	課金されない	DB クラスターを削除しています。
フェイルオーバーしています	請求される	プライマリインスタンスから Aurora レプリカへのフェイルオーバーが実行されています。

DB クラスターのステータス	請求される	説明
暗号化認証情報にアクセスできません	非請求対象	DB クラスターの暗号化または復号に使用する AWS KMS key にアクセスしたり、それを復元したりすることはできません。
inaccessible-encryption-credentials-recoverable	ストレージが請求対象	DB クラスターの暗号化または復号に使用する KMS キーにアクセスできません。ただし、KMS キーがアクティブな場合は、DB クラスターを再起動すると復元できます。 詳細については、「Amazon Aurora DB クラスターの暗号化」を参照してください。
メンテナンス	請求される	Amazon RDS は、DB クラスターにメンテナンス更新を適用しています。このステータスは、RDS が事前に十分スケジュールした DB クラスターレベルのメンテナンスに使用されます。
移行中	請求される	DB クラススナップショットは、DB クラスターに復元されています。
移行に失敗しました	課金されない	移行に失敗しました。
変更	請求される	お客様が DB クラスターの変更をリクエストしたため、DB クラスターは変更中です。
昇格中	請求される	リードレプリカがスタンドアロン DB クラスターに昇格中です。
Preparing-data-migration	請求対象	Amazon RDS は Aurora にデータを移行する準備をしています。
名前の変更	請求される	お客様が名前の変更をリクエストしたため、DB クラスターの名前を変更中です。

DB クラスターのステータス	請求される	説明
マスター認証をリセット中	請求される	お客様がリセットをリクエストしたため、DB クラスターのマスター認証情報をリセット中です。
スタート	ストレージが請求対象	DB クラスターはスタート中です。
停止	ストレージが請求対象	DB クラスターは停止しています。
停止中	ストレージが請求対象	DB クラスターは停止しているところです。
ストレージの最適化	請求される	DB インスタンスが変更されて、ストレージのサイズまたはタイプが変更されています。DB インスタンスが完全に動作しています。ただし、DB インスタンスの状態が [ストレージ最適化] である間は、DB インスタンスのストレージに対する変更をリクエストすることはできません。ストレージ最適化プロセスは通常短時間で終了しますが、場合によっては 24 時間以上かかることもあります。
Update-iam-db-auth	請求される	DB クラスターの IAM 認証が更新されています。
アップグレード	請求される	DB クラスターエンジンのバージョンをアップグレード中です。

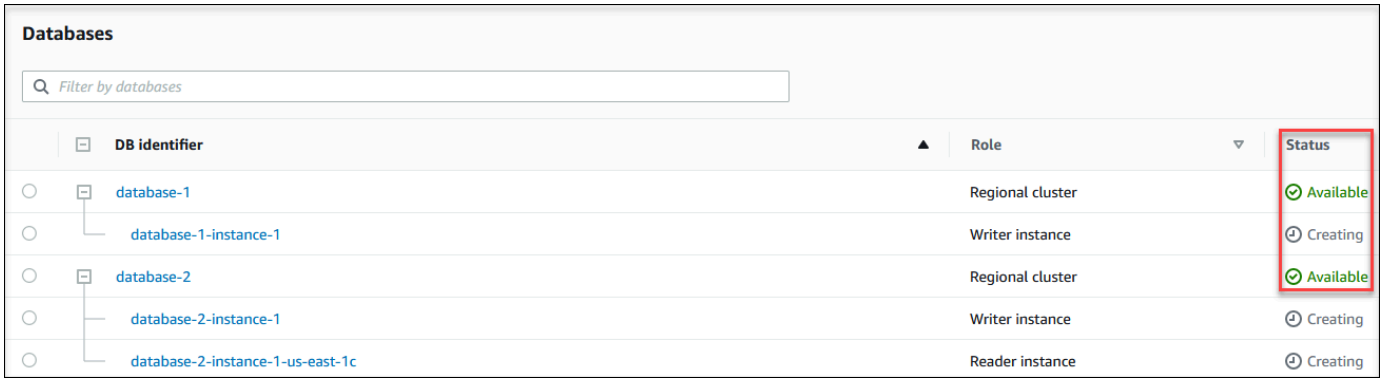
コンソール

DB クラスターのステータスを表示するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。

2. ナビゲーションペインで、データベースを選択します。

   データベースページが DB クラスターのリストとともに表示されます。DB クラスターごとに、ステータス値が表示されます。



CLI

DB クラスターのステータスだけを表示するには、AWS CLI で次のクエリを使用します。

```
aws rds describe-db-clusters --query 'DBClusters[*].[DBClusterIdentifier,Status]' --
output table
```

Aurora クラスター内の DB インスタンスのステータスの表示

Aurora クラスターにある DB インスタンスのステータスは、DB インスタンスの状態を示します。次の手順で、Amazon RDS コンソール、AWS CLI コマンド、または API オペレーションでクラスターの DB インスタンスのステータスを表示できます。

> **Note**
>
> Amazon RDS では、メンテナンスのステータスと呼ばれる別のステータスも使用します。これは、Amazon RDS コンソールの [メンテナンス] 列に表示されます。この値は、DB インスタンスに適用する必要があるメンテナンスパッチのステータスを示します。メンテナンスのステータスは、DB インスタンスのステータスから独立しています。メンテナンスのステータスの詳細については、「DB クラスターのアップデートを適用する」を参照してください。

DB インスタンスの考えられるステータス値を以下の表に示します。また、この表は、DB インスタンスとストレージが請求されるか、ストレージのみ請求されるか、または請求されないかを示します。DB インスタンスのすべてのステータスで、バックアップの使用は常に請求されます。

DB インスタンスのステータス	請求される	説明
[使用可能]	請求される	DB インスタンスは正常で、使用可能です。
バックアップ中	請求される	DB インスタンスを現在バックアップ中です。
バックトラック中	請求される	DB インスタンスを現在バックトラック中です。このステータスは Aurora MySQL にのみ適用されます。

DB インスタンスのステータス	請求される	説明
拡張モニタリングを設定中	請求される	この DB インスタンスに対して拡張モニタリングを有効または無効にしています。
iam データベース認証を設定中	請求される	この DB インスタンスに対して AWS Identity and Access Management (IAM) データベース認証を有効または無効にしています。
ログエクスポートを設定中	請求される	この DB インスタンスに対して Amazon CloudWatch Logs へのログファイルの発行を有効または無効にしています。
vpc に変換中	請求される	DB インスタンスを、Amazon Virtual Private Cloud (Amazon VPC) 外の DB インスタンスから Amazon VPC 内の DB インスタンスに変換中です。
[作成中]	課金されない	DB インスタンスを作成中です。作成中の DB インスタンスにはアクセスできません。
削除 - 事前チェック	非請求対象	Amazon RDS は、リードレプリカが正常で、削除しても安全であることを検証しています。
[Deleting] (削除中)	課金されない	DB インスタンスを削除しています。
[失敗]	課金されない	DB インスタンスでエラーが発生し、Amazon RDS では復旧できません。DB インスタンスの復元可能な直近の時間までポイントインタイムリカバリを実行し、データを復旧してください。

DB インスタンスのステータス	請求される	説明
暗号化認証情報にアクセスできません	非請求対象	DB インスタンスの暗号化または復号に使用する AWS KMS key にアクセスしたり、それを復元したりすることはできません。
inaccessible-encryption-credentials-recoverable	ストレージが請求対象	DB インスタンスの暗号化または復号に使用する KMS キーにアクセスできません。ただし、KMS キーがアクティブな場合は、DB インスタンスを再起動すると復元できます。 詳細については、「Amazon Aurora DB クラスターの暗号化」を参照してください。
互換性のないネットワーク	課金されない	Amazon RDS は、DB インスタンスに対して復旧アクションを実行しようとしていますが、VPC がアクションを完了できない状態にあるため実行できません。このステータスは、例えば、サブネット内の使用可能なすべての IP アドレスが使用中で、Amazon RDS が DB インスタンスの IP アドレスを取得できない場合などに発生する可能性があります。
互換性のないオプショングループ	請求される	Amazon RDS がオプショングループの変更を適用しようとしましたが、適用できませんでした。また、Amazon RDS はオプショングループの前の状態にロールバックできませんでした。詳細については、DB インスタンスの [最近のイベント] 一覧を参照してください。このステータスは、例えば、オプショングループに TDE などのオプションが含まれており、DB インスタンスに暗号化情報が含まれていない場合などに発生する可能性があります。
互換性のないパラメータ	請求される	DB インスタンスの DB パラメータグループに指定されたパラメータが DB インスタンスと互換性がないため、Amazon RDS は DB インスタンスを起動できません。パラメータの変更を元に戻すか、パラメータを DB インスタンスと互換させて、DB インスタンスへのアクセスを回復してください。互換性のないパラメータの詳細については、DB インスタンスの [最近のイベント] 一覧を参照してください。

DB インスタンスのステータス	請求される	説明
互換性のない復元	課金されない	Amazon RDS は、特定の時点への復旧を行うことはできません。この状況の一般的な原因としては、temp テーブルの使用MySQL での MyISAM テーブルの使用が考えられます。
容量不足	非請求対象	十分な容量が現在利用できないため、Amazon RDS はインスタンスを作成できません。同じ AZ に、同じインスタンスタイプで DB インスタンスを作成するには、DB インスタンスを削除し、数時間待ってから、もう一度作成を試みます。または、別のインスタンスクラスまたは AZ を使用して新しいインスタンスを作成します。
メンテナンス	請求される	Amazon RDS は、DB インスタンスにメンテナンス更新を適用しています。このステータスは、RDS が事前に十分スケジュールしたインスタンスレベルのメンテナンスに使用されます。
変更	請求される	ユーザーからの DB インスタンスの変更リクエストに応じて、DB インスタンスを変更中です。
vpc に移動中	請求される	DB インスタンスを新しい Amazon Virtual Private Cloud (Amazon VPC) に移動中です。
再起動	請求される	DB インスタンスの再起動を要求するユーザーのリクエストまたは Amazon RDS プロセスに応じて、インスタンスを再起動中です。
マスター認証をリセット中	請求される	ユーザーからのリセットのリクエストに応じて、DB インスタンスのマスター認証情報をリセット中です。

DB インスタンスのステータス	請求される	説明
名前の変更	請求される	ユーザーからの名前変更のリクエストに応じて、DB インスタンスの名前を変更中です。
復元エラー	請求される	特定時点への復元またはスナップショットからの復元を実行しようとした際に、DB インスタンスでエラーが発生しました。
スタート	ストレージが請求対象	DB インスタンスを起動中です。
停止	ストレージが請求対象	DB インスタンスは停止済みです。
停止中	ストレージが請求対象	DB インスタンスを停止中です。
Storage-config-upgrade	請求対象	DB インスタンスのストレージファイルシステム設定をアップグレード中です。このステータスはブルー/グリーンデプロイ内のグリーンデータベース、または DB インスタンスのリードレプリカにのみ適用されます。

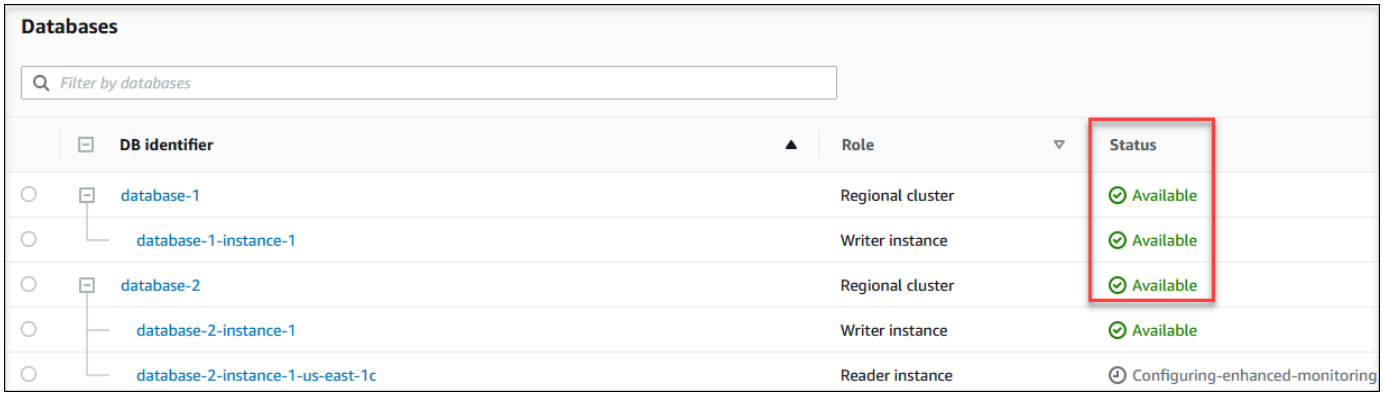
DB インスタンスのステータス	請求される	説明
ストレージ不足	請求される	DB インスタンスが、ストレージ容量の割り当て分に達しました。これは非常に重要なステータスで、この問題はすぐに修正することをお勧めします。これを行うには、DB インスタンスを変更してストレージを拡張します。このような状況を回避するために、ストレージ容量が減少したときに警告を生成する Amazon CloudWatch アラームを設定します。
ストレージの最適化	請求対象	Amazon RDS が、DB インスタンスのストレージを最適化しています。DB インスタンスが完全に動作しています。ストレージ最適化プロセスは通常短時間で終了しますが、場合によっては 24 時間以上かかることもあります。
アップグレード	請求される	データベースエンジンのバージョンをアップグレード中です。

コンソール

DB インスタンスのステータスを表示するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、データベースを選択します。

   データベースページが DB インスタンスのリストとともに表示されます。クラスターの DB クラスターごとに、ステータス値が表示されます。



CLI

AWS CLI を使用して DB インスタンスとそのステータス情報を表示するには、describe-db-instances コマンドを使用します。例えば、次の AWS CLI コマンドは、すべての DB インスタンス情報を一覧表示します。

```
aws rds describe-db-instances
```

特定の DB インスタンスとそのステータスを表示するには、次のオプションを指定して describe-db-instances コマンドを呼び出します。

- `DBInstanceIdentifier` - DB インスタンスの名前です。

```
aws rds describe-db-instances --db-instance-identifier mydbinstance
```

DB インスタンスのステータスだけを表示するには、AWS CLI で次のクエリを使用します。

```
aws rds describe-db-instances --query 'DBInstances[*].
[DBInstanceIdentifier,DBInstanceStatus]' --output table
```

API

Amazon RDS API を使用して DB インスタンスのステータスを表示するには、DescribeDBInstances オペレーションを呼び出します。

Amazon Aurora の推奨事項の表示とこれらに対する対応

Amazon Aurora では、DB インスタンス、DB クラスター、DB パラメータグループなどのデータベースリソースについての推奨事項が自動で表示されます。これらの推奨事項は、DB クラスター構成、DB インスタンス構成、使用状況、パフォーマンスデータを分析して、ベストプラクティスガイダンスを提供します。

Amazon RDS Performance Insights は特定のメトリクスを監視し、特定のリソースで潜在的に問題であると見なされるレベルを分析することで自動的にしきい値を作成します。新しいメトリクス値が事前定義されたしきい値を一定期間にわたって超えた場合に、Performance Insights は事前対応型推奨事項を生成します。この推奨事項は、将来のデータベースパフォーマンスへの影響を防ぐのに役立ちます。例えば、データベースに接続されているセッションがアクティブな作業を行っていないが、データベースリソースがブロックされている可能性がある場合、Aurora PostgreSQL インスタンスに対して「トランザクションでのアイドル状態」という推奨事項が生成されます。事前対応型推奨事項を受け取るには、有料利用枠の保持期間を使用して Performance Insights を有効にする必要があります。Performance Insights を有効にする方法については、「Performance Insights の有効化と無効化」を参照してください。Performance Insights の料金とデータ保持については、「Performance Insights の料金とデータ保持」を参照してください。

DevOps Guru for RDS は特定のメトリクスを監視して、メトリクスの動作が非常に異常または異常になったことを検出します。これらの異常は、推奨事項を含む事後対応型インサイトとして報告されます。例えば、DevOps Guru for RDS は、CPU 容量の増加を検討したり、DB ロードに寄与している待機イベントを調査するよう推奨することがあります。DevOps Guru for RDS は、しきい値ベースの事前対応型推奨事項も提供します。これらの推奨事項を受け取るには、DevOps Guru for RDS を有効にする必要があります。DevOps Guru for RDS を有効にする方法については、「DevOps Guru をオンにしてリソースカバレッジを指定する」を参照してください。

推奨事項のステータスは、アクティブ、却下、保留中、解決済みのいずれかになります。解決済みの推奨事項は 365 日間利用できます。

推奨事項は、表示または却下できます。設定ベースのアクティブな推奨事項をすぐに適用したり、次のメンテナンスウィンドウでスケジュールしたり、却下したりできます。しきい値ベースの事前対応型推奨事項と機械学習ベースの事後対応型推奨事項については、推奨される問題の原因を確認し、推奨アクションを実行して問題を解決する必要があります。

トピック

- Amazon Aurora の推奨事項の表示
- Amazon Aurora 推奨事項への対応

Amazon Aurora の推奨事項の表示

Amazon Aurora では、リソースが作成または変更されると、リソースの推奨事項が生成されます。

設定ベースの推奨事項は、次のリージョンでサポートされています。

- 米国東部 (オハイオ)
- 米国東部 (バージニア北部)
- 米国西部 (北カリフォルニア)
- 米国西部 (オレゴン)
- アジアパシフィック (ムンバイ)
- アジアパシフィック (ソウル)
- アジアパシフィック (シンガポール)
- アジアパシフィック (シドニー)
- アジアパシフィック (東京)
- カナダ (中部)
- 欧州 (フランクフルト)
- 欧州 (アイルランド)
- 欧州 (ロンドン)
- 欧州 (パリ)
- 南米 (サンパウロ)

次の表に設定ベースの推奨事項の例を示します。

型	説明	推奨事項	ダウンタイムが必要	追加情報
リソースの自動バックアップは無効になっています	自動バックアップは DB インスタンスに対して有効ではありません。DB インスタ	最大 14 日間の保存期間で自動バックアップを有効にします。	はい	Aurora DB クラスターのバックアップと復元の概要

型	説明	推奨事項	ダウンタイムが必要	追加情報
	ンスのポイントインタイムリカバリを可能にするため、自動バックアップが推奨されます。			AWS データベースブログの「Amazon RDS バックアップストレージコストの説明」
エンジンのマイナーバージョンアップグレードが必要です	データベースリソースで最新のマイナー DB エンジンバージョンが実行されていません。最新のマイナーバージョンには、最新のセキュリティ修正プログラムやその他の改善が含まれています。	最新のエンジンバージョンにアップグレードします。	はい	Amazon Aurora DB クラスターのメンテナンス
拡張モニタリングは無効になっています	データベースリソースでは拡張モニタリングが有効になっていません。拡張モニタリングにより、モニタリングとトラブルシューティングのためのリアルタイムのオペレーティングシステムメトリクスが提供されます。	Enhanced monitoring] を有効にします。	いいえ	拡張モニタリングを使用した OS メトリクスのモニタリング

型	説明	推奨事項	ダウンタイムが必要	追加情報
ストレージの暗号化は無効になっています。	Amazon RDS では、AWS Key Management Service (AWS KMS) で管理しているキーを使用して、すべてのデータベースエンジンの保存時の暗号化をサポートしています。Amazon RDS 暗号化を使用するアクティブな DB インスタンスでは、ストレージに保存されているデータは、自動バックアップ、リードレプリカ、スナップショットのように暗号化されます。 Aurora DB クラスターの作成時に暗号化が有効になっていない場合は、復号化されたスナップショットを暗号化された DB クラスターに復元する必要があります。	DB クラスターの保管中のデータの暗号化を有効にします。	はい	Amazon Aurora でのセキュリティ

型	説明	推奨事項	ダウンタイムが必要	追加情報
すべてのインスタンスが同じアベイラビリティーゾーンにある DB クラスター	DB クラスターは現在、1 つのアベイラビリティーゾーンにあります。複数のアベイラビリティーゾーンを使用してアベイラビリティーを向上させます。	DB クラスター内の複数のアベイラビリティーゾーンに DB インスタンスを追加します。	いいえ	Amazon Aurora の高可用性
インスタンスサイズが異なるクラスター内の DB インスタンス	DB クラスター内のすべてのインスタンスに同じ DB インスタンスクラスとサイズを使用することをお勧めします。	DB クラスター内のすべての DB インスタンスに同じインスタンスクラスを使用します。	はい	Amazon Aurora でのレプリケーション
異なるインスタンスクラスを持つクラスター内の DB インスタンス	DB クラスター内のすべてのインスタンスに同じ DB インスタンスクラスとサイズを使用することをお勧めします。	DB クラスター内のすべての DB インスタンスに同じインスタンスクラスを使用します。	はい	Amazon Aurora でのレプリケーション
異種のパラメータグループを持つクラスター内の DB インスタンス	DB クラスター内のすべての DB インスタンスが同じ DB パラメータグループを使用することをお勧めします。	DB インスタンスを、DB クラスター内のライターインスタンスに関連付けられた DB パラメータグループに、関連付けます。	いいえ	「パラメータグループを使用する」

型	説明	推奨事項	ダウンタイムが必要	追加情報
Amazon RDS DB クラスターには 1 つの DB インスタンスがあります。	DB クラスターに少なくとも 1 つ以上の DB インスタンスを追加し、可用性とパフォーマンスを向上させます。	リーダー DB インスタンスを DB クラスターに追加します。	いいえ	Amazon Aurora の高可用性
Performance Insights は無効になっている	Performance Insights では、DB インスタンスの負荷をモニタリングし、データベースパフォーマンスの問題の分析と解決をサポートします。Performance Insights を有効にすることをお勧めします。	Performance Insights をオンにします。	いいえ	Amazon Aurora での Performance Insights を使用したDB 負荷のモニタリング
RDS リソースのメジャーバージョンの更新が必須	DB エンジンの、現行メジャーバージョンのデータベースはサポートされません。新しい機能や拡張機能を含む最新のメジャーバージョンにアップグレードすることをお勧めします。	DB エンジンを最新のメジャーバージョンにアップグレードします。	はい	Amazon Aurora の更新 ブルー/グリーンデプロイの作成

型	説明	推奨事項	ダウンタイムが必要	追加情報
DB クラスターは最大 64 TiB のボリュームのみをサポートします	DB クラスターは最大 64 TiB のボリュームをサポートします。最新のエンジンバージョンは、DB クラスターに対して最大 128 TiB のボリュームをサポートします。DB クラスターのエンジンバージョンを最新バージョンにアップグレードして、最大 128 TiB のボリュームをサポートすることをお勧めします。	DB クラスターのエンジンバージョンをアップグレードして、最大 128 TiB のボリュームをサポートするようにします。	はい	Amazon Aurora サイズ制限

型	説明	推奨事項	ダウンタイムが必要	追加情報
すべてのリーダーインスタンスが同じアベイラビリティゾーンにある DB クラスター	アベイラビリティーゾーンは、各 AWS 内で障害が発生した場合に分離できるよう、互いに区別された場所のことです。DB クラスターのプライマリインスタンスとリーダーインスタンスを複数の AZ に配信して、DB クラスターの可用性を改善することをお勧めします。マルチ AZ クラスターはクラスター作成時に、AWS、管理コンソール、AWS CLI または Amazon RDS API を使用して作成できます。また、既存の Aurora クラスターをマルチ AZ クラスターに変更するには、新しいリーダーインスタンスを追加し、別の AZ を指定します。	DB クラスターは、同じアベイラビリティーゾーンにすべての読み込みインスタンスがあります。リーダーインスタンスを複数のアベイラビリティーゾーンに分散することをお勧めします。分散によって可用性が向上し、クライアントとデータベース間のネットワーク遅延が減少して応答時間が改善されます。	いいえ	Amazon Aurora の高可用性

型	説明	推奨事項	ダウンタイムが必要	追加情報
DB のメモリパラメータがデフォルトと異なる	DB インスタンスのメモリパラメータがデフォルト値と大きく異なります。これらの設定はパフォーマンスに影響が及び、エラーの原因となる可能性があります。 DB インスタンスのカスタムメモリパラメータを、DB パラメータグループのデフォルト値に再設定することをお勧めします。	メモリパラメータをデフォルト値にリセットします。	いいえ	「パラメータグループを使用する」

型	説明	推奨事項	ダウンタイムが必要	追加情報
Amazon RDS クエリキャッシュパラメータは有効になっている	変更によってクエリキャッシュの削除が必要になった場合、DB インスタンスは停止しているように見えます。通常ワークロードでは、クエリキャッシュのメリットは得られません。クエリキャッシュは、MySQL バージョン 8.0 から削除されました。query_cache_type パラメータを 0 に設定することをお勧めします。	DB パラメータグループの query_cache_type パラメータを 0 に設定します。	はい	「パラメータグループを使用する」
log_output パラメータが table に設定されている	log_output が TABLE に設定されている場合、log_output が FILE に設定されている場合よりも多くのストレージが使用されます。ストレージサイズの制限に達しないように、パラメーターを FILE に設定することをお勧めします。	DB パラメータグループの log_output パラメータを FILE に設定します。	いいえ	Aurora MySQL データベースのログファイル

型	説明	推奨事項	ダウンタイムが必要	追加情報
synchronous_commit パラメータがオフになっている	synchronous_commit パラメータを無効にすると、データベースのクラッシュでデータが失われる可能性があります。データベースの耐久性が危険にさらされます。 synchronous_commit パラメータをオンにすることをお勧めします。	DB パラメータグループの synchronous_commit パラメータを有効にします。	はい	AWS データベースブログの「Amazon Aurora PostgreSQL パラメータ: データベースブログの「レプリケーション、セキュリティ、ログ記録」
track_counts パラメータがオフになっている	track_counts パラメータが無効の場合、データベースはデータベースアクティビティ統計を収集しません。自動バキュームでは、これらの統計が正しく機能する必要があります。 track_counts パラメータを 1 に設定することをお勧めします。	track_counts パラメータを 1 に設定します。	いいえ	PostgreSQL のランタイム統計

型	説明	推奨事項	ダウンタイムが必要	追加情報
enable_indexonlyscan パラメータがオフになっている	クエリプランナーまたはオプティマイザーは、インデックスのみのスキャン計画タイプが無効になっている場合は使用できません。 enable_indexonlyscan パラメータ値を 1 に設定することをお勧めします。	enable_indexonlyscan パラメータ値を 1 に設定します。	いいえ	PostgreSQL のプランナーメソッド設定
enable_indexscan パラメータがオフになっている	クエリプランナーまたはオプティマイザーは、インデックスのみのスキャン計画タイプが無効になっている場合は使用できません。 enable_indexscan 値を 1 に設定することをお勧めします。	enable_indexscan パラメータ値を 1 に設定します。	いいえ	PostgreSQL のプランナーメソッド設定

型	説明	推奨事項	ダウンタイムが必要	追加情報
innodb_flush_log_at_trx パラメータがオフになっている	DB インスタンスの innodb_flush_log_at_trx パラメータの値は安全ではありません。このパラメータは、ディスクへのコミット操作の持続性を制御します。 innodb_flush_log_at_trx パラメータを 1 に設定することをお勧めします。	innodb_flush_log_at_trx パラメータ値を 1 に設定します。	いいえ	ログバッファをフラッシュする頻度の設定

型	説明	推奨事項	ダウンタイムが必要	追加情報
innodb_stats_persistent パラメータがオフになっている	DB インスタンスは、InnoDB 統計をディスクに保持するように設定されていません。統計が保存されていない場合は、インスタンスが再起動してテーブルにアクセスするたびに再計算されます。これにより、クエリ実行プランにばらつきが生じます。このグローバルパラメータの値はテーブルレベルで変更できます。 innodb_stats_persistent パラメータ値を ON に設定することをお勧めします。	innodb_stats_persistent パラメータ値を ON に設定します。	いいえ	「パラメータグループを使用する」

型	説明	推奨事項	ダウンタイムが必要	追加情報
innodb_open_files パラメータが低い	innodb_open_files パラメータは、InnoDB が一度に開くことができるファイル数を制御します。InnoDB は、mysqld の実行時にすべてのログファイルとシステムテーブルスペースファイルを開きます。 お使いの DB インスタンスは、InnoDB が一度に開くことができる最大ファイル数の値が低くなっています。innodb_open_files パラメータを少なくとも 65 に設定することをお勧めします。	innodb_open_files パラメータを最小値の 65 に設定します。	はい	MySQL 用の InnoDB オープンファイル

型	説明	推奨事項	ダウンタイムが必要	追加情報
max_user_connections パラメータが低い	DB インスタンスは、各データベースアカウントの最大同時接続数の値が低くなっています。 max_user_connections パラメータを 5 より大きい数に設定することをお勧めします。	max_user_connections パラメータの値を 5 より大きい数にします。	はい	MySQL のアカウントリソース制限の設定
リードレプリカは書き込み可能モードで開かれている	DB インスタンスには書き込み可能モードのリードレプリカがあり、クライアントからの更新が可能です。 リードレプリカが書き込み可能モードにならないように、read_only パラメータを TrueIfReplica に設定することをお勧めします。	read_only パラメータ値を TrueIfReplica に設定します。	いいえ	「パラメータグループを使用する」

型	説明	推奨事項	ダウンタイムが必要	追加情報
innodb_default_row_format パラメータ設定が安全ではない	DB インスタンスで既知の問題が発生しました: MySQL バージョン 8.0.26 よりも前のバージョンで、row_format を COMPACT または REDUNDANT に設定して作成されたテーブルは、インデックスが 767 バイトを超えるとアクセスできなくなり、回復できなくなります。 innodb_default_row_format パラメータ値を DYNAMIC に設定することをお勧めします。	innodb_default_row_format パラメータ値を DYNAMIC に設定します。	いいえ	MySQL 8.0.26 での変更

型	説明	推奨事項	ダウンタイムが必要	追加情報
general_logging パラメータがオンになっている	DB インスタンスの一般ログ記録が有効になっています。この設定は、データベースの問題のトラブルシューティングに役立ちます。しかし、一般ログ記録を有効にすると、入出力操作の量と割り当てられるストレージ容量が増え、競合やパフォーマンスの低下につながる可能性があります。 一般ログ記録の使用状況の要件を確認してください。general_logging パラメータ値を 0 に設定することをお勧めします。	一般ログ記録の使用状況の要件を確認してください。必須ではない場合は、general_logging パラメーターの値を 0 に設定することをお勧めします。	いいえ	Aurora MySQL データベースログの概要

型	説明	推奨事項	ダウンタイムが必要	追加情報
読み取りワークロードのプロビジョニングが不十分な DB クラスター	クラスター内のライター DB インスタンスと同じインスタンスクラスとサイズを持つリーダー DB インスタンスを DB クラスターに追加することをお勧めします。現在の設定では、読み取り操作が主な原因となり、データベースの負荷が継続的に高くなっている DB インスタンスが 1 つあります。クラスターに別の DB インスタンスを追加し、読み取りワークロードを DB クラスターの読み取り専用エンドポイントに送信することで、これらの操作を分散します。	リーダー DB インスタンスをクラスターに追加します。	いいえ	DB クラスターに Aurora レプリカを追加する Aurora DB クラスターのパフォーマンスとスケーリングの管理 「Amazon RDS の価格設定」

型	説明	推奨事項	ダウンタイムが必要	追加情報
システムメモリ容量のプロビジョニングが不十分な RDS インスタンス	メモリの使用量を減らすか、メモリの割り当て量の多い DB インスタンスタイプを使用するようにクエリを調整することをお勧めします。インスタンスのメモリが不足すると、データベースのパフォーマンスに影響を及ぼします。	メモリ容量のより高い DB インスタンスを使用する	はい	AWS データベースブログの「Amazon RDS インスタンスの垂直スケーリングと水平スケーリング」 Amazon RDS インスタンスタイプ 「Amazon RDS の価格設定」
システム CPU 容量のプロビジョニングが不十分な RDS インスタンス	より少ないメモリを使用するようにクエリを調整するか、v CPU の割り当て量がより多い DB インスタンスを使用するように DB インスタンスを変更することをお勧めします。DB インスタンスの CPU が少なくなると、データベースのパフォーマンスが低下する可能性があります。	CPU 容量がより多い DB インスタンスを使用する	はい	AWS データベースブログの「Amazon RDS インスタンスの垂直スケーリングと水平スケーリング」 Amazon RDS インスタンスタイプ 「Amazon RDS の価格設定」

型	説明	推奨事項	ダウンタイムが必要	追加情報
RDS リソースは接続プールを正しく利用していません。	Amazon RDS Proxy を有効にして、既存のデータベース接続を効率的にプールして共有することをお勧めします。データベースで既にプロキシを使用している場合は、複数の DB インスタンス間の接続プールと負荷分散を改善するようにプロキシを正しく設定します。RDS Proxy は、接続の枯渇やダウンタイムのリスクを軽減すると同時に、可用性とスケーラビリティを向上させるのに役立ちます。	RDS プロキシを有効にするか、既存のプロキシ設定を変更する	いいえ	AWS データベースブログの「Amazon RDS インスタンスの垂直スケーリングと水平スケーリング」 Amazon RDS Proxy for Aurora の使用 Amazon RDS Proxy の料金

Amazon RDS コンソールを使用して、データベースリソースに関する Amazon Aurora の推奨事項を表示できます。DB クラスターの場合、DB クラスターとそのインスタンスに関する推奨事項が表示されます。

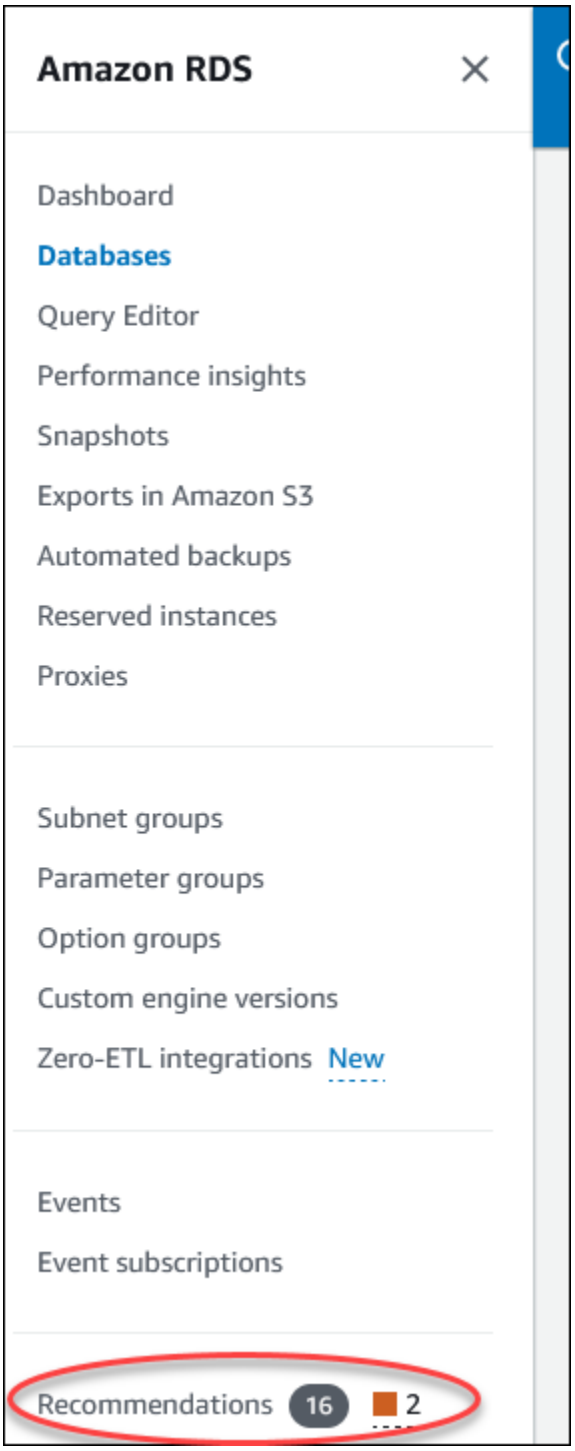
コンソール

Amazon Aurora の推奨事項を表示するには

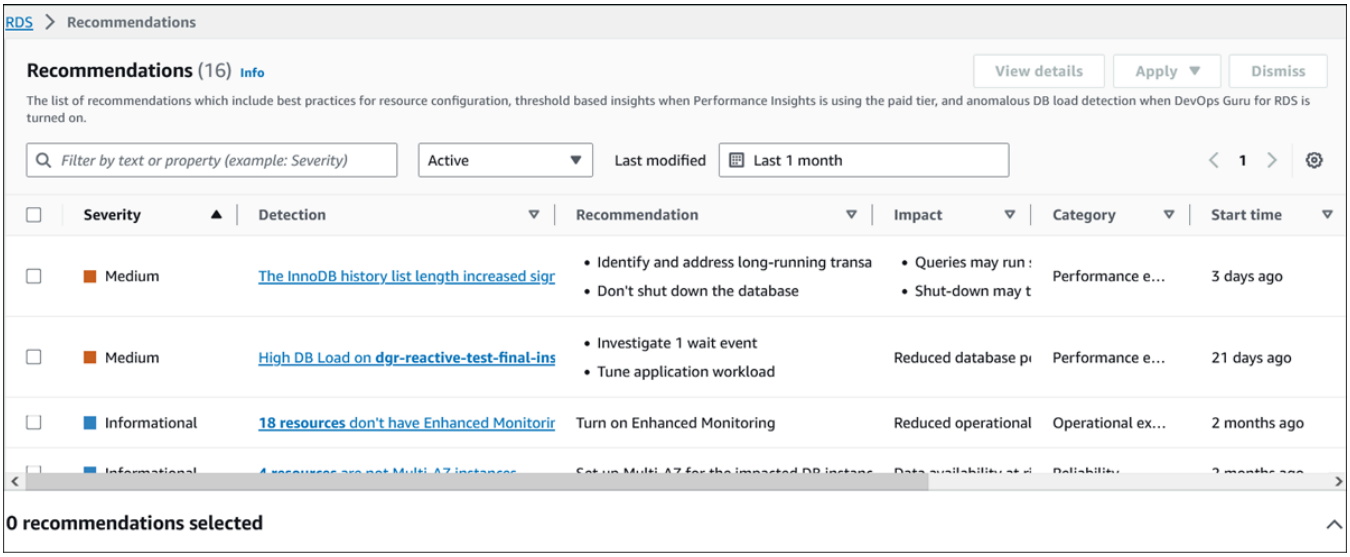
1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。

2. ナビゲーションペインで、次のいずれかを実行します。

   - [レコメンデーション] を選択します。リソースに対するアクティブな推奨事項の数と、前の月に生成された重要度が最も高い推奨事項の数は、[レコメンデーション] の横に表示されます。重要度ごとにアクティブな推奨事項の数を確認するには、最も重要度が高い番号を選択します。

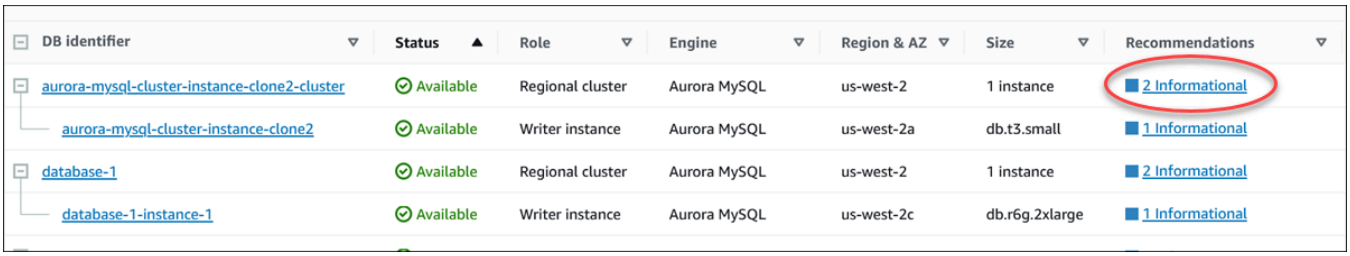


デフォルトでは、[レコメンデーション] ページには前の月の新しい推奨事項のリストが表示されます。Amazon Aurora は、アカウント内のすべてのリソースの推奨事項を提供し、重要度でソートします。

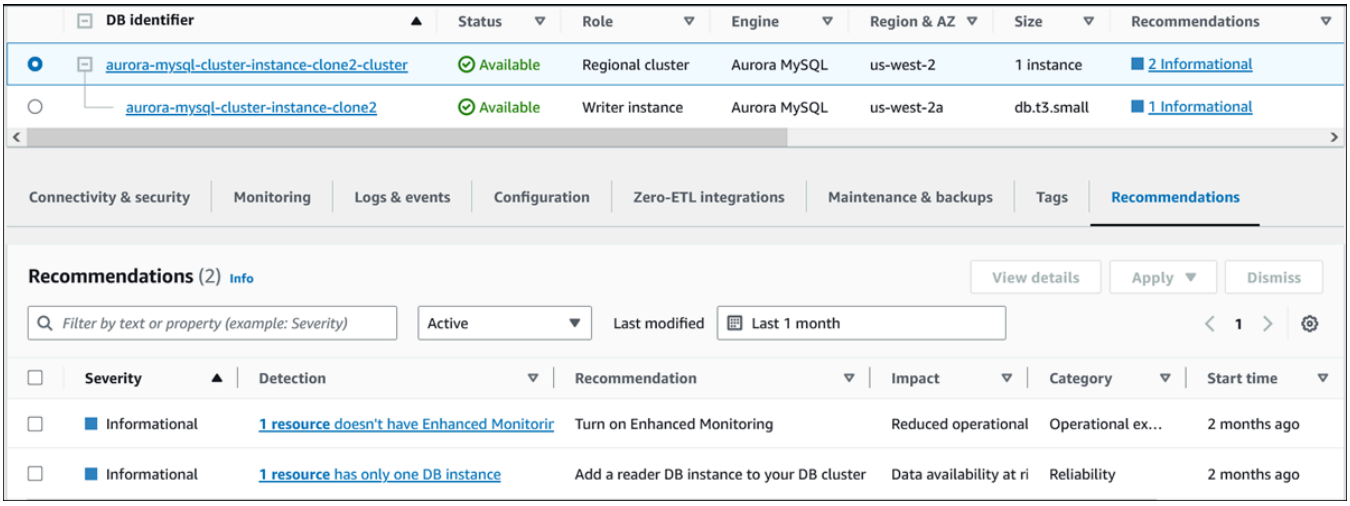


推奨事項を選択すると、ページの下部に、影響を受けるリソースと推奨事項の適用方法の詳細を含むセクションが表示されます。

- [データベース] ページで、リソースの [レコメンデーション] を選択します。



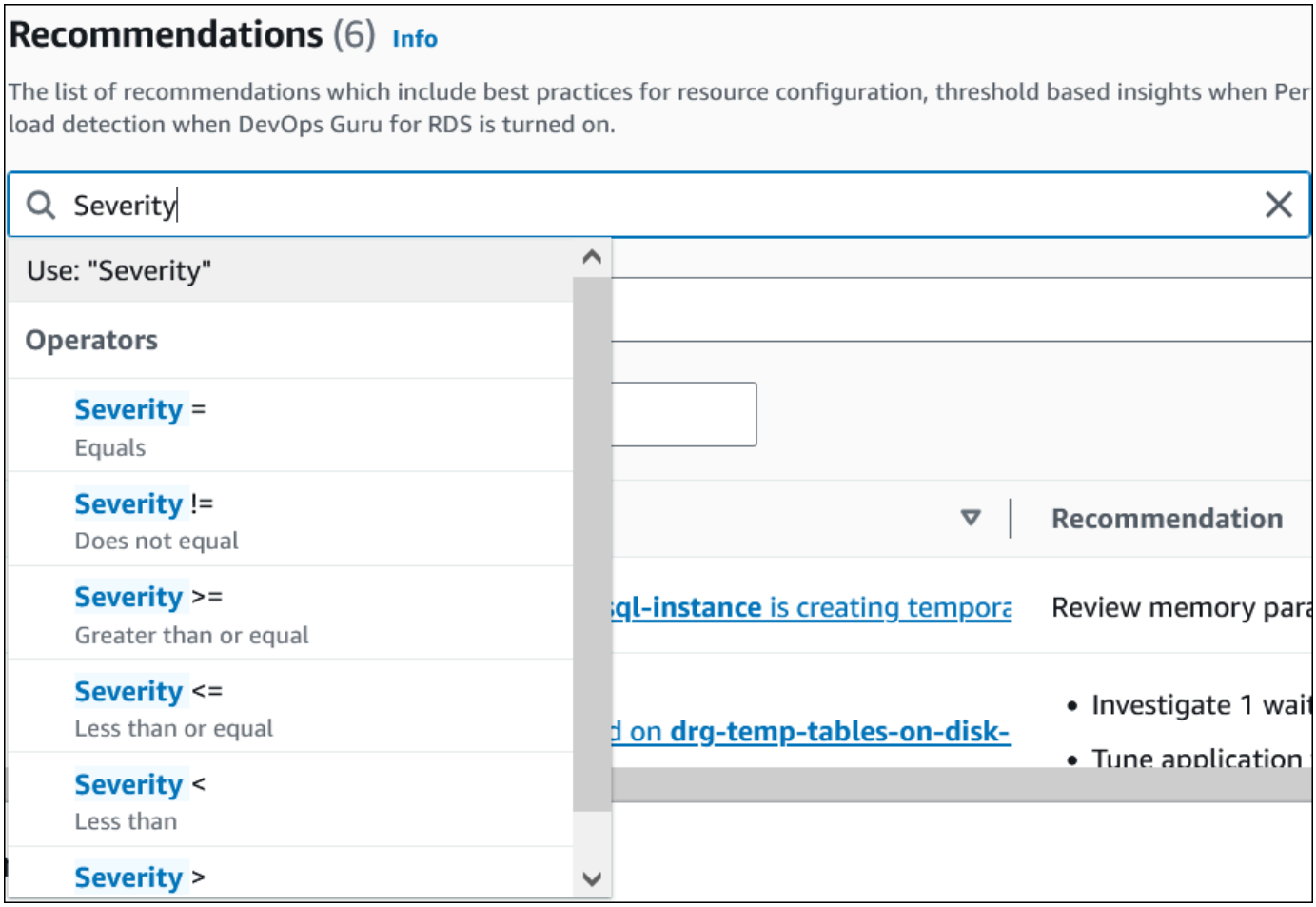
[レコメンデーション] タブには、選択したリソースの推奨事項とその詳細が表示されます。



推奨事項には以下の詳細があります。

- [重要度] — 問題の意味するレベル。重要度レベルは、[高]、[中]、[低]、および [情報] です。
- [検出] — 影響を受けるリソースの数と問題の簡単な説明。このリンクを選択すると、推奨事項と分析の詳細が表示されます。
- [レコメンデーション] — 適用する推奨事項アクションの簡単な説明。
- [影響] — 推奨事項が適用されない場合に発生する可能性のある影響の簡単な説明。
- [カテゴリ] – 推奨事項のタイプ。カテゴリは、[パフォーマンス効率]、[セキュリティ]、[信頼性]、[コスト最適化]、[運用上の優秀性]、[持続可能性]です。
- [ステータス] – ターゲットエンジンの推奨事項のステータス 指定できるステータスは、[すべて]、[アクティブ]、[却下済み]、[解決済み]、[保留中] です。
- [開始時刻] – 問題が開始された時刻。例えば、18 時間前と指定します。
- [最終更新日] – [重要度] の変更により推奨事項がシステムによって最後に更新された時刻、または推奨事項に応答した時刻。例えば、10 時間前と指定します。
- [終了時刻] – 問題が終了した時刻。時間には、継続中の問題は表示されません。
- [リソース識別子] – 1 つ以上のリソースの名前。

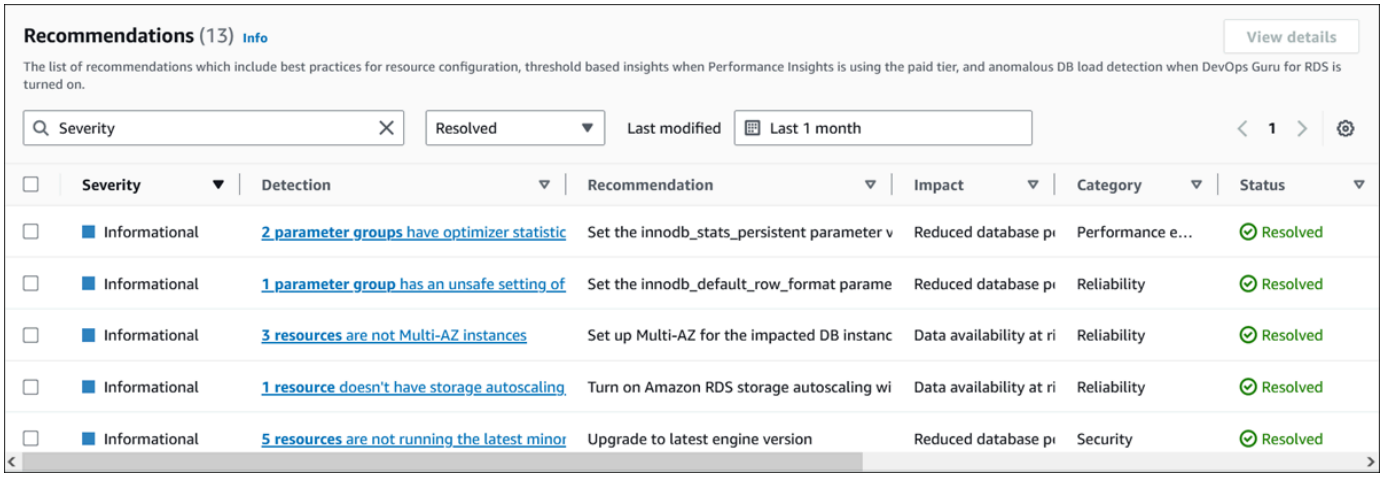
3. (オプション) フィールドで [重要度] または [カテゴリ] 演算子を選択して、推奨事項のリストをフィルタリングします。



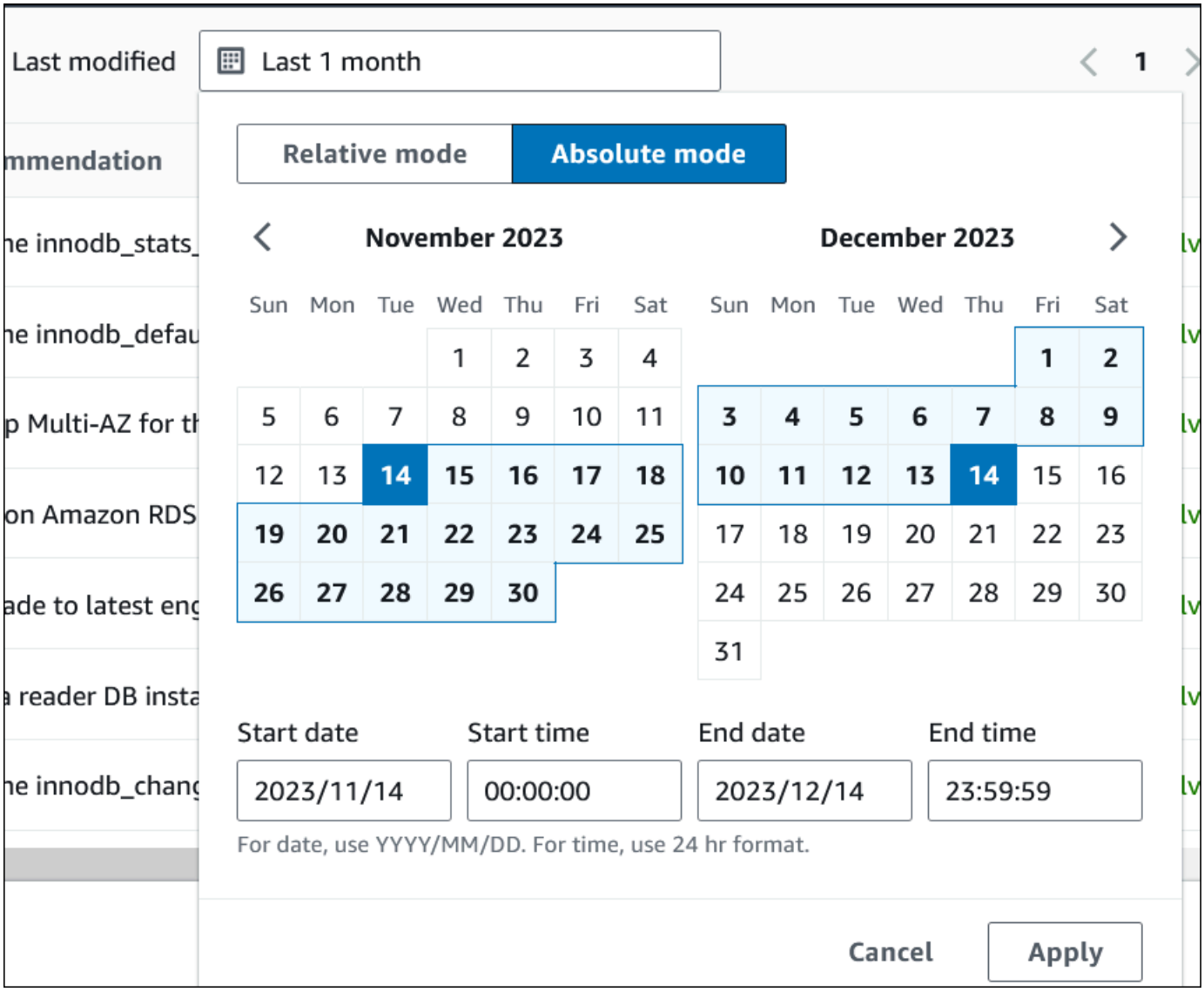
選択したオペレーションの推奨事項が表示されます。

4. (オプション) 次のいずれかの推奨事項ステータスを選択します。

   - [アクティブ] (デフォルト) - 次のメンテナンスウィンドウで適用、スケジュール設定できるか、または却下できる現在の推奨事項が表示されます。
   - [すべて] – 現在のステータスのすべての推奨事項を表示します。
   - [却下済み] – 却下された推奨事項を表示します。
   - [解決済み] — 解決済みの推奨事項を表示します。
   - [保留中] — 推奨アクションが進行中であるか、次のメンテナンスウィンドウにスケジュールされている推奨事項を表示します。



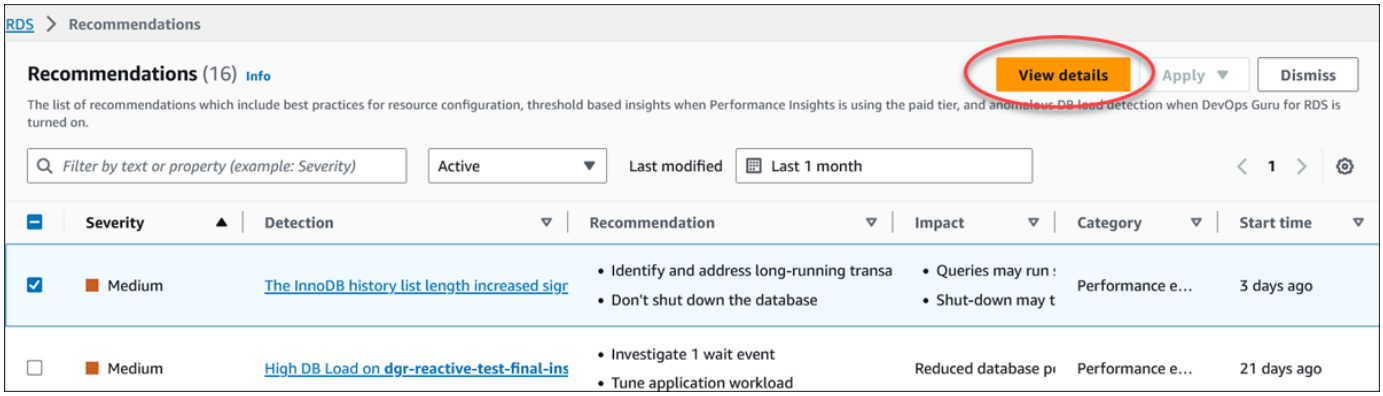
5. (オプション) [最終更新日] の [相対モード] または [絶対モード] を選択して期間を変更します。[レコメンデーション] ページには、期間中に生成された推奨事項が表示されます。デフォルトの期間は過去 1 か月です。[絶対モード] では、期間を選択するか、[開始日] フィールドと [終了日] フィールドに時刻を入力できます。



設定された期間表示の推奨事項。

範囲を [すべて] に設定することで、アカウント内のリソースに関するすべての推奨事項を表示できます。

6. (オプション) 右側の [詳細設定] を選択して、表示する詳細をカスタマイズします。ページサイズを選択し、テキストの行を折り返して、列を許可または非表示にすることができます。
7. (オプション) 推奨事項を選択し、[詳細を表示] を選択します。



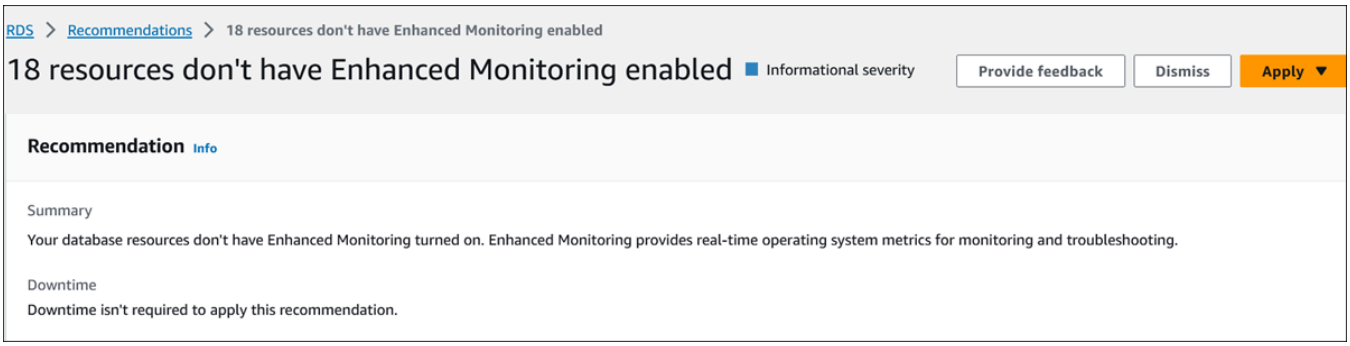
推奨事項の詳細ページが表示されます。タイトルには、問題が検出されたリソースの総数と重要度が表示されます。

異常ベースの事後推奨事項の詳細ページにあるコンポーネントについては、Amazon DevOps Guru ユーザーガイド 「事後対応型異常を表示する」を参照してください。

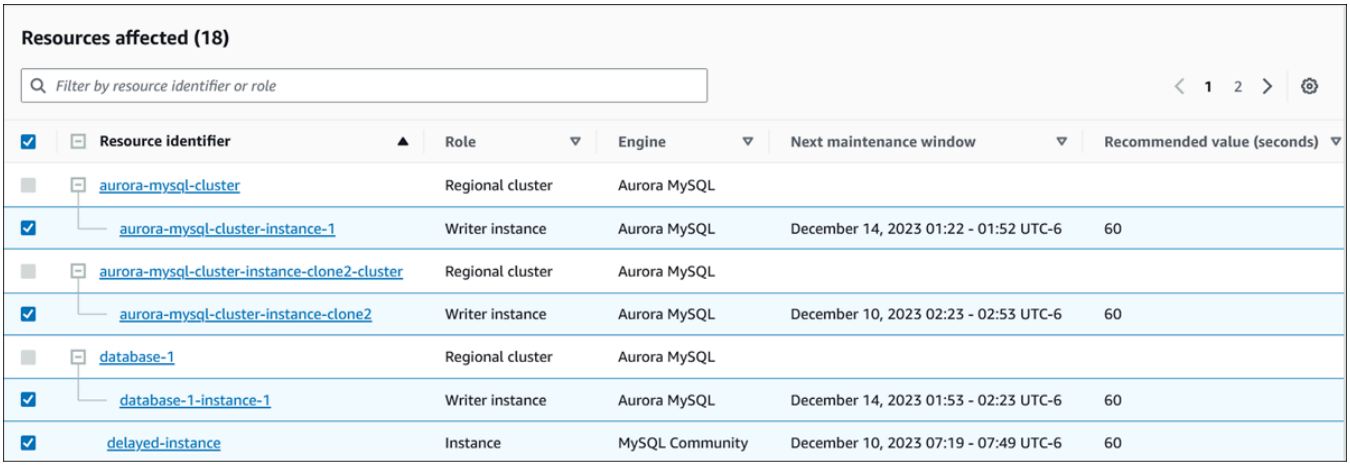
しきい値ベースのプロアクティブ推奨事項の詳細ページのコンポーネントについては、「」を参照してくださいPerformance Insights の事前対応型推奨事項の表示。

その他の自動推奨事項では、推奨事項の詳細ページに次のコンポーネントが表示されます。

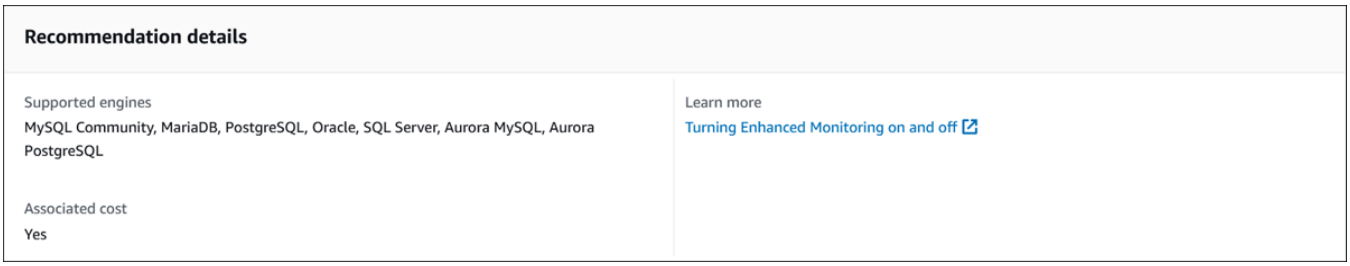
- 推奨事項 — 推奨事項の概要と、推奨事項を適用するためにダウンタイムが必要かどうか。



- [影響を受けるリソース] — 影響を受けるリソースの詳細。



- 推奨事項の詳細 – サポートされるエンジン情報、推奨事項を適用するために必要な関連コスト、および詳細情報を確認するためのドキュメントリンク。



CLI

DB インスタンスまたは DB クラスターの Amazon RDS の推奨事項を表示するには、AWS CLI で次のコマンドを使用します。

```
aws rds describe-db-recommendations
```

RDS API

Amazon RDS API を使用して Amazon RDS の推奨事項を表示するには、DescribeDBRecommendations オペレーションを使用します。

Amazon Aurora 推奨事項への対応

Aurora 推奨事項のリストから、次のことができます。

- 設定ベースの推奨事項をすぐに適用するか、次のメンテナンスウィンドウまで延期します。
- 1 つまたは複数の推奨事項を却下します。

- 却下された推奨事項をアクティブな推奨事項に移動します。

Amazon Aurora 推奨事項の適用

Amazon RDS コンソールを使用して、詳細ページで設定ベースの推奨事項または影響を受けるリソースを選択し、すぐに推奨事項を適用するか、次のメンテナンスウィンドウにスケジュールします。(これらの変更を有効にするには、クラスターの再起動が必要な場合があります。) いくつかの DB パラメータグループの推奨事項については、リソースの再起動が必要になる場合があります。

しきい値ベースの事前対応型推奨事項または異常ベースの事後対応型推奨事項には適用オプションがないため、追加のレビューが必要になる場合があります。

コンソール

設定ベースの推奨事項を適用するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、次のいずれかを実行します。

   - [レコメンデーション] を選択します。

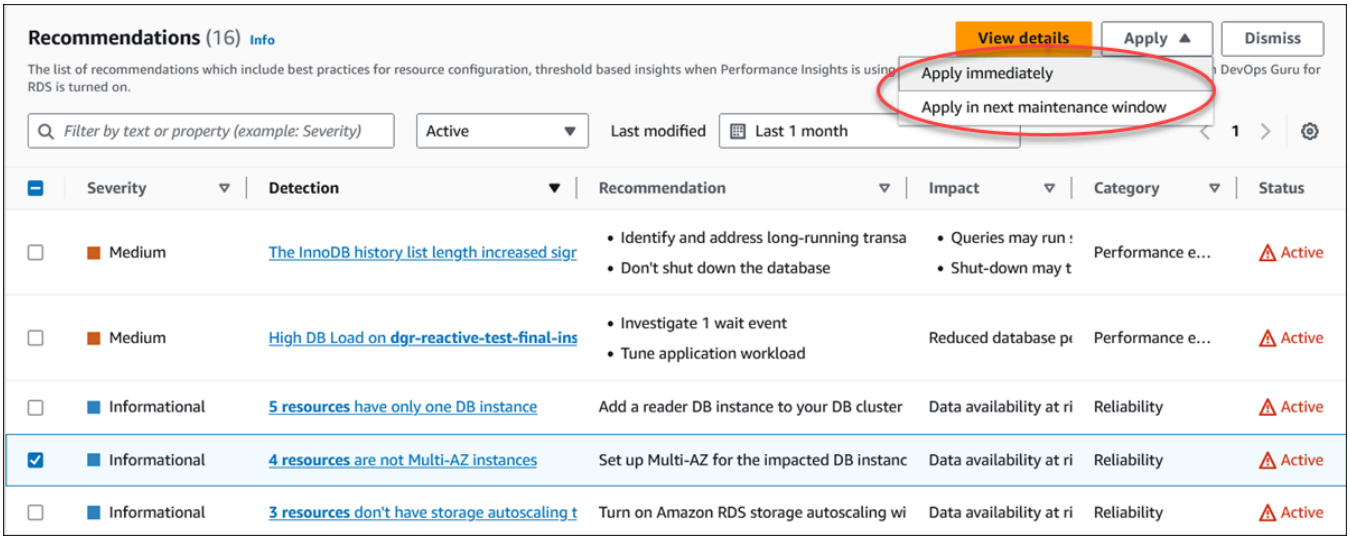
     [レコメンデーション] ページには、すべての推奨事項のリストが表示されます。
   - [データベース] を選択し、データベースページでリソースの [レコメンデーション] を選択します。

     詳細は、選択した推奨事項の [レコメンデーション] タブに表示されます。
   - [レコメンデーション] ページでアクティブな推奨事項の [検出]、または [データベース] ページの [レコメンデーション] タブを選択します。

     推奨事項の詳細ページが表示されます。
3. 推奨事項、または推奨事項の詳細ページで影響を受ける 1 つ以上のリソースを選択し、次のいずれかを実行します。

   - [適用] を選択し、[今すぐ適用] を選択して、すぐに推奨事項を適用します。
   - [適用] を選択し、次に [次のメンテナンスウィンドウで適用] を選択して、次のメンテナンスウィンドウ中にスケジュール設定します。

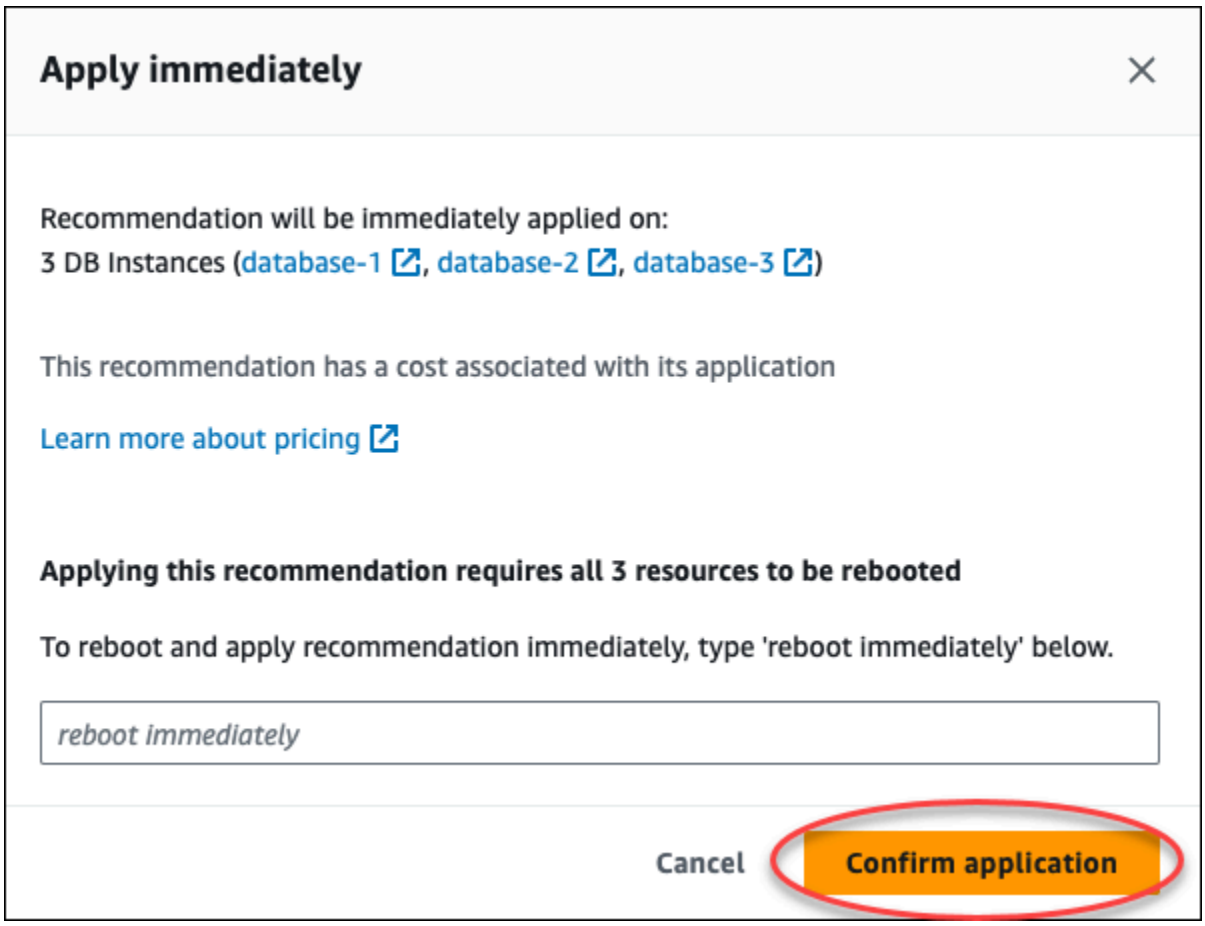
     選択した推奨事項ステータスは、次のメンテナンスウィンドウまで保留に更新されます。



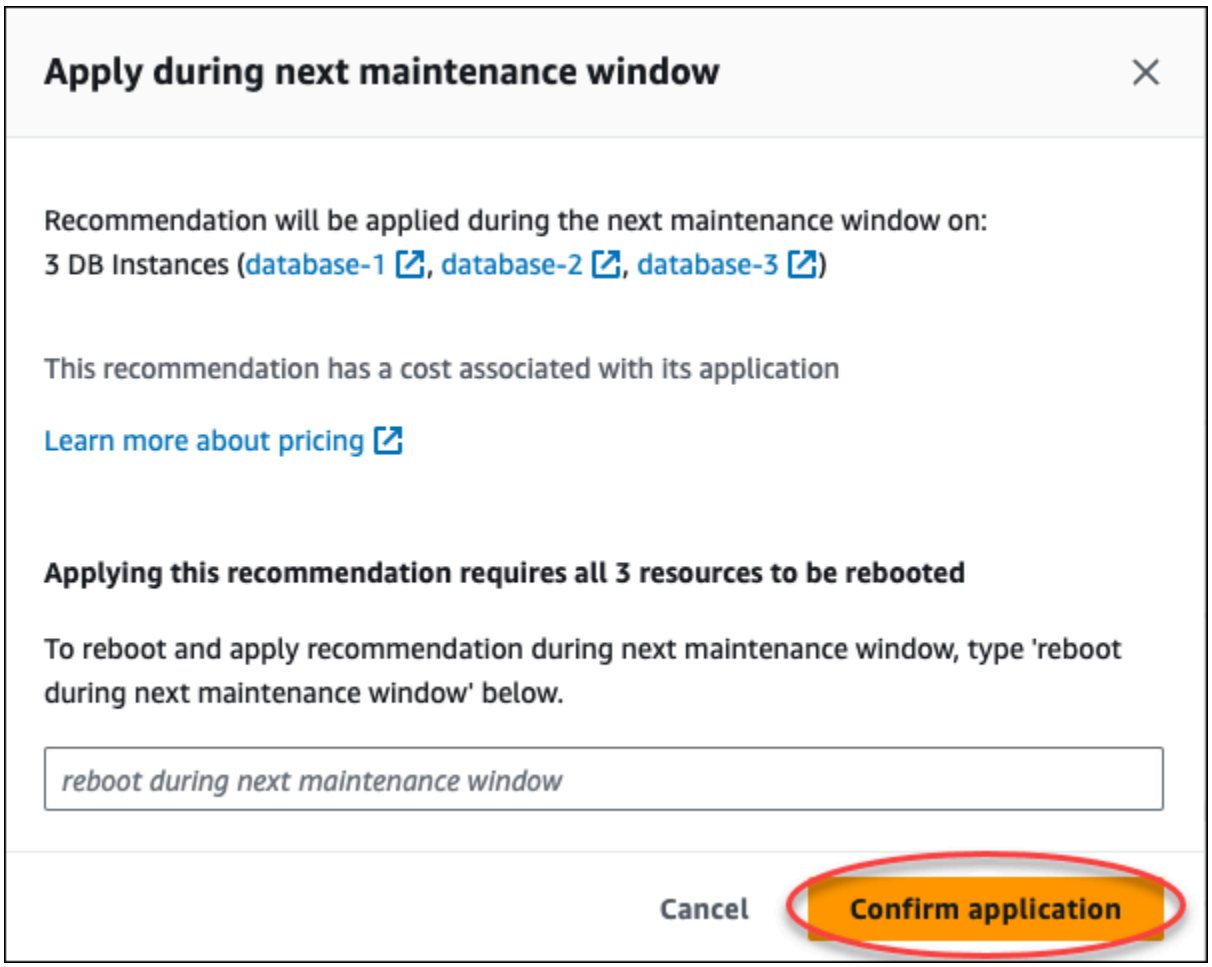
確認ウィンドウが表示されます。

4. [アプリケーションの確認] を選択して、推奨事項を適用します。このウィンドウは、変更を有効にするためにリソースを自動再起動する必要があるか手動で再起動する必要があるかを確認します。

   次の例は、推奨事項をすぐに適用するための確認ウィンドウを示しています。

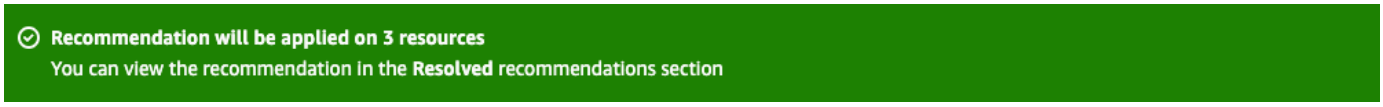


次の例は、次のメンテナンスウィンドウで推奨事項の適用をスケジュールする確認ウィンドウを示しています。



バナーには、適用された推奨事項が成功または失敗したときにメッセージが表示されます。

次の例は、成功メッセージを含むバナーを示しています。



次の例は、失敗メッセージを含むバナーを示しています。

RDS API

Amazon RDS API を使用して設定ベースの Aurora 推奨事項を適用するには

1. DescribeDBRecommendations オペレーションを使用します。出力の RecommendedActions には、1 つ以上の推奨アクションを含めることができます。
2. ステップ 1 の推奨アクションごとに RecommendedAction オブジェクトを使用します。出力には Operation と Parameters が含まれます。

   次の例は、1 つの推奨アクションを含む出力を示しています。

```
   "RecommendedActions": [
       {
           "ActionId": "0b19ed15-840f-463c-a200-b10af1b552e3",
           "Title": "Turn on auto backup", // localized
           "Description": "Turn on auto backup for my-mysql-instance-1", //
localized
           "Operation": "ModifyDbInstance",
           "Parameters": [
               {
                   "Key": "DbInstanceIdentifier",
                   "Value": "my-mysql-instance-1"
               },
               {
                   "Key": "BackupRetentionPeriod",
                   "Value": "7"
               }
           ],
           "ApplyModes": ["immediately", "next-maintenance-window"],
           "Status": "applied"
       },
       ... // several others
   ],
```

3. ステップ 2 の出力からの推奨アクションごとに operation を使用し、Parameters 値を入力します。
4. ステップ 2 の オペレーションが成功したら、ModifyDBRecommendation オペレーションを使用して推奨事項のステータスを変更します。

Amazon Aurora の推奨事項の却下

1 つまたは複数の推奨事項を却下できます。

コンソール

1 つまたは複数の推奨事項を却下するには

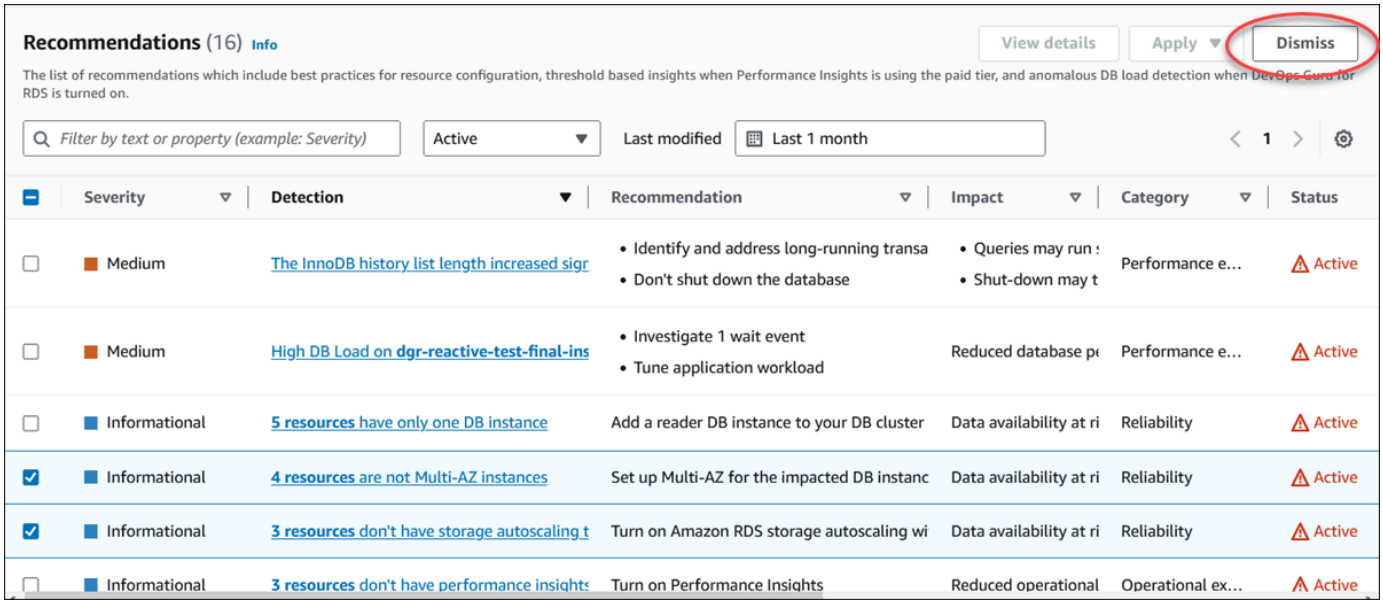
1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、次のいずれかを実行します。
   - [レコメンデーション] を選択します。

     [レコメンデーション] ページには、すべての推奨事項のリストが表示されます。
   - [データベース] を選択し、データベースページでリソースの [レコメンデーション] を選択します。

     詳細は、選択した推奨事項の [レコメンデーション] タブに表示されます。
   - [レコメンデーション] ページでアクティブな推奨事項の [検出]、または [データベース] ページの [レコメンデーション] タブを選択します。

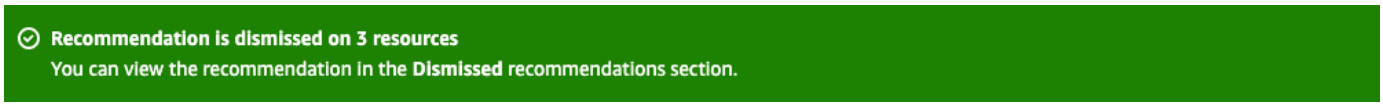
     推奨事項の詳細ページには、影響を受けるリソースのリストが表示されます。
3. 1 つ以上の推奨事項を選択するか、推奨事項の詳細ページで影響を受けるリソースを 1 つ以上選択し、[閉じる] を選択します。

   次の例は、却下するアクティブな推奨事項が複数選択されている [レコメンデーション] ページを示しています。



選択した 1 つ以上の推奨事項が却下されると、バナーにメッセージが表示されます。

次の例は、成功メッセージを含むバナーを示しています。



次の例は、失敗メッセージを含むバナーを示しています。

CLI

AWS CLI を使用して Aurora 推奨事項を却下するには

1. `aws rds describe-db-recommendations --filters "Name=status,Values=active"` コマンドを実行します。

   出力には、`active` ステータスの推奨事項のリストが表示されます。
2. ステップ 1 で却下する推奨事項の `recommendationId` を見つけます。
3. ステップ 2 の `recommendationId` で `>aws rds modify-db-recommendation --status dismissed --recommendationId <ID>` コマンドを実行して、推奨事項を却下します。

RDS API

Amazon RDS API を使用して Aurora 推奨事項を却下するには、ModifyDBRecommendation オペレーションを使用します。

却下された Amazon Aurora 推奨事項のアクティブな推奨事項への変更

却下された推奨事項をアクティブな推奨事項に移動します。

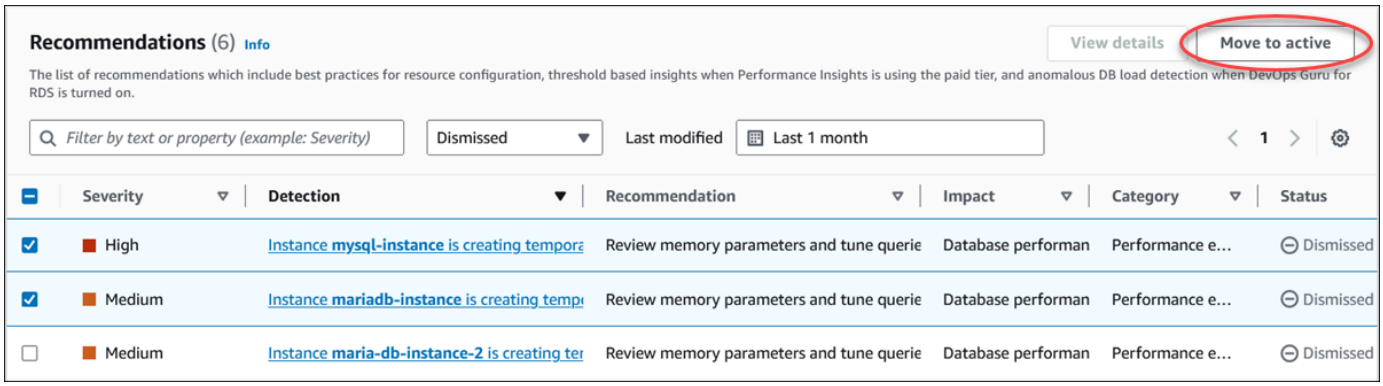
コンソール

却下された推奨事項をアクティブな推奨事項に移動するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、次のいずれかを実行します。
   - [レコメンデーション] を選択します。

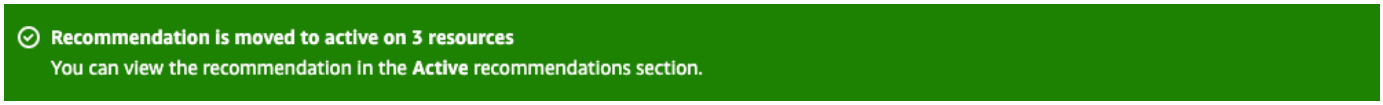
     [レコメンデーション] ページには、アカウント内のすべてのリソースの重要度でソートされた推奨事項のリストが表示されます。
   - [データベース] を選択し、データベースページでリソースの [レコメンデーション] を選択します。

     [レコメンデーション] タブには、選択したリソースの推奨事項とその詳細が表示されます。
3. リストから却下された推奨事項を 1 つ以上選択し、[アクティブに移動] を選択します。



選択した推奨事項を却下からアクティブなステータスに移行すると、バナーに成功または失敗のメッセージが表示されます。

次の例は、成功メッセージを含むバナーを示しています。



次の例は、失敗メッセージを含むバナーを示しています。

CLI

AWS CLI を使用して却下された Aurora 推奨事項をアクティブな推奨事項に変更するには

1. `aws rds describe-db-recommendations --filters "Name=status,Values=dismissed"` コマンドを実行します。

   出力には、`dismissed` ステータスの推奨事項のリストが表示されます。
2. ステップ 1 でステータスを変更する推奨事項の `recommendationId` を見つけます。
3. ステップ 2 の `recommendationId` で `>aws rds modify-db-recommendation --status active --recommendationId <ID>` コマンドを実行して、アクティブな推奨事項を変更します。

RDS API

Amazon RDS API を使用して却下された Aurora 推奨事項を変更するには、ModifyDBRecommendation オペレーションを使用します。

Amazon RDS コンソールでのメトリクスの表示

Amazon RDS は Amazon Cloudwatch と統合し、さまざまな Aurora DB クラスターメトリクスを RDS コンソールで表示できるようになりました。一部のメトリクスはクラスターレベルで適用され、その他のメトリクスはインスタンスレベルで適用されます。インスタンスレベルおよびクラスターレベルのメトリクスの詳細については、「Amazon Aurora のメトリクスリファレンス」を参照してください。

Aurora DB クラスター については、次のカテゴリのメトリクスがモニタリングされます。

- CloudWatch – RDS コンソールからアクセスできる Aurora の Amazon CloudWatch メトリクスを表示します。これらのメトリクスには、CloudWatch コンソールからアクセスすることもできます。各メトリクスには、特定の期間にわたってモニタリングされたメトリクスを示すグラフが含まれます。CloudWatch メトリクスのリストについては、「Amazon Aurora の Amazon CloudWatch メトリクス」を参照してください。
- [Enhanced monitoring] (拡張モニタリング) – Aurora DB クラスターで、拡張モニタリングがオンにされたときのオペレーティングシステムメトリクスの概要を表示します。RDS は、拡張モニタリングのメトリクスを Amazon CloudWatch Logs アカウントに配信します。各 OS メトリクスには、特定の期間にわたってモニタリングされたメトリクスを示すグラフが含まれます。概要については、「拡張モニタリングを使用した OS メトリクスのモニタリング」を参照してください。拡張モニタリングメトリクスのリストについては、「拡張モニタリングの OS メトリクス」を参照してください。
- [OS Process list] (OS プロセスリスト) – DB クラスターで実行中の各プロセスの詳細を表示します。
- [Performance Insights] – Aurora DB クラスター内の DB インスタンスで Amazon RDS Performance Insights ダッシュボードを開きます。Performance Insights は、クラスターレベルではサポートされていません。Performance Insights の概要については、「Amazon Aurora での Performance Insights を使用したDB 負荷のモニタリング」を参照してください。Performance Insights メトリクスのリストについては、「Performance Insights の Amazon CloudWatch メトリクス」を参照してください。

Amazon RDS の Performance Insights ダッシュボードで、Performance Insights と CloudWatch メトリクスの統合ビューが提供されるようになりました。このビューを使用するには、DBDB クラスターの Performance Insights がオンになっている必要があります。[モニタリング] タブまたは、ナビゲーションペインの [Performance Insights] で新しいモニタリングビューを選択できます。この

ビューを選択する手順については、「Amazon RDS コンソールでの組み合わせたメトリクスの表示」を参照してください。

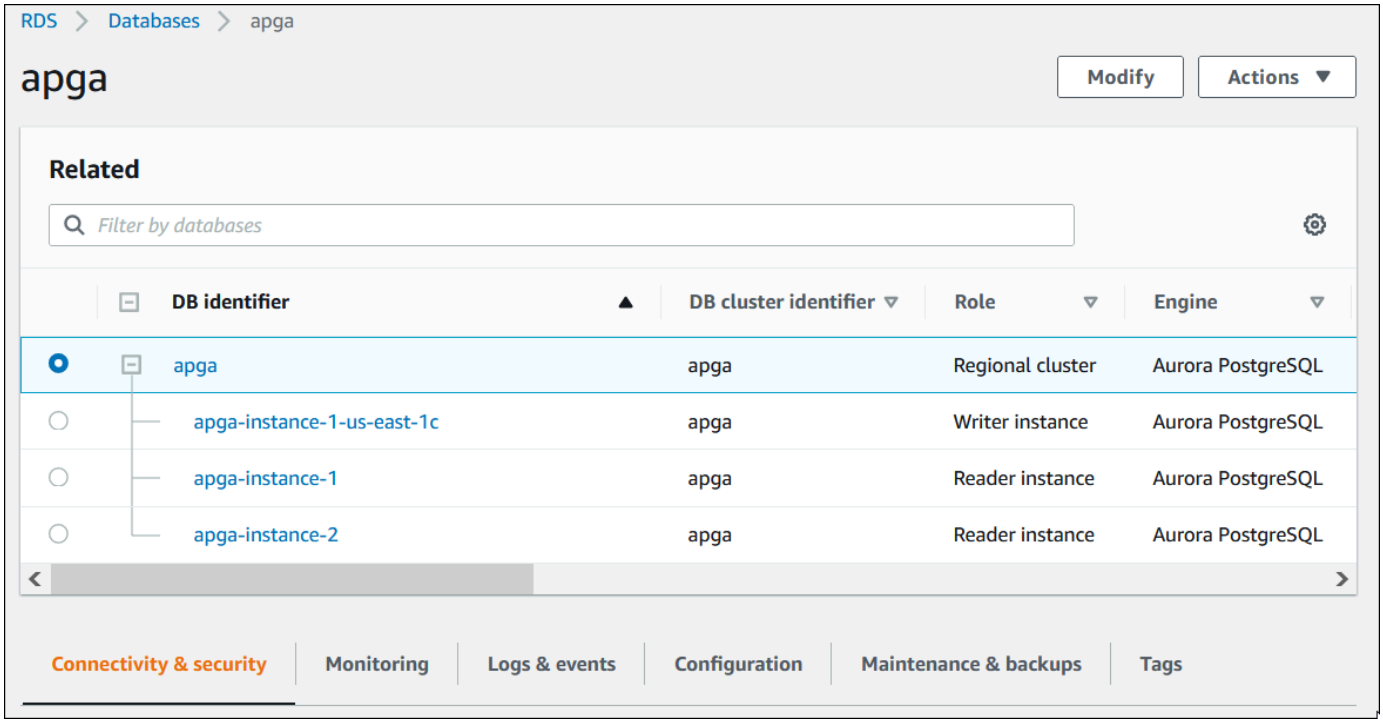
レガシーのモニタリングビューを続行する場合は、この手順を続行してください。

> **Note**
>
> レガシーのモニタリングビューは、2023 年 12 月 15 日に廃止されます。

DB クラスター DB:

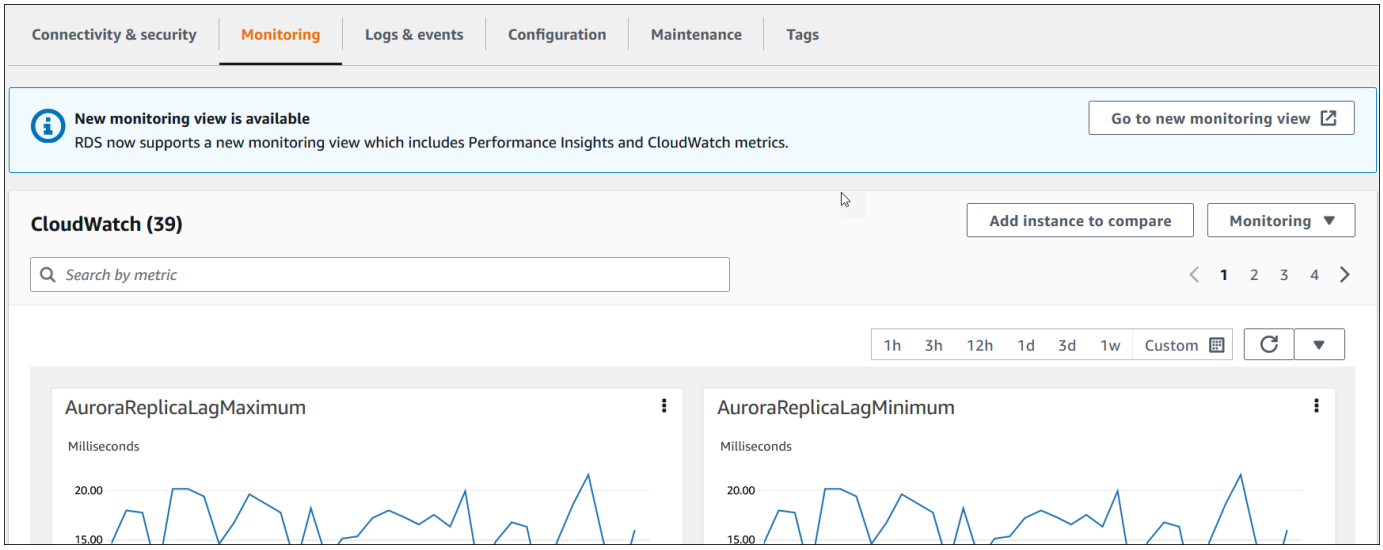
1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. モニタリングする Aurora DB クラスターの名前を選択します。

   [Database] (データベース) ページが表示されます。次の例は、 apga という名前の Amazon Aurora PostgreSQL データベースを示しています。

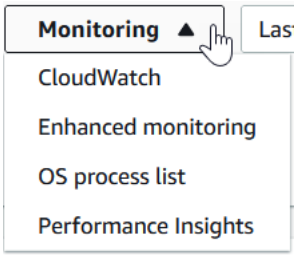


4. 下にスクロールし、[Monitoring] (モニタリング) を選択します。

[Monitoring] (モニタリング) セクションが表示されます。デフォルトでは、CloudWatch メトリクスが表示されます。これらのメトリクスの詳細については、「Amazon Aurora の Amazon CloudWatch メトリクス」を参照してください。

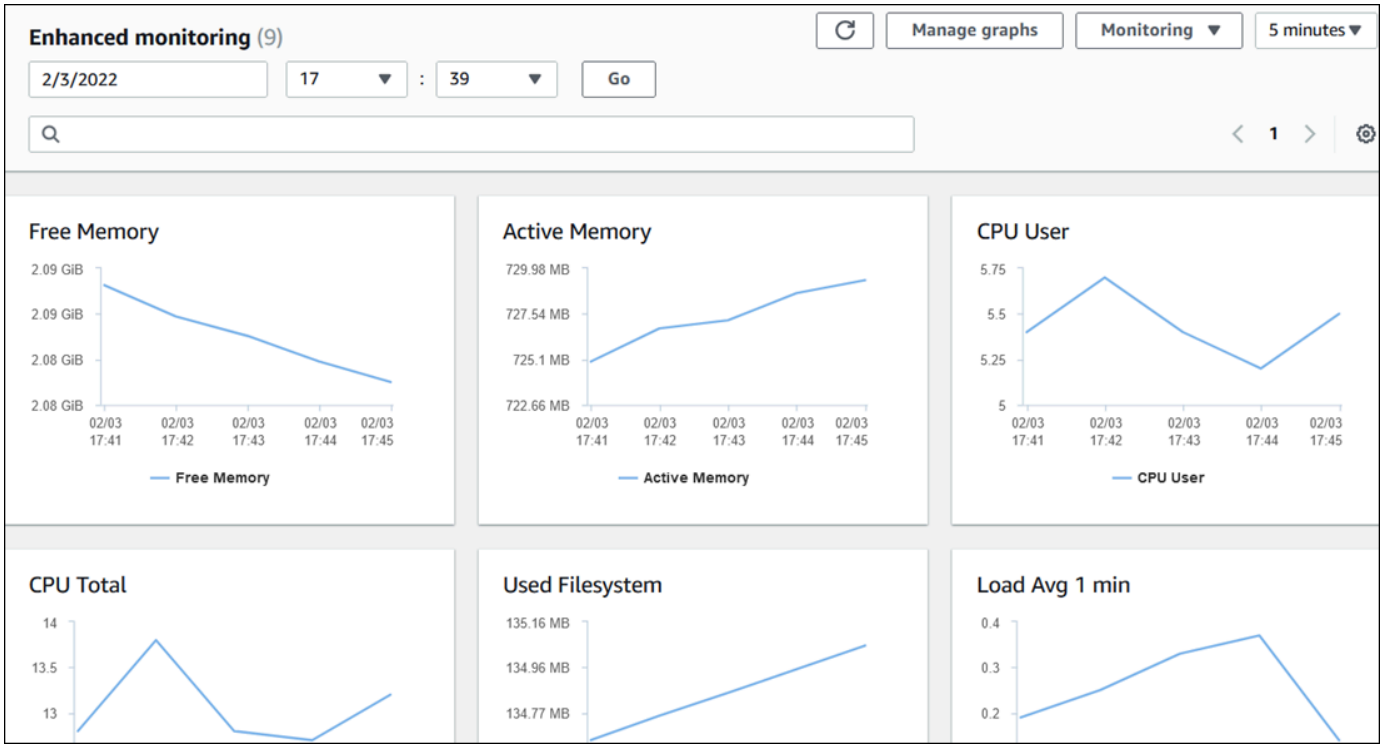


5. [Monitoring] (モニタリング) を選択して、メトリクスのカテゴリを表示します。



6. 表示するメトリクスのカテゴリを選択します。

   次の例は、拡張モニタリングメトリクスを示しています。これらのメトリクスの詳細については、「拡張モニタリングの OS メトリクス」を参照してください。



Tip

グラフで表されるメトリクスの時間範囲を選択するには、時間範囲リストを使用します。

より詳細なビューを表示するには、任意のグラフを選択します。メトリック固有のフィルターをデータに適用することもできます。

Amazon RDS コンソールでの組み合わせたメトリクスの表示

Amazon RDS の Performance Insights ダッシュボードでは、DB インスタンスに対して Performance Insights と CloudWatch メトリクスの統合ビューが提供されるようになりました。事前設定されたダッシュボードを使用するか、カスタムダッシュボードを作成できます。事前設定されたダッシュボードには、データベースエンジンのパフォーマンス問題の診断に役立つ最も一般的に使用されるメトリクスが表示されます。また、分析要件を満たすデータベースエンジンのメトリダッシュボードを作成することもできます。次に、このダッシュボードを AWS アカウント内のそのデータベースエンジンタイプのすべての DB インスタンスに使用します。

[モニタリング] タブまたは、ナビゲーションペインの [Performance Insights] で新しいモニタリングビューを選択できます。[Performance Insights] ページに移動すると、新しいモニタリングビューとレガシービューのいずれかを選択できるオプションが表示されます。選択したオプションはデフォルトビューとして保存されます。

Performance Insights ダッシュボードに組み合わせたメトリクスを表示するには、DB クラスター に対して Performance Insights がオンになっている必要があります。Performance Insights をオンにする方法の詳細については、「Performance Insights の有効化と無効化」を参照してください。

> **Note**
>
> 新しいモニタリングビューを選択することをお勧めします。2023 年 12 月 15 日に廃止されるまで、従来のモニタリングビューを引き続き使用できます。

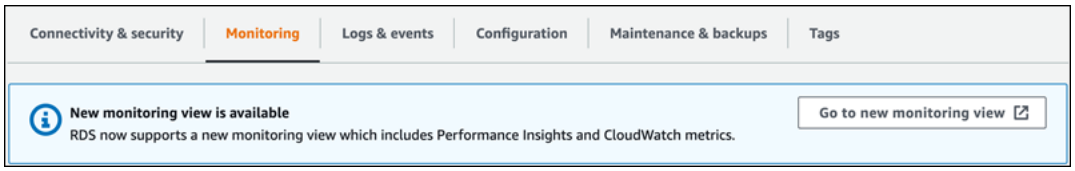
[モニタリング] タブで新しいモニタリングビューを選択する

[モニタリング] タブで新しいモニタリングビューを選択するには:

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. 左のナビゲーションペインの [データベース] を選択します。
3. モニタリングする Aurora DB クラスターの名前を選択します。

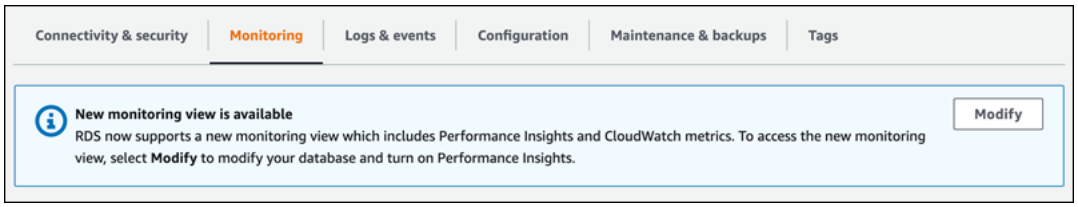
   [Database] (データベース) ページが表示されます。
4. 下にスクロールし、[モニタリング] タブを選択します。

新しいモニタリングビューを選択するオプションを含むバナーが表示されます。次の例は、新しいモニタリングビューを選択するためのバナーを示しています。



5. [新しいモニタリングビューに移動] を選択すると、DB クラスター インスタンスに対して、Performance Insights と CloudWatch メトリクスが表示された Performance Insights ダッシュボードが開きます。
6. (オプション) DB インスタンスの Performance Insights がオフになっている場合、DB インスタンスを変更して Performance Insights をオンにするオプションを含むバナーが表示されます。

   次の例は、[モニタリング] タブの DB インスタンスを変更するためのバナーを示しています。



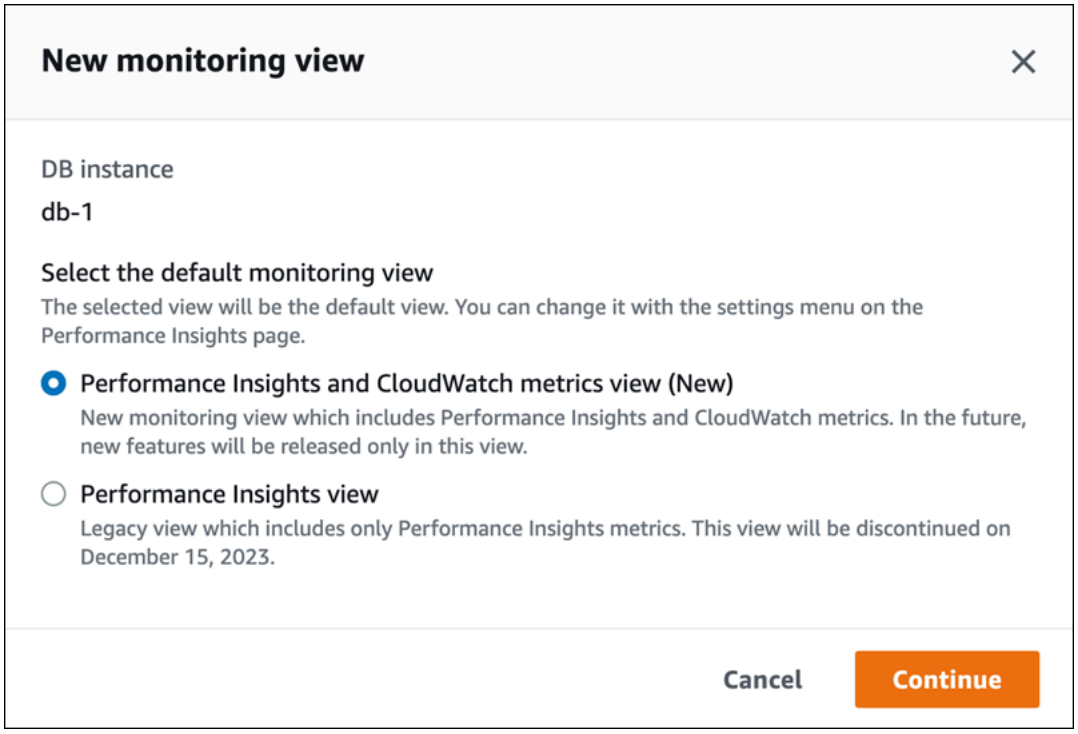
   [変更] を選択して DB インスタンスを変更し、Performance Insights をオンにします。Performance Insights をオンにする方法の詳細については、「Performance Insights の有効化と無効化」を参照してください。

ナビゲーションペインの Performance Insights を使用して新しいモニタリングビューを選択する

ナビゲーションペインの Performance Insights を使用して新しいモニタリングビューを選択するには:

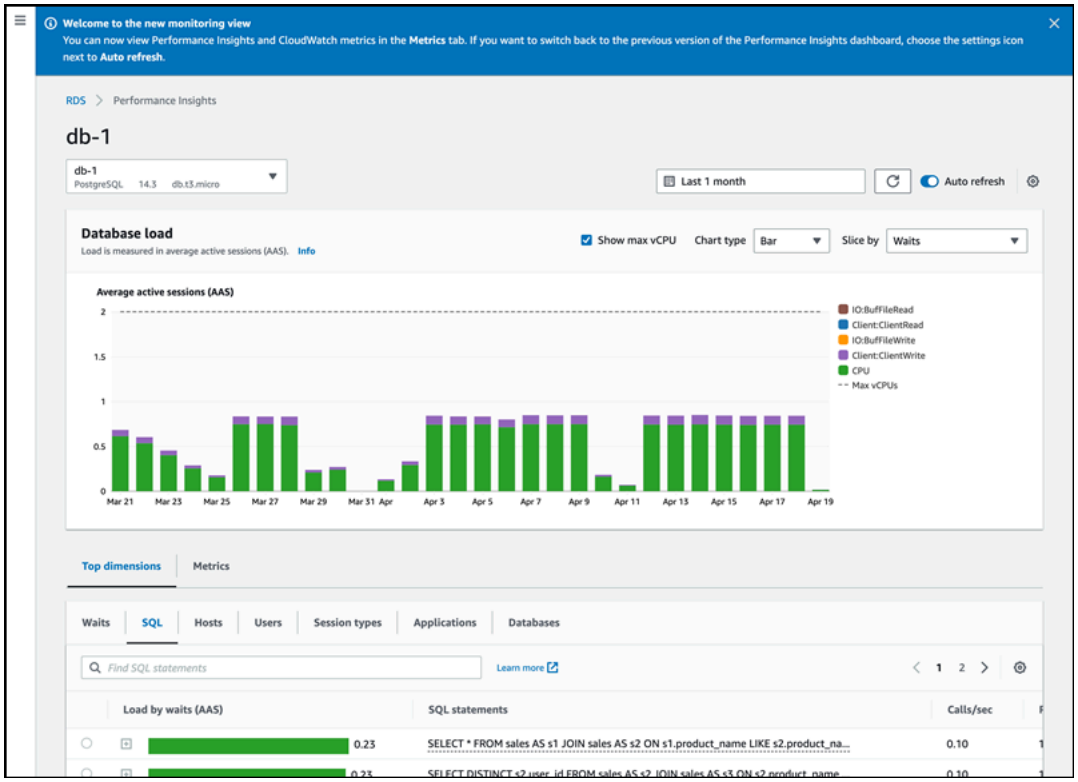
1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択すると、モニタリングビューオプションを含むウィンドウが開きます。

   次の例は、モニタリングビューオプションを示しています。



4. [Performance Insights および CloudWatch メトリクスビュー (新規)] オプションを選択し、[続行] を選択します。

   DB インスタンスに対する Performance Insights と CloudWatch メトリクスの組み合わせが表示された Performance Insights ダッシュボードを表示できるようになりました。次の例は、ダッシュボードの Performance Insights と CloudWatch メトリクスを示しています。



ナビゲーションペインの Performance Insights を使用してレガシービューを選択する

レガシーモニタリングビューを選択すると、DB インスタンスの Performance Insights メトリクスのみを表示できます。

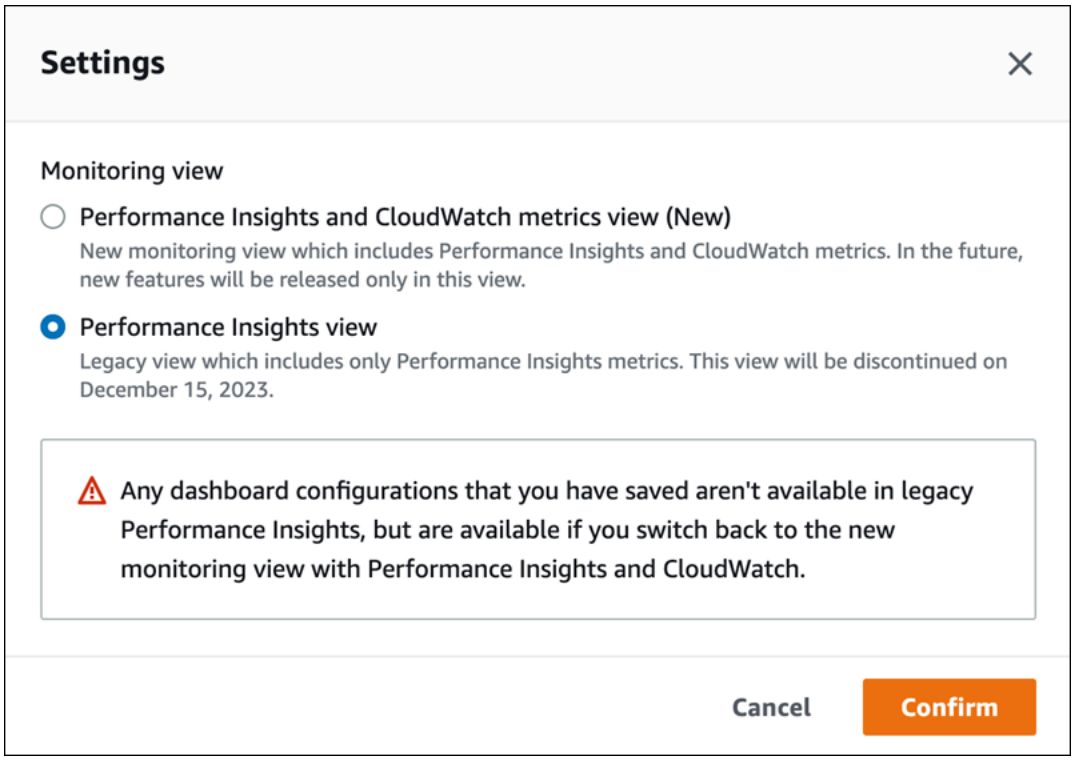
> **Note**
>
> このビューは 2023 年 12 月 15 日に廃止されます。

ナビゲーションペインの Performance Insights を使用してレガシーモニタリングビューを選択するには:

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。
4. Performance Insights ダッシュボードで設定アイコンを選択します。

これにより、従来の Performance Insights ビューを選択するオプションが表示される [設定] ウィンドウが表示されます。

次の例は、レガシーモニタリングビューのオプションが表示されたウィンドウを示しています。



5. [Performance Insights ビュー] オプションを選択し、[続行] を選択します。

   警告メッセージが表示されます。保存したダッシュボード設定は、このビューでは使用できません。

6. [確認] を選択してレガシーの Performance Insights ビューに進みます。

   DB インスタンスに対する Performance Insights メトリクスのみが表示された Performance Insights ダッシュボードを表示できるようになりました。

ナビゲーションペインに Performance Insights が表示されたカスタムダッシュボードの作成

新しいモニタリングビューでは、分析要件を満たすために必要なメトリクスを含むカスタムダッシュボードを作成できます。

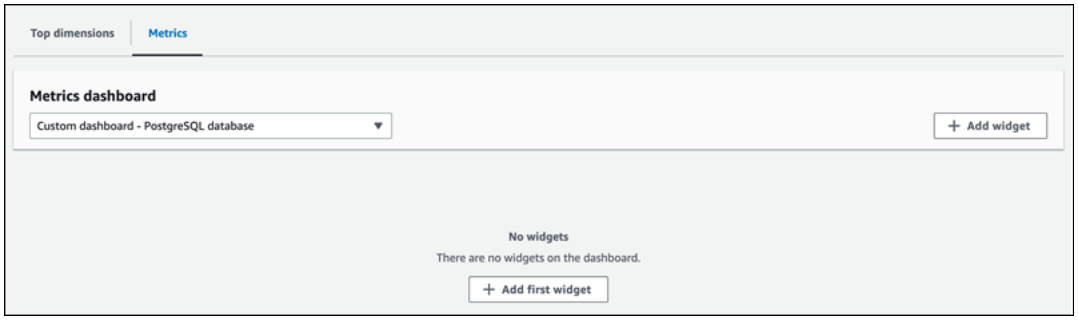
DB インスタンスに対して Performance Insights と CloudWatch メトリクスを選択することで、カスタムダッシュボードを作成できます。次に、このカスタムダッシュボードを AWS アカウント内の同じデータベースエンジンタイプのすべての DB インスタンスに使用します。

> **Note**
>
> カスタマイズされたダッシュボードは、最大 50 のメトリクスをサポートします。

ウィジェット設定メニューを使用して、ダッシュボードを編集または削除したり、ウィジェットウィンドウを移動したり、サイズを変更したりできます。

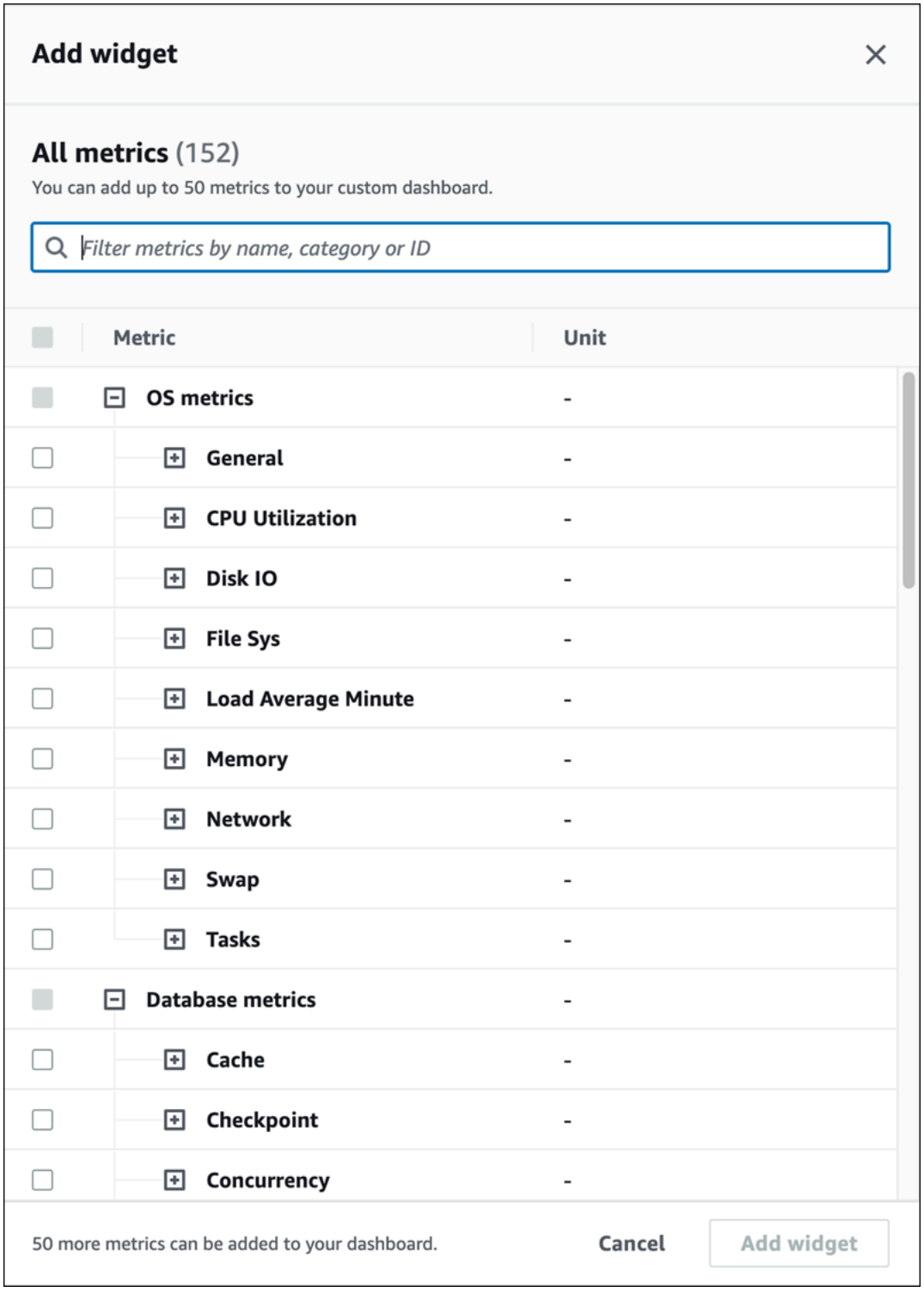
ナビゲーションペインに Performance Insights が表示されたカスタムダッシュボードを作成するには:

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。
4. ウィンドウの [メトリクス] タブが表示されるまで下にスクロールします。
5. ドロップダウンリストから、カスタムダッシュボードを選択します。次の例は、カスタムダッシュボードの作成を示しています。



6. [ウィジェットの追加] を選択して、[ウィジェットの追加] ウィンドウを開きます。使用可能なオペレーティングシステム (OS) メトリクス、データベースメトリクス、および CloudWatch メトリックスをウィンドウで開いて表示できます。

   次の例は、メトリクスが表示された[ウィジェットの追加] ウィンドウを示しています。



7. ダッシュボードで表示するメトリクスを選択してから、[ウィジェットの追加] を選択します。検索フィールドを使用して、特定のメトリクスを検索できます。

   選択したメトリクスがダッシュボードに表示されます。

8. (オプション) ダッシュボードを変更または削除する場合は、ウィジェットの右上にある設定アイコンを選択し、メニューで次のいずれかのアクションを選択します。

   - 編集 — ウィンドウ内のメトリクスリストを変更します。ダッシュボードのメトリクスを選択したら、[ウィジェットの更新] を選択します。
   - 削除 — ウィジェットを削除します。確認ウィンドウで、[削除] を選択します。

ナビゲーションペインの Performance Insights で、事前設定されたダッシュボードを選択する

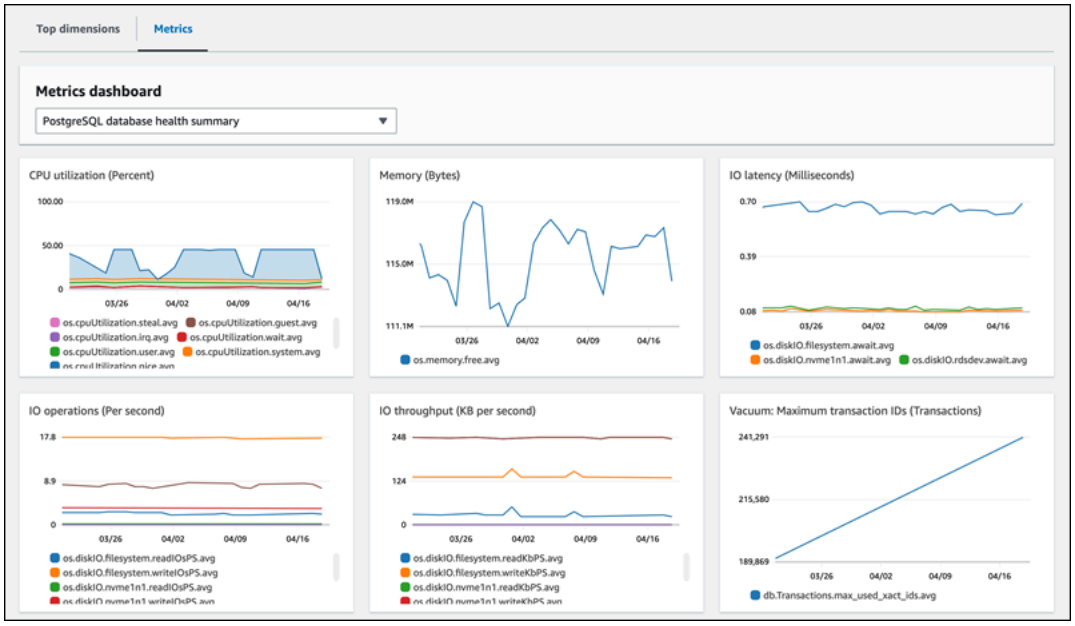
事前設定されたダッシュボードを使用して、最も一般的に使用されるメトリクスを表示できます。このダッシュボードは、データベースエンジンのパフォーマンスの問題を診断し、平均復旧時間を数時間から数分に短縮するのに役立ちます。

> **Note**
>
> このダッシュボードは編集できません。

ナビゲーションペインの Performance Insights で、事前設定されたダッシュボードを選択するには:

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。
4. ウィンドウの [メトリクス] タブが表示されるまで下にスクロールします
5. ドロップダウンリストから、事前設定されたダッシュボードを選択します。

   ダッシュボードで DB インスタンスのメトリクスを表示できます。次の例は、事前設定されたメトリクスダッシュボードを示しています。


Top dimensions
Metrics
Metrics dashboard
PostgreSQL database health summary
CPU utilization (Percent)
100.00
50.00
0
03/26
04/02
04/09
04/16
os.cpuUtilization.steal.avg
os.cpuUtilization.guest.avg
os.cpuUtilization.irq.avg
os.cpuUtilization.wait.avg
os.cpuUtilization.user.avg
os.cpuUtilization.system.avg
Memory (Bytes)
119.0M
115.0M
111.1M
os.memory.free.avg
IO latency (Milliseconds)
0.70
0.39
0.08
os.diskIO.filesystem.await.avg
os.diskIO.nvme1n1.await.avg
os.diskIO.rdsdev.await.avg
IO operations (Per second)
17.8
8.9
os.diskIO.filesystem.readIOsPS.avg
os.diskIO.filesystem.writeIOsPS.avg
os.diskIO.nvme1n1.readIOsPS.avg
IO throughput (KB per second)
248
124
os.diskIO.filesystem.readKbPS.avg
os.diskIO.filesystem.writeKbPS.avg
os.diskIO.nvme1n1.readKbPS.avg
Vacuum: Maximum transaction IDs (Transactions)
241,291
215,580
189,869
db.Transactions.max_used_xact_ids.avg

Amazon CloudWatch を使用した Amazon Aurora メトリクスのモニタリング

Amazon CloudWatch はメトリクスリポジトリです。リポジトリは、Amazon Aurora から raw データを収集し、リアルタイムに近い読み取り可能なメトリクスに加工することができます。CloudWatch に送信される Amazon Aurora メトリクスの詳細なリストについては、「Amazon Aurora のメトリクスリファレンス」を参照してください。

トピック

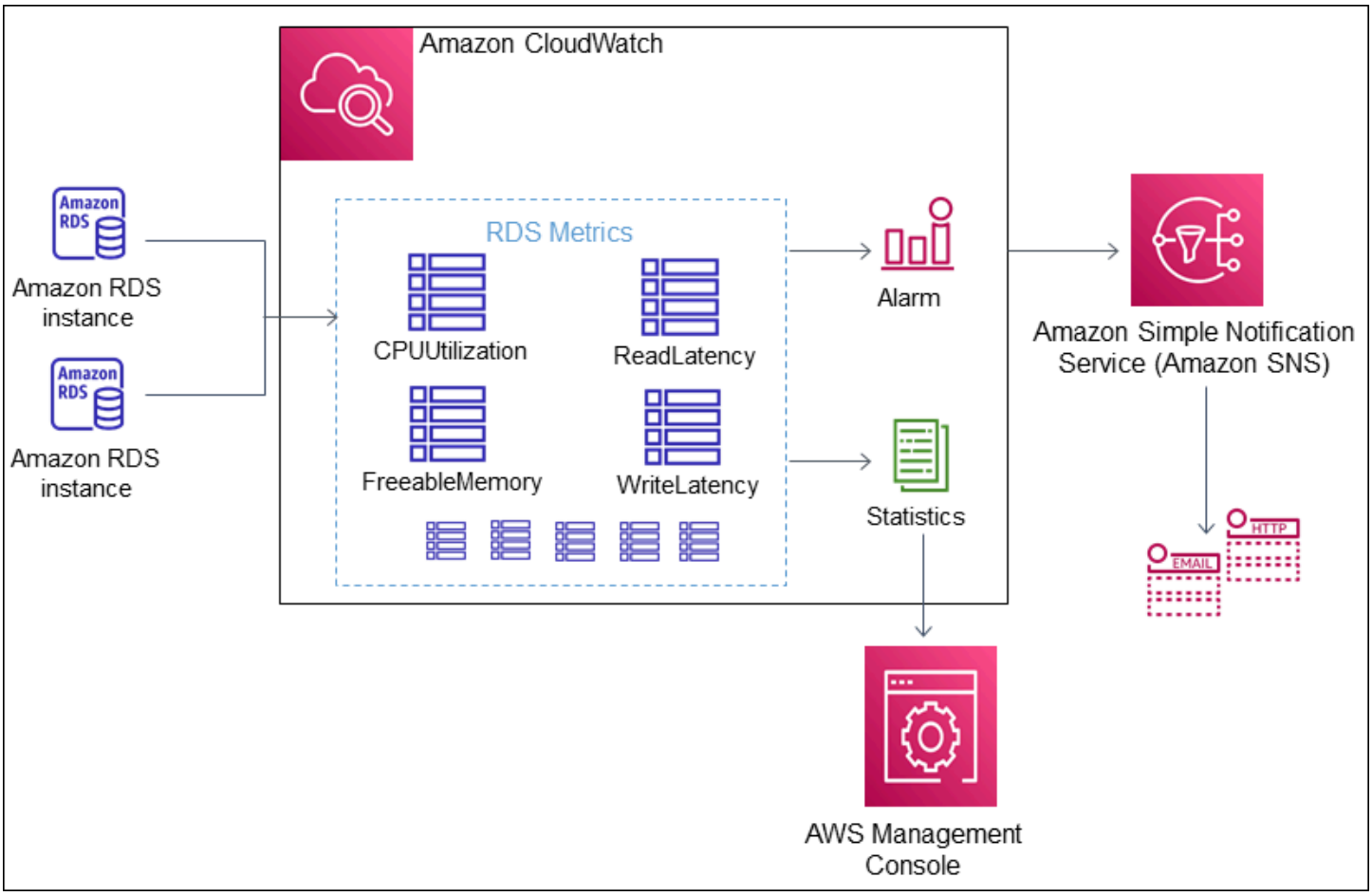
- Amazon Aurora および Amazon CloudWatch の概要
- CloudWatch コンソールおよび AWS CLI での DB クラスターメトリクスの表示
- Performance Insights メトリクスの CloudWatch へのエクスポート
- Amazon Aurora をモニタリングするための CloudWatch アラームの作成

Amazon Aurora および Amazon CloudWatch の概要

デフォルトでは、Amazon Aurora はメトリクスデータを 1 分間隔で CloudWatch に自動的に送信します。例えば、`CPUUtilization` メトリクスは、DB インスタンスの CPU 使用率のパーセンテージを時間の経過とともに記録します。期間が 60 秒 (1 分) のデータポイントは、15 日間使用できます。これにより、履歴情報にアクセスし、ウェブアプリケーションやサービスのパフォーマンスを確認できます。

Performance Insights メトリクスダッシュボードを Amazon RDS から Amazon CloudWatch にエクスポートできるようになりました。事前設定またはカスタマイズされたメトリクスダッシュボードを新しいダッシュボードとしてエクスポートするか、それらを既存の CloudWatch ダッシュボードに追加できます。エクスポートしたメトリクスはCloudWatch コンソールに表示できます。Performance Insights メトリクスダッシュボードを CloudWatch にエクスポートする方法の詳細については、「Performance Insights メトリクスの CloudWatch へのエクスポート」を参照してください。

次の図に示すように、CloudWatch メトリクスのアラームを設定できます。例えば、インスタンスの CPU 使用率が 70% を超えたときに通知するアラームを作成できます。Amazon Simple Notification Service を設定して、しきい値が過ぎたときにメールを送信できます。



Amazon RDS は、次のタイプのメトリクスを Amazon CloudWatch に発行します。

- クラスターレベルとインスタンスレベルの両方での Aurora メトリクス

  これらのメトリクスの表については、「Amazon Aurora の Amazon CloudWatch メトリクス」を参照してください。

- Performance Insights メトリクス

  これらのメトリクスの表については、「Performance Insights の Amazon CloudWatch メトリクス」および「Performance Insights カウンターメトリクス」を参照してください。

- 拡張モニタリングメトリクス (Amazon CloudWatch Logs に公開)

  これらのメトリクスの表については、「拡張モニタリングの OS メトリクス」を参照してください。

- AWS アカウント の Amazon RDS サービスクォータの使用状況メトリクス

  これらのメトリクスの表については、「Amazon Aurora の Amazon CloudWatch 使用状況メトリクス」を参照してください。Amazon RDS のクォータの詳細については、Amazon Aurora のクォータと制約 を参照してください。

CloudWatch の詳細については、Amazon CloudWatch ユーザーガイドの「Amazon CloudWatch とは」を参照してください。CloudWatch メトリクスの保持の詳細については、「メトリクスの保持」を参照してください。

CloudWatch コンソールおよび AWS CLI での DB クラスターメトリクスの表示

CloudWatch を使用して DB インスタンスのメトリクスを表示する方法の詳細を次に示します。CloudWatch Logs を使用して DB インスタンスのオペレーティングシステムのメトリクスをリアルタイムでモニタリングする方法については、「拡張モニタリングを使用した OS メトリクスのモニタリング」を参照してください。

Amazon Aurora リソースを使用する場合、Amazon Aurora が 1 分ごとにメトリクスとディメンションを Amazon CloudWatch に送信します。

Performance Insights メトリクスダッシュボードを Amazon RDS から Amazon CloudWatch にエクスポートし、これらのメトリクスを CloudWatch コンソールで表示できるようになりました。Performance Insights メトリクスダッシュボードを CloudWatch にエクスポートする方法の詳細

については、「Performance Insights メトリクスの CloudWatch へのエクスポート」を参照してください。

CloudWatch コンソールや CLI で Amazon Aurora のメトリクスを表示する場合は、次の手順に従います。

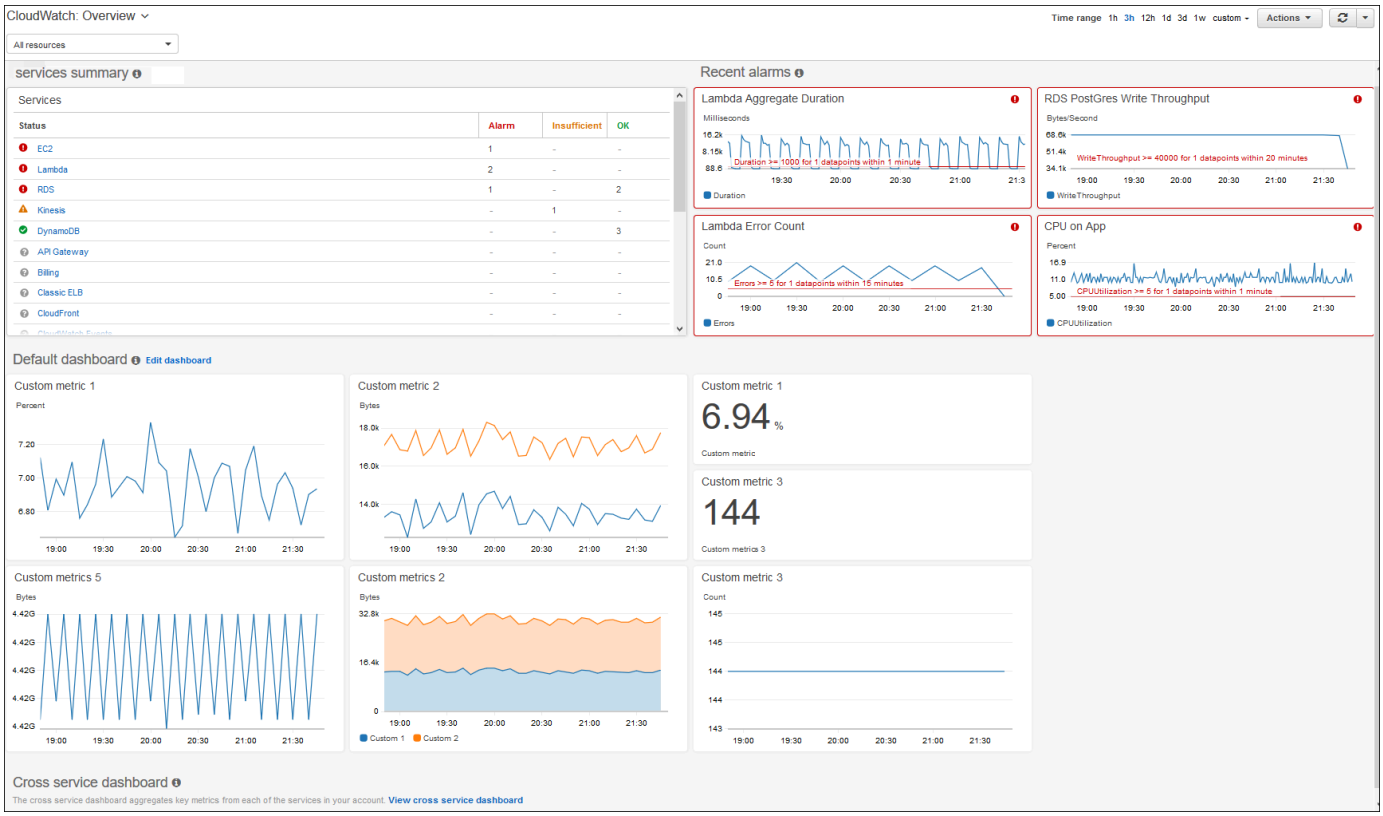
コンソール

Amazon CloudWatch コンソールを使用してメトリクスを表示するには

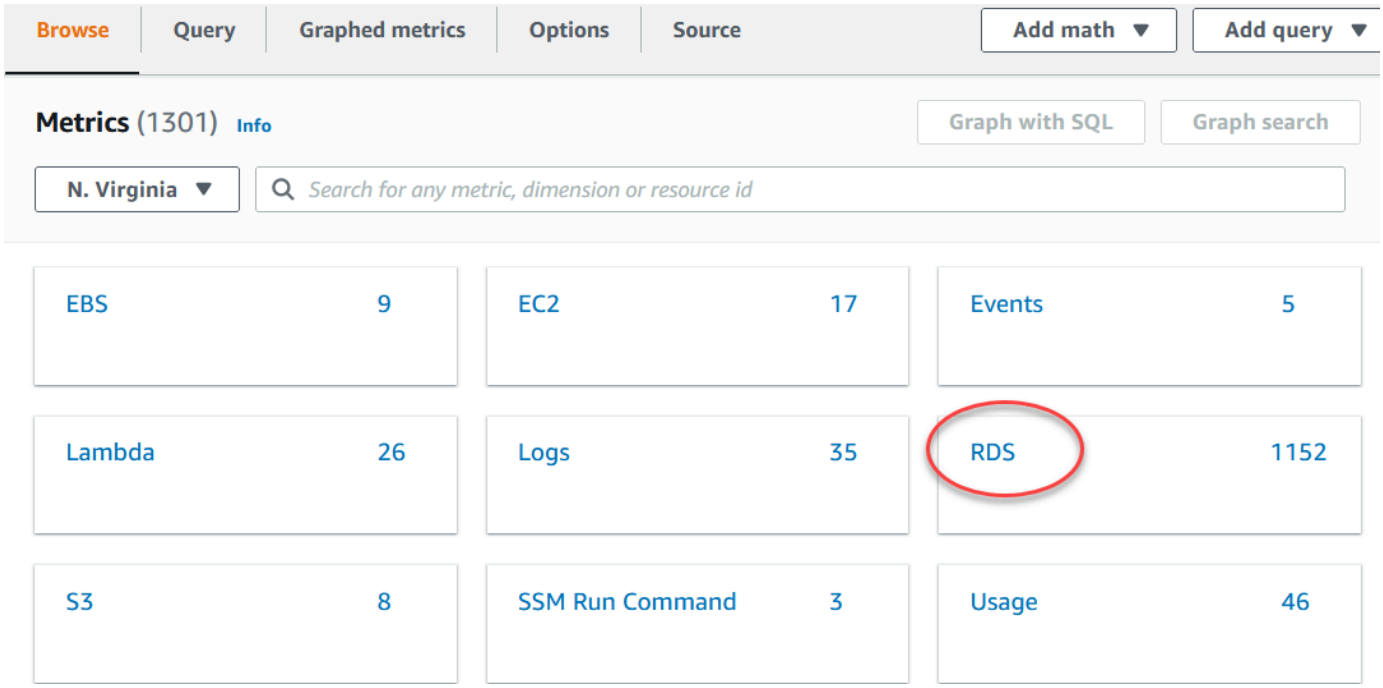
メトリクスはまずサービスの名前空間ごとにグループ化され、次に各名前空間内のさまざまなディメンションの組み合わせごとにグループ化されます。

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。

   CloudWatch 概要のホームページが表示されます。

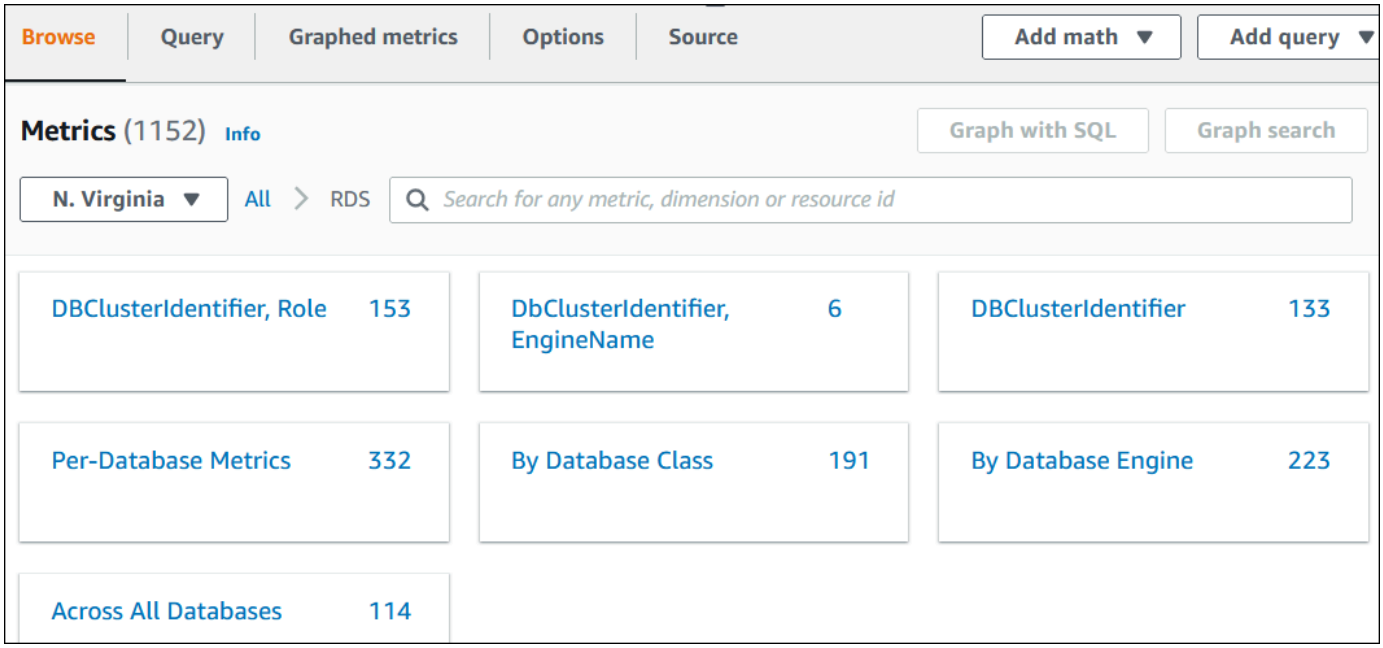


2. 必要に応じて AWS リージョン を変更します。ナビゲーションバーから、AWS リソースがある AWS リージョン リージョンを選択します。詳細については、「リージョンとエンドポイント」を参照してください。
3. ナビゲーションペインで、[Metrics] (メトリクス)、[All metrics] (すべてのメトリクス) の順に選択します。

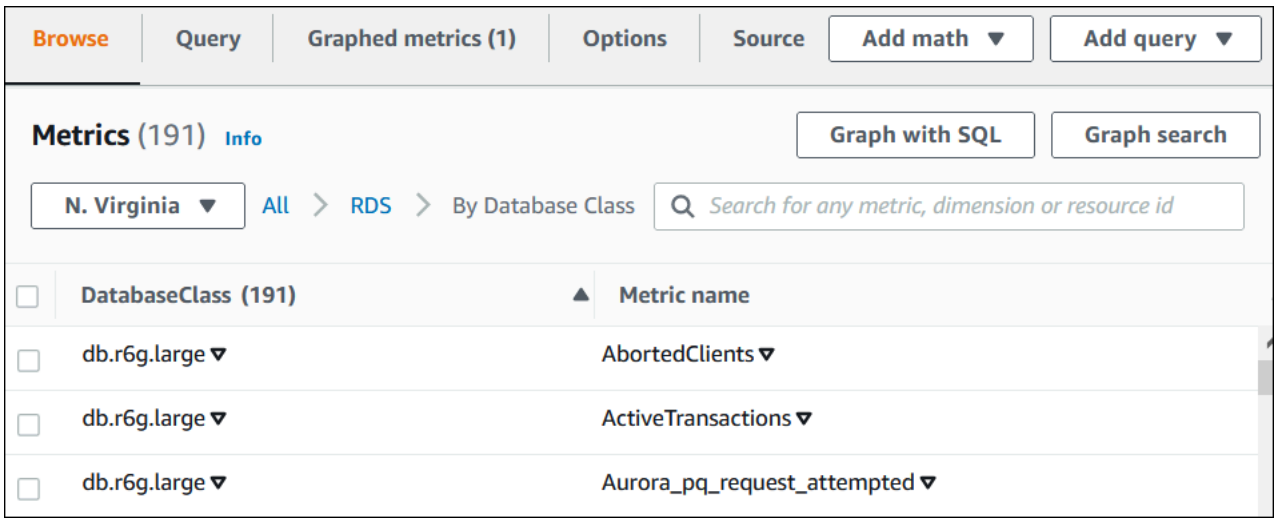


4. 下にスクロールし、RDS メトリクス名前空間を選択します。

   ページに Amazon Aurora ディメンションが表示されます。これらのディメンションの詳細については、「Aurora の Amazon CloudWatch ディメンション」を参照してください。



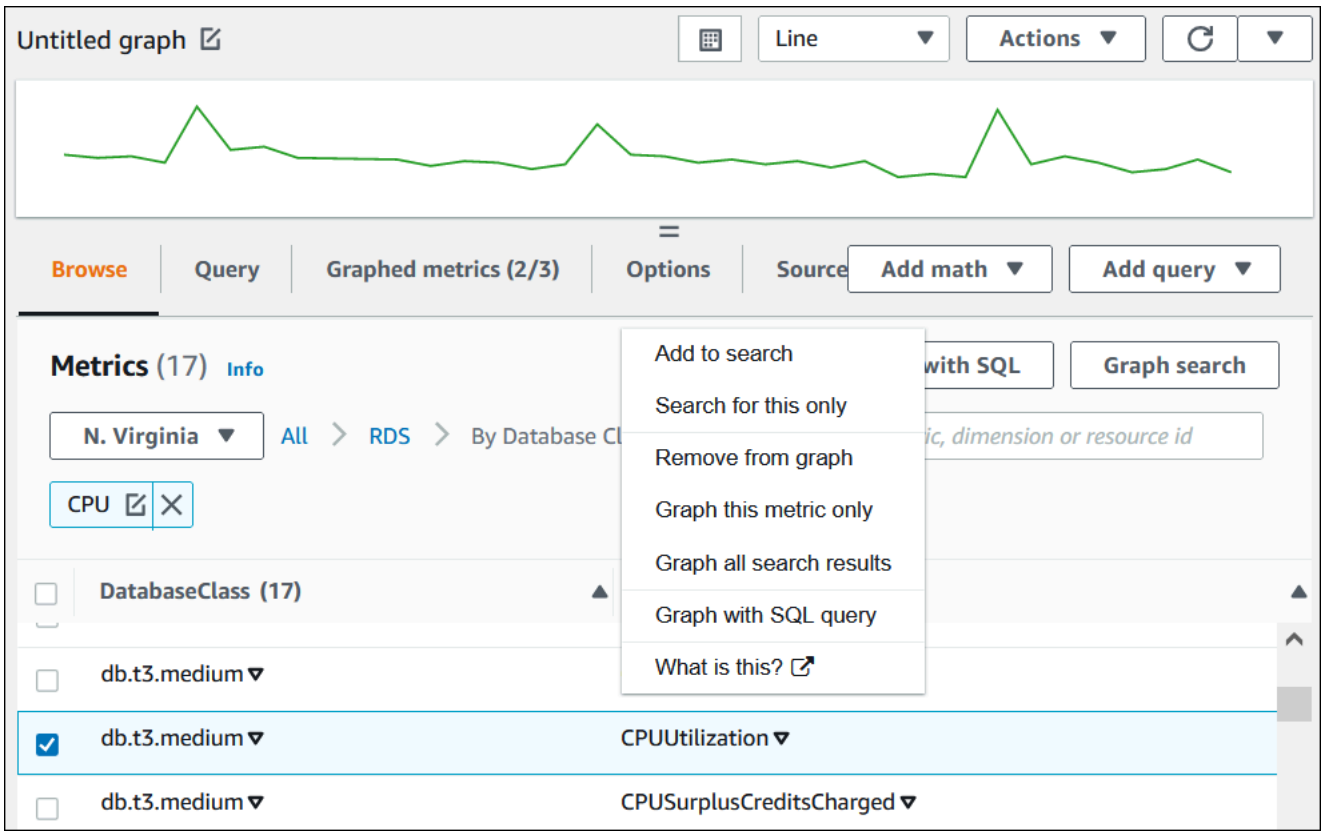
5. メトリクスディメンションを選択します。例えば [データベースクラス別] を選択します。



6. 次のアクションのいずれかを実行します。

   - メトリクスを並べ替えるには、列見出しを使用します。
   - メトリクスをグラフ表示するには、メトリクスの横にあるチェックボックスを選択します。
   - リソースでフィルタするには、リソース ID を選択し、[Add to search] (検索に追加) を選択します。
   - メトリクスでフィルターするには、メトリクス名を選択し、[Add to search] (検索に追加) を選択します。

   次の例では、db.t3.medium クラスをフィルタリングし、CPUUtilization メトリクスをグラフ化します。



CloudWatch メトリクスを使用して Aurora PostgreSQL のリソース使用状況を分析する方法の詳細を次に示します。詳細については、「Amazon CloudWatch メトリクスを使用して Aurora PostgreSQL のリソース使用状況を分析する」を参照してください。

AWS CLI

AWS CLI を使用してメトリクスの情報を取得するには、CloudWatch の `list-metrics` コマンドを使用します。次の例では、AWS/RDS 名前空間にすべてのメトリクスがリストされています。

```
aws cloudwatch list-metrics --namespace AWS/RDS
```

メトリクスデータを取得するには、`get-metric-data` コマンドを使用します。

次の例では、特定の 24 時間において 5 分の精度でインスタンス my-instance のCPUUtilization 統計情報を取得します。

次の内容を含む JSON ファイル CPU_metric.json を作成します。

```
{
   "StartTime" : "2023-12-25T00:00:00Z",
```

```
   "EndTime" : "2023-12-26T00:00:00Z",
   "MetricDataQueries" : [{
     "Id" : "cpu",
     "MetricStat" : {
    "Metric" : {
        "Namespace" : "AWS/RDS",
        "MetricName" : "CPUUtilization",
        "Dimensions" : [{ "Name" : "DBInstanceIdentifier" , "Value" : my-instance}]
    },
       "Period" : 360,
       "Stat" : "Minimum"
     }
   }]
}
```

Example

Linux、macOS、Unix の場合:

```
aws cloudwatch get-metric-data \
    --cli-input-json file://CPU_metric.json
```

Windows の場合:

```
aws cloudwatch get-metric-data ^
     --cli-input-json file://CPU_metric.json
```

出力例は次のとおりです。

```
{
    "MetricDataResults": [
        {
            "Id": "cpu",
            "Label": "CPUUtilization",
            "Timestamps": [
                "2023-12-15T23:48:00+00:00",
                "2023-12-15T23:42:00+00:00",
                "2023-12-15T23:30:00+00:00",
                "2023-12-15T23:24:00+00:00",
                ...
            ],
            "Values": [
```

```
                13.299778337027714,
                13.677507543049558,
                14.24976250395827,
                13.02521708695145,
                ...
            ],
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```

詳細については、「Amazon CloudWatch ユーザーガイド」の「メトリクスの統計の取得」を参照してください。

Performance Insights メトリクスの CloudWatch へのエクスポート

Performance Insights では、DB インスタンスの事前設定済みまたはカスタムメトリクスダッシュボードを Amazon CloudWatch にエクスポートできます。メトリクスダッシュボードを新しいダッシュボードとしてエクスポートするか、それらを既存の CloudWatch ダッシュボードに追加できます。ダッシュボードを既存の CloudWatch ダッシュボードに追加することを選択した場合、ヘッダーラベルを作成して、メトリクスが CloudWatch ダッシュボードの個別のセクションに表示されるようにすることができます。

エクスポートしたメトリクスダッシュボードは、CloudWatch コンソールで表示できます。Performance Insights メトリクスダッシュボードをエクスポートした後に、新しいメトリクスを追加した場合、CloudWatch コンソールに新しいメトリクスを表示するには、このダッシュボードを再度エクスポートする必要があります。

Performance Insights ダッシュボードでメトリクスウィジェットを選択し、CloudWatch コンソールでメトリクスデータを表示することもできます。

CloudWatch コンソールでメトリクスの表示する詳細については、「CloudWatch コンソールおよび AWS CLI での DB クラスターメトリクスの表示」を参照してください。

Performance Insights メトリクスを新しいダッシュボードとして CloudWatch にエクスポート

Performance Insights ダッシュボードから事前設定済みまたはカスタムのメトリクスダッシュボードを選択し、新しいダッシュボードとして CloudWatch にエクスポートします。エクスポートしたダッシュボードは、CloudWatch コンソールで表示できます。

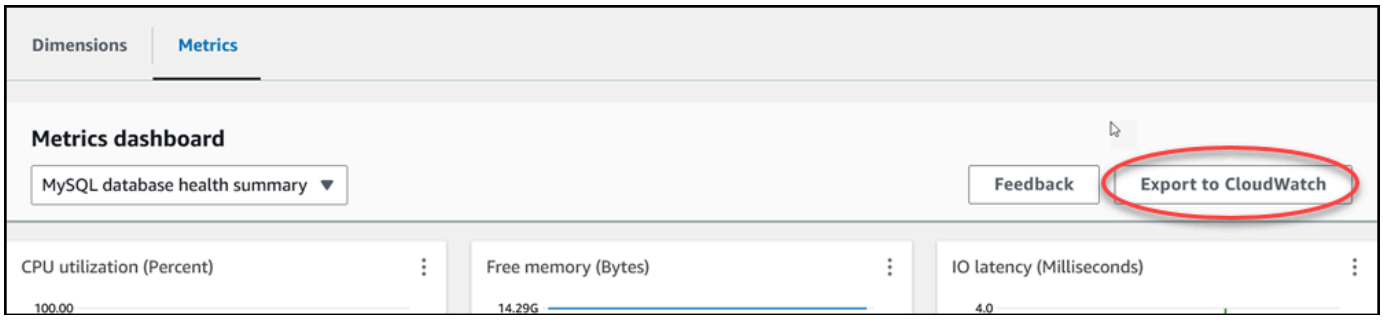
Performance Insights メトリクスダッシュボードを新しいダッシュボードとして CloudWatch にエクスポートするには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。

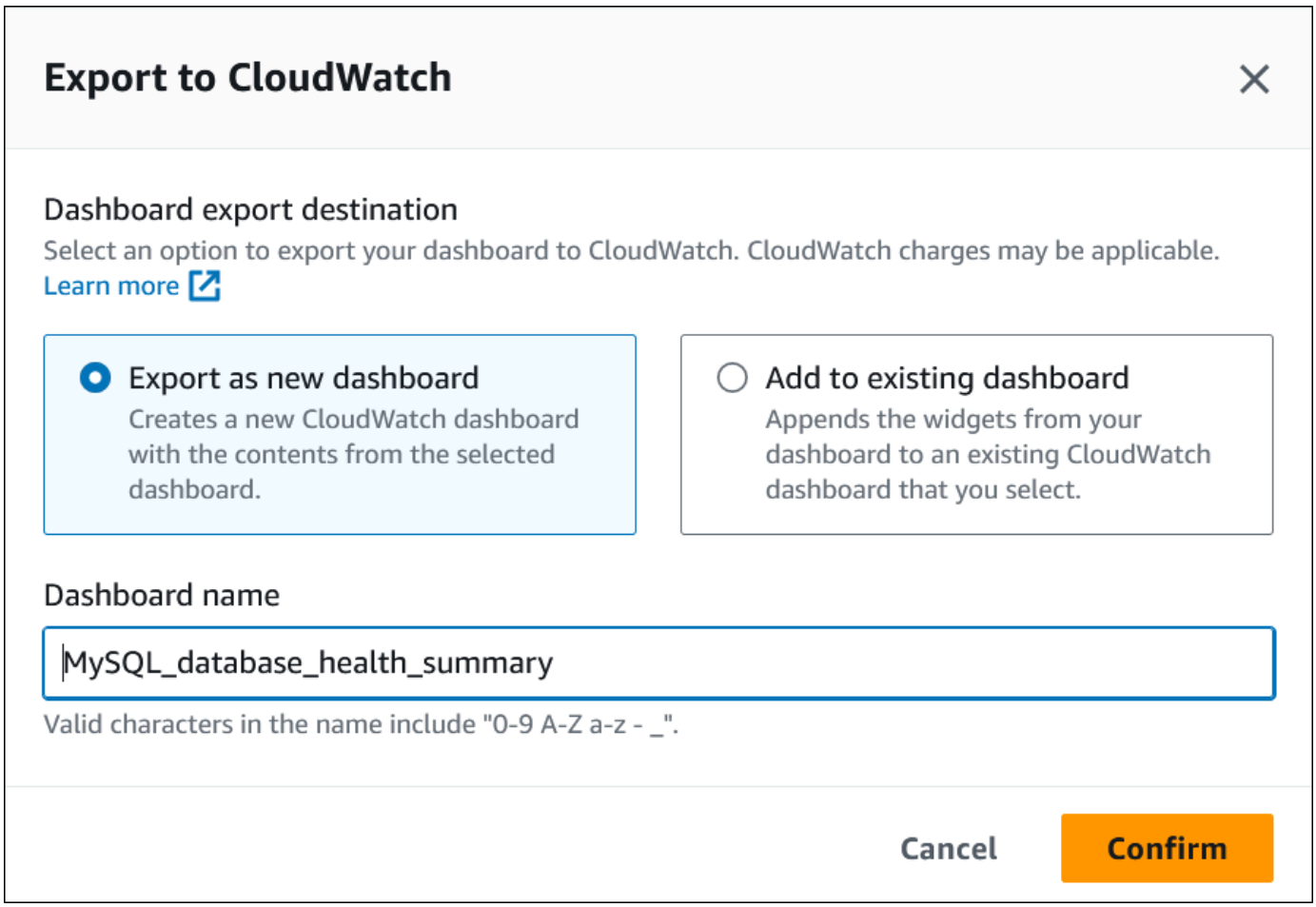
   この DB インスタンスに Performance Insights ダッシュボードが表示されます。
4. 下にスクロールして [メトリクス] を選択します。

   デフォルトでは、Performance Insights メトリクスで、事前設定されたダッシュボードが表示されます。
5. 事前設定されたダッシュボードまたはカスタムダッシュボードを選択してから、[CloudWatch にエクスポート] を選択します。

   [CloudWatch にエクスポート] ウィンドウが表示されます。

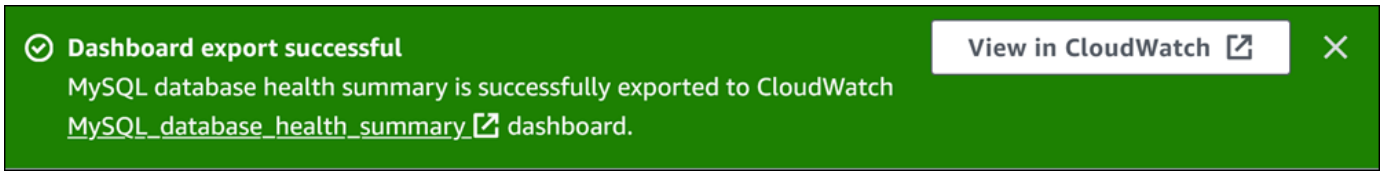


6. [新しいダッシュボードとしてエクスポート] を選択します。



7. [ダッシュボード名] フィールドに新しいダッシュボードの名前を入力し、[確認] を選択します。

   ダッシュボードのエクスポートが成功すると、バナーにメッセージが表示されます。



8. リンクまたはバナーの [CloudWatch で表示] を選択すると、CloudWatch コンソールにメトリクスダッシュボードが表示されます。

既存の CloudWatch ダッシュボードへの Performance Insights メトリクスの追加

事前設定済みまたはカスタムのメトリクスダッシュボードを既存の CloudWatch ダッシュボードに追加します。メトリクスダッシュボードにラベルを追加して、CloudWatch ダッシュボードの個別のセクションに表示することができます。

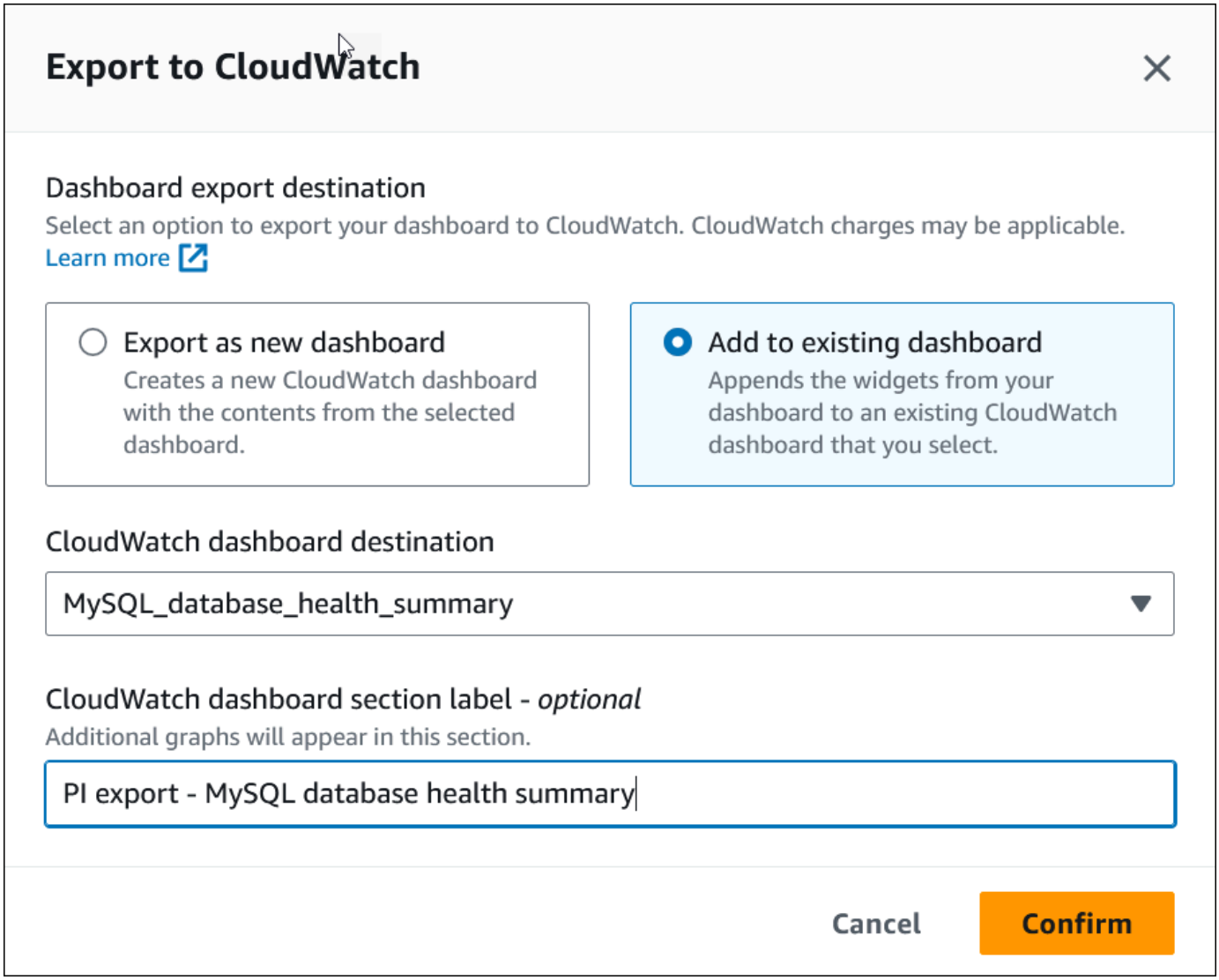
メトリクスを既存の CloudWatch ダッシュボードにエクスポートするには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。

   この DB インスタンスに Performance Insights ダッシュボードが表示されます。
4. 下にスクロールして [メトリクス] を選択します。

   デフォルトでは、Performance Insights メトリクスで、事前設定されたダッシュボードが表示されます。
5. 事前設定されたダッシュボードまたはカスタムダッシュボードを選択し、[CloudWatch にエクスポート] を選択します。

   [CloudWatch にエクスポート] ウィンドウが表示されます。
6. [既存のダッシュボードに追加] を選択します。



7. ダッシュボードの送信先とラベルを指定し、[確認] を選択します。

   - [CloudWatch ダッシュボードの送信先] - 既存の CloudWatch ダッシュボードを選択します。
   - [CloudWatch ダッシュボードセクションラベル - オプション] - CloudWatch ダッシュボードのこのセクションに表示する Performance Insights メトリクスの名前を入力します。

   ダッシュボードのエクスポートが成功すると、バナーにメッセージが表示されます。

8. リンクまたはバナーの [CloudWatch で表示] を選択すると、CloudWatch コンソールにメトリクスダッシュボードが表示されます。

CloudWatch での Performance Insights メトリクスウィジェットの表示

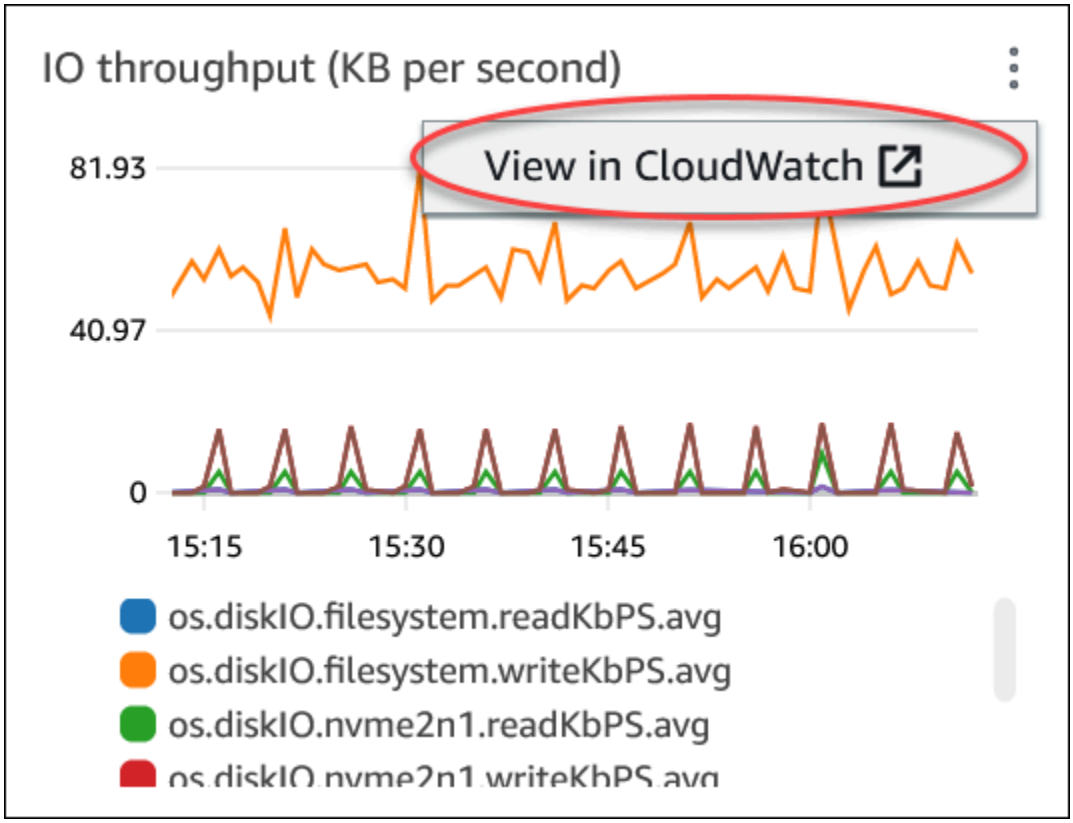
Amazon RDS Performance Insights ダッシュボードで Performance Insights メトリクスウィジェットを選択し、CloudWatch コンソールでメトリクススデータを表示します。

メトリクスウィジェットをエクスポートして CloudWatch コンソールでメトリクスデータを表示するには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。

   この DB インスタンスに Performance Insights ダッシュボードが表示されます。
4. [メトリクス] までスクロールします。

   デフォルトでは、Performance Insights メトリクスで、事前設定されたダッシュボードが表示されます。
5. メトリクスウィジェットを選択し、メニューで [CloudWatch で表示] を選択します。



   メトリクスデータは CloudWatch コンソールに表示されます。

Amazon Aurora をモニタリングするための CloudWatch アラームの作成

アラームの状態が変わったら、Amazon SNS メッセージを送信する Amazon CloudWatch のアラームを作成することができます。1 つのアラームで、指定した期間中、1 つのメトリクスをモニタリングします。アラームは、指定された複数の期間にわたるしきい値に関連するメトリクスの値に基づいて、1 つ以上のアクションを実行することもできます。アクションは、Amazon SNS トピックまたは Amazon EC2 Auto Scaling ポリシーに送信される通知です。

アラームは、持続している状態変化に対してのみアクションを呼び出します。CloudWatch アラームは、特定の状態にあるというだけの理由ではアクションを呼び出しません。状態が変わって、変わった状態が指定期間にわたって維持される必要があります。

> **Note**
>
> Aurora では、特定の DB インスタンスのメトリクスに依存せずに、`WRITER` ロールまたは `READER` ロールのメトリクスを使用してアラームを設定します。Aurora DB インスタンスのロールは、時間の経過とともにロールを変更できます。これらのロールベースのメトリクスは、CloudWatch コンソールで確認できます。
>
> Aurora Auto Scaling は、`READER` ロールのメトリクスに基づいて自動的にアラームを設定します。Aurora Auto Scaling の詳細については、「Aurora レプリカでの Amazon Aurora Auto Scaling の使用」を参照してください。

CloudWatch コンソールの DB_PERF_INSIGHTS メトリクス数学関数を使用して Amazon RDS にクエリを実行し、Performance Insights カウンターメトリクスを取得できます。DB_PERF_INSIGHTS 関数には、1 分未満の間隔での DBLoad メトリクスも含まれます。これらのメトリクスに基づいた CloudWatch アラームを設定することができます。

アラームの作成方法の詳細については、「AWS データベースから Performance Insights カウンターメトリクスのアラームを作成する」を参照してください。

AWS CLI を使用してアラームを設定するには

- `put-metric-alarm` を呼び出します。詳細については、「AWS CLI コマンドリファレンス」を参照してください。

CloudWatch API を使用してアラームを設定するには

- を呼び出します。`PutMetricAlarm`詳細については、Amazon CloudWatch API リファレンスを参照してください。

Amazon SNS トピックの設定およびアラームの作成の詳細については、「Amazon CloudWatch アラームの使用」を参照してください。

Amazon Aurora での Performance Insights を使用したDB 負荷のモニタリング

Performance Insights は、既存の Amazon Aurora モニタリング機能を拡張して、クラスターのパフォーマンスを明確にし、分析しやすくします。Performance Insights ダッシュボードを使用してAmazon Aurora クラスターのロードのデータベースロードを視覚化したり、ロードを待機、SQL ステートメント、ホスト、ユーザー別にフィルタリングしたりできます。Amazon DocumentDB での Performance Insights 使用については、「Amazon DocumentDB デベロッパーガイド」を参照してください。

トピック

Amazon Aurora での Performance Insights の概要

デフォルトで RDS は、すべての Amazon RDS エンジンのコンソール作成ウィザードで Performance Insights を有効にします。DB クラスターレベルで Performance Insights を有効にすると、RDS はクラスター内のすべての DB インスタンスに対して Performance Insights を有効にします。DB インスタンスに複数のデータベースがある場合、Performance Insights はパフォーマンスデータを集計します。

Amazon Aurora の Performance Insights の概要は次の動画で確認できます。

Amazon Aurora PostgreSQL のパフォーマンスを分析するパフォーマンスインサイトを使用する

トピック

- Amazon Aurora DB エンジンとインスタンスクラスでサポートされている Performance Insights
- Performance Insights の料金とデータ保持

データベース負荷

データベース負荷 (DB 負荷) は、データベース内のセッションアクティビティのレベルを測定します。DBLoad は Performance Insights の主要なメトリクスで、Performance Insights は 1 秒ごとに DB 負荷を収集します。

トピック

- アクティブなセッション
- 平均アクティブセッション
- 平均アクティブ実行
- ディメンション

アクティブなセッション

データベースセッションは、リレーショナルデータベースとのアプリケーションのダイアログを表します。アクティブなセッションとは、DB エンジンに作業を送信し、レスポンスを待っている接続です。

セッションは、CPU での動作中、またはリソースが使用可能になるのを待っているときにアクティブになります。例えば、アクティブなセッションでは、ページ (またはブロック) がメモリに読み込まれるのを待機し、ページからデータを読み取る間に CPU を消費することがあります。

平均アクティブセッション

平均アクティブセッション (AAS)はDBLoadPerformance Insights のメトリクスの単位です。データベース上で同時にアクティブなセッション数を測定します。

毎秒、Performance Insights は、クエリを同時に実行するセッションの数をサンプリングします。Performance Insights は、アクティブなセッションごとに以下のデータを収集します。

- SQL ステートメント
- セッション状態 (CPU で実行中または待機中)
- ホスト
- SQL を実行しているユーザー

Performance Insights は、特定期間の総セッション数を総サンプル数で割って AAS を計算します。たとえば、次の表は、1 秒間隔で実行中のクエリの連続する 5 つのサンプルを示しています。

例	クエリを実行しているセッション数	AAS	計算
1	2	2	合計 2 セッション/1 サンプル
2	0	1	合計 2 セッション/2 サンプル
3	4	2	合計 6 セッション/3 サンプル
4	0	1.5	合計 6 セッション/4 サンプル
5	4	2	合計 10 セッション/5 サンプル

前述の例では、時間間隔の DB ロードは 2 AAS でした。この測定は、5 つのサンプルを採取した期間に、平均して 2 つのセッションがある時点でアクティブであったことを意味します。

平均アクティブ実行

1 秒あたりの平均アクティブ実行 (AAE) は AAS に関連しています。AAE を計算するために、Performance Insightsでは、クエリの合計実行時間を時間間隔で割ります。次の表に、前述の表の同じクエリに対する AAE 計算を示します。

経過時間 (秒)	合計実行時間 (秒)	AAE	計算
60	120	2	120 実行秒 / 60 経過秒
120	120	1	120 実行秒 / 120 経過秒
180	380	2.11	380 実行秒 /180秒経過
240	380	(1.58)	380 実行秒/240秒経過

経過時間 (秒)	合計実行時間 (秒)	AAE	計算
300	600	2	600 実行秒/300 経過秒

ほとんどの場合、クエリの AAS と AAE はほぼ同じです。ただし、計算への入力は異なるデータソースであるため、計算はわずかに異なります。

ディメンション

この db.load メトリクスは、ディメンションと呼ばれるサブコンポーネントに分割できるため、他の時系列メトリクスとは異なります。ディメンションは、DBLoad メトリクスのさまざまな特性のカテゴリにより「スライス化されている」と考えることができます。

パフォーマンスの問題を診断する場合、多くの場合、以下のディメンションが最も役立ちます。

トピック

- 待機イベント
- 上位の SQL

のディメンションの詳細なリストについては、Auroraエンジン、「ディメンションでスライスされた DB の負荷」を参照してください。

待機イベント

待機イベントを指定すると、SQL ステートメントは、特定のイベントが発生するまで待機してから、実行を継続できます。待機イベントは、作業が妨げられる場所を示すため、DB ロードの重要なディメンションまたはカテゴリになります。

すべてのアクティブなセッションはCPU 上で実行されているか、待っています。例えば、セッションがメモリでバッファを検索したり、計算を実行したり、プロシージャコードを実行したりするときに CPU を消費します。セッションが CPU を消費していないときは、メモリバッファが空くのを待っているか、データファイルの読み取りやログの書き込みを待っている可能性があります。セッションのリソース待機時間が長くなると、CPU 上で動作する時間は短くなります。

データベースのチューニングのとき、セッションが待っているリソースを見つけようとすることがよくあります。例えば、2 つまたは 3 つの待機イベントが DBロードの 90% を占めることがあります。これは、平均して、アクティブなセッションが少数のリソースを待機するためにほとんどの時間を費やしていることを意味します。これらの待機の原因がわかれば、解決策を試すことができます。

待機イベントは、DB エンジンごとに異なります。

- Aurora MySQL の代表的な待機イベントのリストについては、「Aurora MySQL の待機イベント」を参照してください。これらの待機イベントを使用して調整する方法については、「Aurora MySQL のチューニング」を参照してください。
- MySQL のすべての待機イベントの詳細については、MySQL ドキュメントの「イベント待機サマリーテーブル」を参照してください。
- Aurora PostgreSQL の代表的な待機イベントのリストについては、「Amazon Aurora PostgreSQL のイベント」を参照してください。これらの待機イベントを使用して調整する方法については、「Aurora PostgreSQL の待機イベントでのチューニング」を参照してください。
- すべての PostgreSQL 待機イベントの詳細については、PostgreSQL ドキュメントの「統計コレクター > 待機イベントテーブル」を参照してください。

上位の SQL

待機イベントはボトルネックを示しますが、上位の SQL は、どのクエリが DB ロードの最も大きな原因になっているかを示します。例えば、多くのクエリが現在データベースで実行されている可能性がありますが、1 つのクエリが DB ロードの 99% を占めている可能性もあります。この場合、負荷が高いと、クエリに問題がある可能性があります。

デフォルトでは、Performance Insights コンソールには、データベース負荷の原因となっている上位の SQL クエリが表示されます。コンソールには、各ステートメントに関連する統計情報も表示されます。特定のステートメントのパフォーマンスの問題を診断するには、その実行プランを調べます。

最大 CPU 容量

ダッシュボードの [データベースロード] グラフで、セッション情報が収集、集計、表示されます。アクティブなセッションが最大 CPU 容量を超えているかどうかを確認するには、最大 vCPU ラインとの関係を調べます。Performance Insights は、最大 vCPU 値を DB インスタンスの vCPU (仮想 CPU) のコア数によって決定します。Aurora Serverless v2 の場合、[Max vCPU] (最大 vCPU) は、推定される vCPU の数を表します。

vCPU では一度に 1 つのプロセスを実行できます。プロセスの数が vCPU の数を超えると、プロセスはキューイングを開始します。キューイングが増加すると、パフォーマンスに影響します。DB 負荷が [Max vCPU (最大 vCPU)] ラインをしばしば超過し、プライマリ待機状態が CPU である場合、CPU が過負荷になっています。この場合、インスタンスへの接続を抑制したり、CPU ロードの高い SQL クエリを調整したり、より大きなインスタンスクラスを検討する必要があります。待機状

態の高い一貫したインスタンスは、解決するボトルネックまたはリソースの競合問題がある可能性があることを示します。これは、DB ロードが最大 vCPU ラインを超えていない場合にも該当します。

Amazon Aurora DB エンジンとインスタンスクラスでサポートされている Performance Insights

次の表に、Performance Insights をサポートしている Amazon Aurora DB エンジンを示します。

Amazon Aurora DB エンジン	サポート対象のエンジンバージョンとリージョン	インスタンスクラスに関する制限
Amazon Aurora MySQL 互換エディション	Aurora MySQL に関する Performance Insights のバージョンとリージョンの可用性の詳細については、「Aurora MySQL で Performance Insights を使用する」を参照してください。	Performance Insights のエンジンクラスには、以下の制限があります。 • db.t2 – サポートされていません • db.t3 – サポートされていません • db.t4g.micro および db.t4g.small — サポートされていません
Amazon Aurora PostgreSQL 互換エディション	Aurora PostgreSQL に関する Performance Insights のバージョンとリージョンの可用性の詳細については、「Aurora PostgreSQL で Performance Insights を使用する」を参照してください。	該当なし

Amazon Aurora DB エンジン、リージョン、およびインスタンスクラスでサポートされている Performance Insights 機能

次の表に、Performance Insights 機能をサポートしている Amazon Aurora DB エンジンを示します。

機能	料金階層	サポートされるリージョン	サポートされる DB エンジン	サポートされるインスタンスクラス
Performance Insights の SQL 統計	すべて	すべて	すべて	すべて
一定期間のデータベースパフォーマンスの分析	有料階層のみ	• 米国東部 (オハイオ) • 米国東部 (バージニア北部) • 米国西部 (北カリフォルニア) • 米国西部 (オレゴン) • アジアパシフィック (ムンバイ) • アジアパシフィック (ソウル) • アジアパシフィック (シンガポール) • アジアパシフィック (シドニー) • アジアパシフィック (東京) • カナダ (中部) • 欧州 (フランクフルト)	すべて	db.serverless (Aurora Serverless v2) を除くすべて

機能	料金階層	サポートされるリージョン	サポートされる DB エンジン	サポートされるインスタンスクラス
		• 欧州 (アイルランド) • 欧州 (ロンドン) • 欧州 (パリ) • 欧州 (ストックホルム)		

機能	料金階層	サポートされるリージョン	サポートされる DB エンジン	サポートされるインスタンスクラス
Performance Insights の事前対応型推奨事項の表示	有料階層のみ	• 米国東部 (オハイオ) • 米国東部 (バージニア北部) • 米国西部 (北カリフォルニア) • 米国西部 (オレゴン) • アジアパシフィック (ムンバイ) • アジアパシフィック (ソウル) • アジアパシフィック (シンガポール) • アジアパシフィック (シドニー) • アジアパシフィック (東京) • カナダ (中部) • 欧州 (フランクフルト) • 欧州 (アイルランド)	すべて	db.serverless (Aurora Serverless v2) を除くすべて

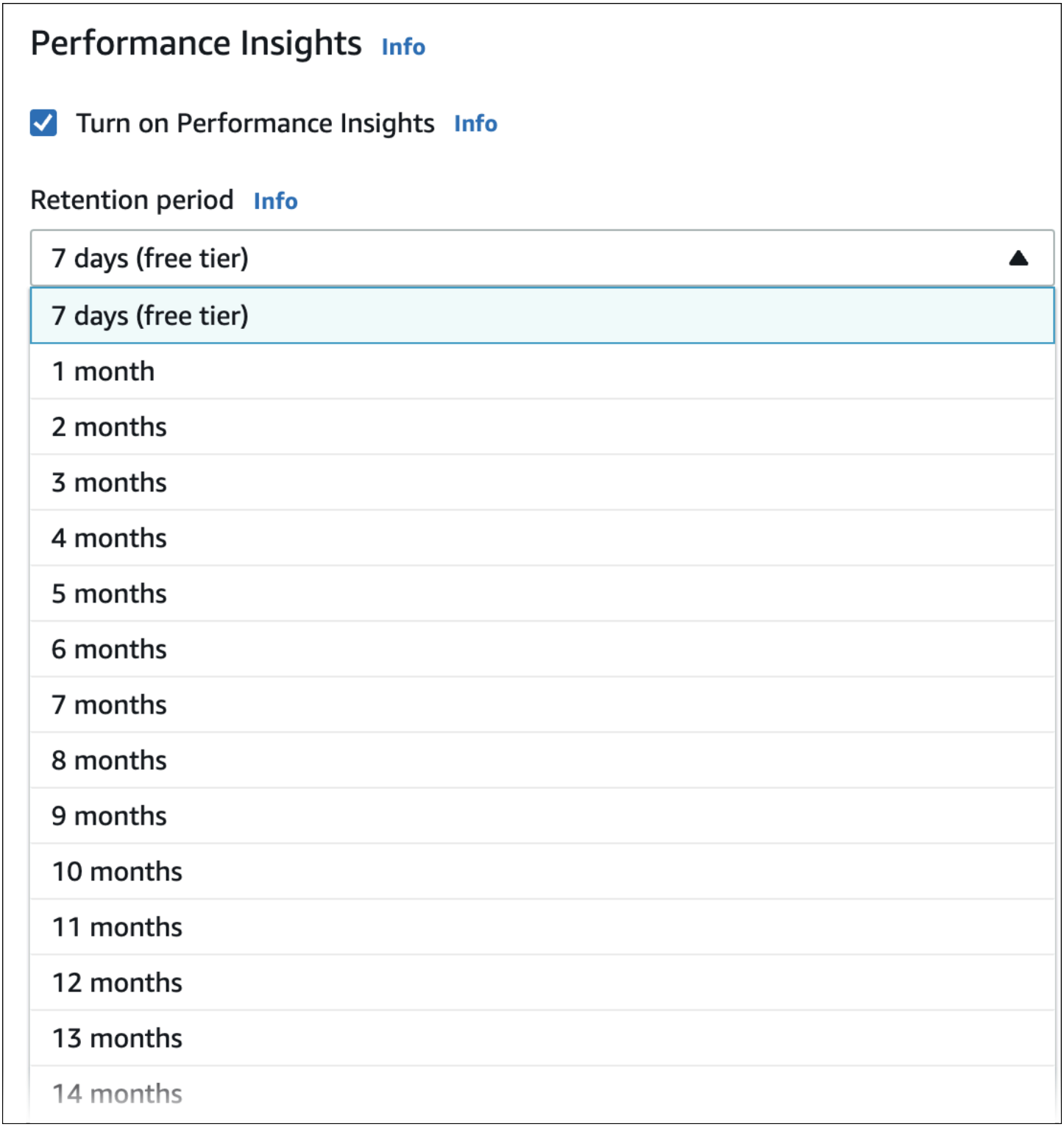
機能	料金階層	サポートされるリージョン	サポートされる DB エンジン	サポートされるインスタンスクラス
		• 欧州 (ロンドン) • 欧州 (パリ) • 欧州 (ストックホルム) • 南米 (サンパウロ)		

Performance Insights の料金とデータ保持

デフォルトでは、Performance Insights には、7 日間のパフォーマンスデータ履歴と 1 か月あたり 100 万件の API リクエストを含む無料利用枠が用意されています。また、より長い保持期間を購入することもできます。料金情報の詳細については、「Performance Insights の料金」を参照してください。

RDS コンソールでは、Performance Insights データの保持期間を次の中から選択できます。

- デフォルト (7 日)
- ***n*** か月 (***n*** は 1 ~ 24 の数値)



AWS CLI を使用して保持期間を設定する方法については、「AWS CLI」を参照してください。

Performance Insights の有効化と無効化

DB クラスターを作成する際に、Performance Insights を有効にすることができます。必要に応じて、後でDB クラスター内の任意のインスタンスのインスタンスレベルでオフにすることができます。Performance Insights を有効化または無効化した場合も、ダウンタイム、再起動、フェイルオーバーが発生することはありません。

> **Note**
>
> Performance Schema は、Aurora MySQL で使用される、オプションのパフォーマンスツールです。Performance Schema のオンとオフを切り替える場合は、再起動する必要があります。ただし、Performance Insights のオンとオフを切り替えた場合は、再起動する必要はありません。詳細については、「Aurora MySQL における Performance Insights の Performance Schema の有効化」を参照してください。

Aurora グローバルデータベースで Performance Insights を使用する場合は、各 AWS リージョン の DB インスタンスで Performance Insights を個別にオンにします。詳細については、「Amazon RDS Performance Insights を使用した Amazon Aurora Global Database のモニタリング」を参照してください。

Performance Insights エージェントは DB ホストの限られた CPU とメモリを消費します。DB の ロードが高い場合、エージェントはデータ収集の頻度を下げることでパフォーマンスへの影響を抑えます。

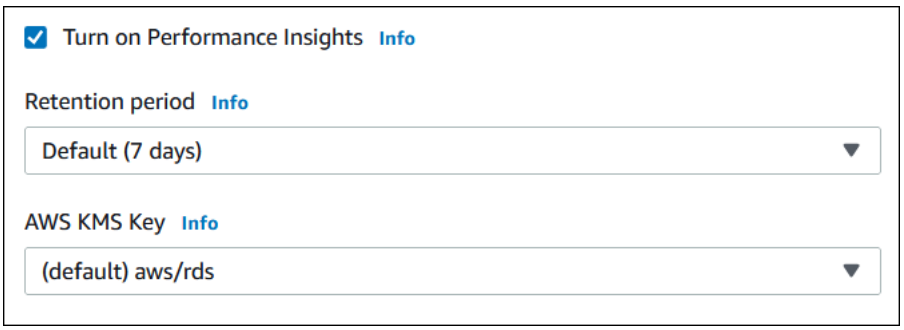
コンソール

DB クラスターを作成するときに、コンソールで Performance Insights のオンとオフを切り替えることができます。クラスター内の DB インスタンスを変更して、そのインスタンスの Performance Insights をオンまたはオフにすることができます。

DB クラスターの作成時に Performance Insights のオンとオフを切り替える

新しい DB クラスターを作成する場合、[Performance Insights] セクションの [Enable Performance Insights] (Performance Insights を有効にする) を選択して Performance Insights をオンにします。または、[Performance Insights の無効化] を選択します。DB クラスターを作成するには、「Amazon Aurora DB クラスターの作成」の DB エンジンの手順に従ってください。

次のスクリーンショットは [Performance Insights] セクションを示しています。



[Performance Insights の有効化] を選択すると、次のオプションがあります。

- 保持期間 - Performance Insights データを保持する期間。無料利用枠の保持設定は「デフォルト (7 日)」です。パフォーマンスデータをさらに長期間保持するには、1 ~ 24 か月を指定します。保持期間の詳細については、「Performance Insights の料金とデータ保持」を参照してください。
- AWS KMS key – AWS KMS key を指定します。Performance Insights は、潜在的に機密性の高いすべてのデータを KMS キーを使用して暗号化します。データは、転送中と不使用時のいずれも暗号化されます。詳細については、「Performance Insights 用の AWS KMS ポリシーの設定」を参照してください。

DB クラスターの DB インスタンスの変更時に Performance Insights のオンとオフを切り替える

コンソールでは、Performance Insights のオンとオフを切り替えるように DB クラスターの DB インスタンスを変更できます。クラスターレベルでは Performance Insights をオンまたはオフにすることはできません。クラスター内のインスタンスごとに実行する必要があります。

コンソールを使用して DB クラスター内の DB インスタンスの Performance Insights のオンとオフを切り替える

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. [データベース] をクリックします。
3. DB インスタンスを選択した上で、[Modify] (変更) を選択します。
4. [Performance Insights] セクションで、[Performance Insights の有効化] または [Performance Insightsの無効化] を選択します。

   [Performance Insights の有効化] を選択すると、次のオプションがあります。

   - 保持期間 - Performance Insights データを保持する期間。無料利用枠の保持設定は「デフォルト (7 日)」です。パフォーマンスデータをさらに長期間保持するには、1 ~ 24 か月を指定しま

す。保持期間の詳細については、「Performance Insights の料金とデータ保持」を参照してください。

- AWS KMS key - KMS キーを指定します。Performance Insights は、潜在的に機密性の高いすべてのデータを KMS キーを使用して暗号化します。データは、転送中と不使用時のいずれも暗号化されます。詳細については、「Amazon Aurora リソースの暗号化」を参照してください。

5. [続行] を選択します。
6. [変更のスケジュール] で、[今すぐ適用] を選択します。次にスケジュールされたメンテナンスウィンドウで [Apply] (適用) を選択すると、インスタンスではこの設定が無視され、Performance Insights が直ちにオンになります。
7. [インスタンスの変更] を選択します。

AWS CLI

create-db-instance AWS CLI コマンドを使用する場合は、`--enable-performance-insights` を指定して Performance Insights をオンにします。または、`--no-enable-performance-insights` を指定して Performance Insights をオフにします。

以下の AWS CLI コマンドを使用してこれらの値を指定することもできます。

- create-db-instance-read-replica
- modify-db-instance
- restore-db-instance-from-s3

次の手順では、AWS CLI を使用して DB クラスター内の既存の DB インスタンスで Performance Insights のオンとオフを切り替える方法について説明します。

AWS CLI を使用して DB クラスター内の DB インスタンスで Performance Insights のオンとオフを切り替えるには

- modify-db-instance AWS CLI コマンドを呼び出して以下の値を渡します。
  - `--db-instance-identifier` — DB クラスター内の DB インスタンスの名前です。
  - オンにする場合は `--enable-performance-insights`、オフにする場合は `--no-enable-performance-insights`

次の例では、sample-db-instance で Performance Insights をオンにします。

Linux、macOS、Unix の場合:

```
aws rds modify-db-instance \
    --db-instance-identifier sample-db-instance \
    --enable-performance-insights
```

Windows の場合:

```
aws rds modify-db-instance ^
    --db-instance-identifier sample-db-instance ^
    --enable-performance-insights
```

CLI で Performance Insights をオンにする際に、--performance-insights-retention-period オプションを使用して Performance Insights のデータを保持する日数を指定できます (オプション)。7、*month* * 31 (*month* は 1 ~ 23 の範囲の数値)、または 731 を指定できます。例えば、パフォーマンスデータを 3 か月間保持する場合は、93 (3 * 31) を指定します。デフォルトは 7 日間です。保持期間の詳細については、「Performance Insights の料金とデータ保持」を参照してください。

次の例では、sample-db-instance で Performance Insights をオンにして、Performance Insights のデータの保持期間を 93 日間 (3 か月) に指定します。

Linux、macOS、Unix の場合:

```
aws rds modify-db-instance \
    --db-instance-identifier sample-db-instance \
    --enable-performance-insights \
    --performance-insights-retention-period 93
```

Windows の場合:

```
aws rds modify-db-instance ^
    --db-instance-identifier sample-db-instance ^
    --enable-performance-insights ^
    --performance-insights-retention-period 93
```

94 日などの有効な値ではない保持期間を指定すると、RDS はエラーを発行します。

```
An error occurred (InvalidParameterValue) when calling the CreateDBInstance operation:
Invalid Performance Insights retention period. Valid values are: [7, 31, 62, 93, 124,
 155, 186, 217,
248, 279, 310, 341, 372, 403, 434, 465, 496, 527, 558, 589, 620, 651, 682, 713, 731]
```

RDS API

Amazon RDS API 操作の CreateDBInstance オペレーションを使用して DB クラスター内に新しい DB インスタンスを作成する場合、`EnablePerformanceInsights` を `True` に設定して、Performance Insights をオンにします。Performance Insights をオフにするには、`EnablePerformanceInsights` を `False` に設定します。

以下の API オペレーションを使用して `EnablePerformanceInsights` 値を指定することもできます。

- ModifyDBInstance
- CreateDBInstanceReadReplica
- RestoreDBInstanceFromS3

Performance Insights をオンにする際に、`PerformanceInsightsRetentionPeriod` パラメータを使用して Performance Insights のデータを保持する期間を日数で指定できます (オプション)。7、*`month`* * 31 (*`month`* は 1 ~ 23 の範囲の数値)、または 731 を指定できます。例えば、パフォーマンスデータを 3 か月間保持する場合は、93 (3 * 31) を指定します。デフォルトは 7 日間です。保持期間の詳細については、「Performance Insights の料金とデータ保持」を参照してください。

Aurora MySQL における Performance Insights の Performance Schema の有効化

Performance Schema は、Aurora MySQL ランタイムのパフォーマンスを低い詳細レベルでモニタリングするオプション機能です。Performance Schema は、データベースのパフォーマンスへの影響を最小限に抑えるように設計されています。Performance Insights は、Performance Schema の有無に関係なく使用できる独立した機能です。

トピック

- Performance Schema の概要

- Performance Insights と Performance Schema
- Performance Insights による Performance Schema の自動管理
- Performance Schema の再起動による影響
- Performance Insights が Performance Schema を管理しているかどうかの確認
- 自動管理用 Performance Schema の設定

Performance Schema の概要

Performance Schema は、Aurora MySQL データベースのイベントをモニタリングします。イベントとは、時間を消費し、タイミング情報を収集できるように実装されたデータベースサーバーアクションです。イベントの例には、以下のようなものがあります。

- 関数呼び出し
- オペレーティングシステムの待機
- SQL 実行のステージ
- SQL ステートメントのグループ

`PERFORMANCE_SCHEMA` ストレージエンジンは、Performance Schema 機能を実装するためのメカニズムです。このエンジンは、データベースのソースコード内の計測を使用してイベントデータを収集します。エンジンは、イベントを `performance_schema` データベースのメモリ専用テーブルに保存します。他のテーブルにクエリを実行するのと同様に、`performance_schema` をクエリできます。詳細については、MySQL リファレンスマニュアルの「MySQL Performance Schema」を参照してください。

Performance Insights と Performance Schema

Performance Insights と Performance Schema は別々の機能ですが、両者は関連しています。Aurora MySQL の Performance Insights の動作は、Performance Schema がオンになっているかどうか、およびオンになっている場合は、Performance Insights が Performance Schema を自動的に管理するかどうかによって異なります。次の表は、動作の説明です。

Performance Schema がオンになっている	Performance Insights 管理モード	Performance Insights の動作
はい	自動	•

Performance Schema がオンになっている	Performance Insights 管理モード	Performance Insights の動作
		詳細な低レベルのモニタリング情報を収集します。 • アクティブなセッションメトリクスを毎秒収集します。 • DB ロードを詳細な待機イベントごとに分類して表示し、ボトルネックの特定に使用できます。
はい	手動	• 待機イベントと SQL ごとのメトリクスを収集します • アクティブなセッションメトリックスを毎秒ではなく 5 秒ごとに収集します。 • 挿入や送信などのユーザー状態をレポートしますが、ボトルネックの特定には役立ちません。
いいえ	該当なし	• 待機イベント、SQL ごとのメトリクス、またはその他の詳細な低レベルのモニタリング情報を収集しません。 • アクティブなセッションメトリックスを毎秒ではなく 5 秒ごとに収集します。 • 挿入や送信などのユーザー状態をレポートしますが、ボトルネックの特定には役立ちません。

Performance Insights による Performance Schema の自動管理

Performance Insights を有効にした状態で Aurora MySQL DB インスタンスを作成すると、Performance Schema も有効になります。この場合、Performance Insights は Performance Schema パラメータを自動的に管理します。この設定を推奨します。

> **Note**
>
> t4g.medium インスタンスクラスでは、パフォーマンススキーマの自動管理はサポートされていません。

Performance Insights で Performance Schema を自動管理するには、`performance_schema` が `0` に設定されている必要があります。デフォルトでは、[Source] (ソース) の値は `system` です。

Performance Schema は、手動で管理することもできます。このオプションを選択した場合は、次の表の値に従ってパラメータを設定します。

パラメータ名	パラメータ値
`performance_schema`	1 ([Source] (ソース) 列には値 `system` があります)
`performance-schema-consumer-events-waits-current`	`ON`
`performance-schema-instrument`	`wait/%=ON`
`performance_schema_consumer_global_instrumentation`	`1`
`performance_schema_consumer_thread_instrumentation`	`1`

`performance_schema` パラメータの値を手動で変更し、後で自動管理に戻す方法については、「自動管理用 Performance Schema の設定」を参照してください。

> **Important**
>
> Performance Insights で Performance Schema を有効にしても、パラメータグループ値は変更されません。ただし、値は実行中の DB インスタンスで変更されます。変更された値を表示する唯一の方法は、`SHOW GLOBAL VARIABLES` コマンドを実行することです。

Performance Schema の再起動による影響

Performance Insights と Performance Schema は、DB インスタンスの再起動の要件が異なります。

Performance Schema

この機能をオンまたはオフにするには、DB インスタンスを再起動する必要があります。

Performance Insights

この機能をオンまたはオフにするために、DB インスタンスを再起動する必要はありません。

Performance Schema が現在有効になっていない場合、DB インスタンスを再起動せずに Performance Insights を有効にすると、Performance Schema は有効になりません。

Performance Insights が Performance Schema を管理しているかどうかの確認

Performance Insights が、現在、主要なエンジンバージョン 5.6、5.7、8.0 の Performance Schema を管理しているかどうかを確認するには、次の表を参照してください。

performance_schema パラメータの設定	[Source] (ソース) 列の設定	Performance Insights が Performance Schema を管理しているかどうか
`0`	`system`	はい
`0`、、または `1`	`user`	いいえ

Performance Insights が Performance Schema を自動管理しているかどうかを確認するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. [パラメータグループ] を選択します。
3. DB インスタンスのパラメータグループを選択します。
4. 検索バーに **`performance_schema`** と入力します。
5. [Source] (ソース) がシステムデフォルト、[Values] (値) が [0] であることをチェックします。上記の設定の場合、Performance Insights は Performance Schema を自動管理します。上記の設定ではない場合、Performance Insights は Performance Schema を自動管理していません。

自動管理用 Performance Schema の設定

DB インスタンスで Performance Insights がオンになっているが、現在 Performance Schema は管理していないと仮定します。Performance Insights が Performance Schema を自動管理できるようにするには、次のステップを実行します。

自動管理用 Performance Schema を設定するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. [パラメータグループ] を選択します。
3. DB インスタンスのパラメータグループの名前を選択します。
4. 検索バーに **performance_schema** と入力します。
5. performance_schema パラメータを選択します。
6. [パラメータの編集] を選択します。
7. performance_schema パラメータを選択します。
8. [値] で 0 を選択します。
9. [Save changes] (変更の保存) をクリックします。
10. DB インスタンスを再起動します。

> **Important**
>
> Performance Schema のオンとオフを切り替えるたびに、DB インスタンスを必ず再起動します。

インスタンスのパラメータの変更の詳細については、「DB パラメータグループのパラメータの変更」を参照してください。ダッシュボードのページの詳細については、「Performance Insights ダッシュボードを使用してメトリクスを分析する」を参照してください。MySQL Performance Schema の詳細については、MySQL 8.0 Reference Manual を参照してください。

Performance Insights 用のアクセスポリシーの設定

Performance Insights にアクセスするには、プリンシパルが AWS Identity and Access Management (IAM) から適切な許可を得る必要があります。以下の方法でアクセス権を付与することができます。

- `AmazonRDSPerformanceInsightsReadOnly` 管理ポリシーを、Performance Insights API のすべての読み取り専用操作にアクセスするためのアクセス許可セットまたはロールにアタッチします。
- `AmazonRDSPerformanceInsightsFullAccess` 管理ポリシーを、Performance Insights API のすべての操作にアクセスするためのアクセス許可セットまたはロールにアタッチします。
- カスタム IAM ポリシーを作成し、アクセス許可セットまたはロールにアタッチします。

また、Performance Insights を有効にしたときにカスタマーマネージドキーを指定した場合は、アカウント内のユーザーが AWS KMS key に対する `kms:Decrypt` アクセス許可および `kms:GenerateDataKey` アクセス許可を持っていることを確認します。

AmazonRDSPerformanceInsightsReadOnly ポリシーの IAM プリンシパルへのアタッチ

`AmazonRDSPerformanceInsightsReadOnly` は Amazon RDS Performance Insights API のすべての読み取り専用オペレーションへのアクセス許可を付与する AWS マネージドポリシーです。

`AmazonRDSPerformanceInsightsReadOnly` をアクセス許可セットまたはロールにアタッチすると、受取人は他のコンソール機能とともに Performance Insights を使用できます。

詳細については、「AWS マネージドポリシー: AmazonRDSPerformanceInsightsReadOnly」を参照してください。

AmazonRDSPerformanceInsightsFullAccess ポリシーの IAM プリンシパルへのアタッチ

`AmazonRDSPerformanceInsightsFullAccess` は Amazon RDS Performance Insights API のすべての操作へのアクセス許可を付与する AWS マネージドポリシーです。

`AmazonRDSPerformanceInsightsFullAccess` をアクセス許可セットまたはロールにアタッチすると、受取人は他のコンソール機能とともに Performance Insights を使用できます。

詳細については、「AWS 管理ポリシー: AmazonRDSPerformanceInsightsFullAccess」を参照してください。

Performance Insights 用のカスタム IAM ポリシーの作成

`AmazonRDSPerformanceInsightsReadOnly` または `AmazonRDSPerformanceInsightsFullAccess` ポリシーを持たないユーザーの場合、ユーザーマネージド IAM ポリシーを作成または変更することによって、Performance Insights へのアクセス許可を付与できます。ポリシーを IAM アクセス許可セットまたはロールにアタッチすると、受取人は Performance Insights を使用できます。

カスタムポリシーを作成するには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで、ポリシー を選択します。
3. [ポリシーの作成] を選択します。
4. [ポリシーの作成] ページで、[JSON] オプションを選択します。
5. 「AWS マネージドポリシーリファレンスガイド」の「JSON ポリシードキュメント」セクションに記載されている文字列をコピーして、AmazonRDSPerformanceInsightsReadOnly または AmazonRDSPerformanceInsightsFullAccess ポリシーに貼り付けます。
6. [ポリシーの確認] を選択します。
7. ポリシーの名前と (必要に応じて) 説明を入力し、[ポリシーの作成] を選択します。

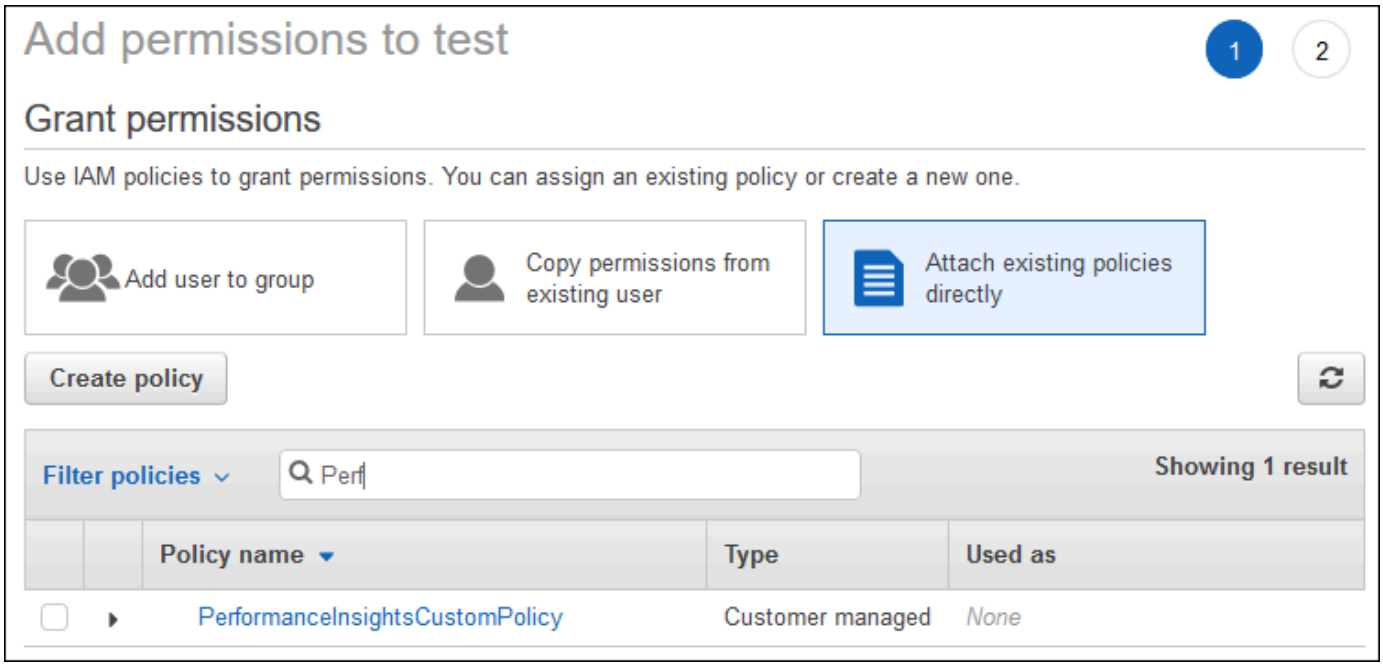
これで、そのポリシーをアクセス許可セットまたはロールにアタッチできます。次の手順では、この目的で使用できるユーザーが既に存在することを前提としています。

ポリシーをユーザーにアタッチするには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで [ユーザー] を選択します。
3. リストから存在するユーザーを 1 人選択します。

> **Important**
>
> Performance Insights を使用するには、カスタムポリシーのほかに別のポリシーで、Amazon RDS へのアクセスを許可されている必要があります。例えば、`AmazonRDSPerformanceInsightsReadOnly` 事前定義ポリシーで、ユーザーに Amazon RDS への読み取り専用アクセスを許可します。詳細については、「ポリシーを使用したアクセスの管理」を参照してください。

4. [Summary] ページで、[Add permissions] を選択します。
5. [Attach existing policies directly (既存のポリシーを直接アタッチする)] を選択します。検索を行う場合は、次の画像に示すようにポリシー名の初期の数文字を入力します。



6. ポリシーを選択し、[次へ: レビュー] を選択します。
7. [アクセス権限の追加] を選択します。

Performance Insights 用の AWS KMS ポリシーの設定

Performance Insights は、AWS KMS key を使用して機密データを暗号化します。API またはコンソールを通じて Performance Insights を有効にする場合は、次のいずれかを実行します。

- デフォルト AWS マネージドキー を選択します。

  Amazon RDS は、新しい DB インスタンスに AWS マネージドキー を使用します。Amazon RDS は、AWS アカウント に AWS マネージドキー を作成します。AWS アカウント には、AWS リージョン ごとに Amazon RDS の AWS マネージドキー が別々にあります。
- カスタマーマネージドキーを選択します。

  カスタマーマネージドキーを指定する場合、Performance Insights API を呼び出すアカウント内のユーザーは、KMS キーに対する `kms:Decrypt` および `kms:GenerateDataKey` アクセス許可が必要です。IAM ポリシーを使用して、これらのアクセス許可を設定できます。ただし、KMS キーポリシーを使用してこれらのアクセス許可を管理することをお勧めします。詳細については、

「AWS Key Management Service デベロッパーガイド」の「AWS KMS でのキーポリシー」を参照してください。

Example

次の例では、KMS キーポリシーにステートメントを追加する方法を示します。これらのステートメントは、Performance Insights へのアクセスを許可します。KMS キーの使用方法によっては、いくつかの制限を変更することもできます。ポリシーにステートメントを追加する前に、すべてのコメントを削除してください。

```
{
"Version" : "2012-10-17",
 "Id" : "your-policy",
 "Statement" : [ {
    //This represents a statement that currently exists in your policy.
 }
 ....,
 //Starting here, add new statement to your policy for Performance Insights.
 //We recommend that you add one new statement for every RDS instance
{
    "Sid" : "Allow viewing RDS Performance Insights",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            //One or more principals allowed to access Performance Insights
            "arn:aws:iam::444455556666:role/Role1"
        ]
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "*",
    "Condition" : {
        "StringEquals" : {
            //Restrict access to only RDS APIs (including Performance Insights).
            //Replace region with your AWS Region.
            //For example, specify us-west-2.
            "kms:ViaService" : "rds.region.amazonaws.com"
        },
        "ForAnyValue:StringEquals": {
            //Restrict access to only data encrypted by Performance Insights.
```

```
            "kms:EncryptionContext:aws:pi:service": "rds",
            "kms:EncryptionContext:service": "pi",

            //Restrict access to a specific RDS instance.
            //The value is a DbiResourceId.
           "kms:EncryptionContext:aws:rds:db-id": "db-AAAAABBBBBCCCCDDDDDEEEEE"
        }
    }
}
```

Performance Insights が AWS KMS カスタマー管理キーを使用する方法

Performance Insights は、カスタマー管理型のキー (CMK) を使用して機密データを暗号化します。Performance Insights を有効にすると、API を介して AWS KMS キーを提供できます。Performance Insights は、このキーの KMS 権限を作成します。キーを使用して、機密データを処理するために必要な操作を実行します。機密データには、ユーザー、データベース、アプリケーション、SQL クエリテキストなどのフィールドが含まれます。Performance Insights により、データは保存中も転送中も暗号化されたままになります。

Performance Insights IAM と AWS KMS の連携

IAM は特定の API にアクセス権限を付与します。Performance Insights には以下のパブリック API があり、IAM ポリシーを使用して制限できます。

- `DescribeDimensionKeys`
- `GetDimensionKeyDetails`
- `GetResourceMetadata`
- `GetResourceMetrics`
- `ListAvailableResourceDimensions`
- `ListAvailableResourceMetrics`

機密データを取得するには、以下の API リクエストを使用できます。

- `DescribeDimensionKeys`
- `GetDimensionKeyDetails`
- `GetResourceMetrics`

API を使用して機密データを取得する場合、Performance Insights は呼び出し元の認証情報を利用します。このチェックにより、機密データへのアクセスが KMS キーにアクセスできるユーザーに限定されます。

これらの API を呼び出すときは、IAM ポリシーを通じて API を呼び出す権限と、AWS KMS キーポリシーを通じて kms:decrypt アクションを呼び出す権限が必要となります。

GetResourceMetrics API は、機密データと非機密データの両方を返すことができます。リクエストパラメータにより、レスポンスに機密データを含めるかどうかが決定されます。リクエストのフィルターパラメーターまたは group-by パラメーターのいずれかに機密ディメンションが含まれている場合、API は機密データを返します。

GetResourceMetrics API で使用できるディメンションの詳細については、DimensionGroup を参照してください。

Example 例

次の例では、db.user グループの機密データをリクエストしています:

```
POST / HTTP/1.1
Host: <Hostname>
Accept-Encoding: identity
X-Amz-Target: PerformanceInsightsv20180227.GetResourceMetrics
Content-Type: application/x-amz-json-1.1
User-Agent: <UserAgentString>
X-Amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
Content-Length: <PayloadSizeBytes>
{
  "ServiceType": "RDS",
  "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
  "MetricQueries": [
    {
      "Metric": "db.load.avg",
      "GroupBy": {
        "Group": "db.user",
        "Limit": 2
      }
    }
  ],
  "StartTime": 1693872000,
```

```
  "EndTime": 1694044800,
  "PeriodInSeconds": 86400
}
```

Example

次の例では、db.load.avg メトリクスの非機密データをリクエストしています:

```
POST / HTTP/1.1
Host: <Hostname>
Accept-Encoding: identity
X-Amz-Target: PerformanceInsightsv20180227.GetResourceMetrics
Content-Type: application/x-amz-json-1.1
User-Agent: <UserAgentString>
X-Amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
Content-Length: <PayloadSizeBytes>
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "MetricQueries": [
        {
            "Metric": "db.load.avg"
        }
    ],
    "StartTime": 1693872000,
    "EndTime": 1694044800,
    "PeriodInSeconds": 86400
}
```

Performance Insights への詳細なアクセス許可の付与

詳細なアクセスコントロールを使用して、Performance Insights データへの追加のアクセスコントロールを適用することができます。このアクセスコントロールは、Performance Insights の GetResourceMetrics、DescribeDimensionKeys、および GetDimensionKeyDetails アクションの個々のディメンションへのアクセスを許可または拒否できます。詳細なアクセスコントロールを使用するには、条件キーを使用して IAM ポリシーでディメンションを指定します。アクセスの評価は、IAM ポリシーの評価ロジックに基づきます。詳細については、「 IAM ユーザーガイド」の

「ポリシーの評価論理」を参照してください。IAM ポリシーステートメントでディメンションが指定されていない場合、ステートメントは指定されたアクションのすべてのディメンションへのアクセスをコントロールします。使用可能なディメンションのリストについては、「DimensionGroup」を参照してください。

認証情報でアクセスが許可されているディメンションを確認するには、ListAvailableResourceDimensions の AuthorizedActions パラメータを使用してアクションを指定します。AuthorizedActions の許容値は、次のとおりです。

- GetResourceMetrics
- DescribeDimensionKeys
- GetDimensionKeyDetails

例えば、AuthorizedActions パラメータに GetResourceMetrics を指定すると、ListAvailableResourceDimensions は GetResourceMetrics アクションがアクセスを許可されているディメンションのリストを返します。AuthorizedActions パラメータで複数のアクションを指定すると、ListAvailableResourceDimensions はそれらのアクションがアクセスを許可されているディメンションの交差を返します。

Example

次の例では、GetResourceMetrics および DescribeDimensionKeys アクションに指定されたディメンションへのアクセスを付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowToDiscoverDimensions",
            "Effect": "Allow",
            "Action": [
                "pi:ListAvailableResourceDimensions"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ]
        },
        {
            "Sid": "SingleAllow",
```

```
            "Effect": "Allow",
            "Action": [
                "pi:GetResourceMetrics",
                "pi:DescribeDimensionKeys"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    // only these dimensions are allowed. Dimensions not included in
                    // a policy with "Allow" effect will be denied
                    "pi:Dimensions": [
                        "db.sql_tokenized.id",
                        "db.sql_tokenized.statement"
                    ]
                }
            }
        }


    ]
}
```

リクエストされたディメンションのレスポンスを次に示します。

```
 // ListAvailableResourceDimensions API
// Request
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "Metrics": [ "db.load" ],
    "AuthorizedActions": ["DescribeDimensionKeys"]
}

// Response
{
    "MetricDimensions": [ {
        "Metric": "db.load",
        "Groups": [
```

```
            {
                "Group": "db.sql_tokenized",
                "Dimensions": [
                    { "Identifier": "db.sql_tokenized.id" },
                  //  { "Identifier": "db.sql_tokenized.db_id" }, // not included
 because not allows in the IAM Policy
                    { "Identifier": "db.sql_tokenized.statement" }
                ]
            }

        ] }
    ]
}
```

次の例は、ディメンションに対する 1 つの許可アクセスと 2 つの拒否アクセスを指定します。

```
{
    "Version": "2012-10-17",
    "Statement": [
          {
            "Sid": "AllowToDiscoverDimensions",
            "Effect": "Allow",
            "Action": [
                "pi:ListAvailableResourceDimensions"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ]
          },

          {
            "Sid": "O01AllowAllWithoutSpecifyingDimensions",
            "Effect": "Allow",
            "Action": [
                "pi:GetResourceMetrics",
                "pi:DescribeDimensionKeys"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ]
        },
```

```
        {
            "Sid": "O01DenyAppDimensionForAll",
            "Effect": "Deny",
            "Action": [
                "pi:GetResourceMetrics",
                "pi:DescribeDimensionKeys"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "pi:Dimensions": [
                        "db.application.name"
                    ]
                }
            }
        },

        {
            "Sid": "O01DenySQLForGetResourceMetrics",
            "Effect": "Deny",
            "Action": [
                "pi:GetResourceMetrics"
            ],
            "Resource": [
                "arn:aws:pi:us-east-1:123456789012:metrics/rds/db-
ABC1DEFGHIJKL2MNOPQRSTUV3W"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "pi:Dimensions": [
                        "db.sql_tokenized.statement"
                    ]
                }
            }
        }
    ]
}
```

リクエストされたディメンションのレスポンスを次に示します。

```
   // ListAvailableResourceDimensions API
// Request
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "Metrics": [ "db.load" ],
    "AuthorizedActions": ["GetResourceMetrics"]
}

// Response
{
    "MetricDimensions": [ {
        "Metric": "db.load",
        "Groups": [
            {
                "Group": "db.application",
                "Dimensions": [

                  // removed from response because denied by the IAM Policy
                  //  { "Identifier": "db.application.name" }
                ]
            },
            {
                "Group": "db.sql_tokenized",
                "Dimensions": [
                    { "Identifier": "db.sql_tokenized.id" },
                    { "Identifier": "db.sql_tokenized.db_id" },

                  // removed from response because denied by the IAM Policy
                  //  { "Identifier": "db.sql_tokenized.statement" }
                ]
            },
            ...
        ] }
    ]
}
```

```
// ListAvailableResourceDimensions API
// Request
```

```
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "Metrics": [ "db.load" ],
    "AuthorizedActions": ["DescribeDimensionKeys"]
}

// Response
{
    "MetricDimensions": [ {
        "Metric": "db.load",
        "Groups": [
            {
                "Group": "db.application",
                "Dimensions": [
                  // removed from response because denied by the IAM Policy
                  //  { "Identifier": "db.application.name" }
                ]
            },
            {
                "Group": "db.sql_tokenized",
                "Dimensions": [
                    { "Identifier": "db.sql_tokenized.id" },
                    { "Identifier": "db.sql_tokenized.db_id" },

                  // allowed for DescribeDimensionKeys because our IAM Policy
                  // denies it only for GetResourceMetrics
                    { "Identifier": "db.sql_tokenized.statement" }
                ]
            },
            ...
        ] }
    ]
}
```

Performance Insights ダッシュボードを使用してメトリクスを分析する

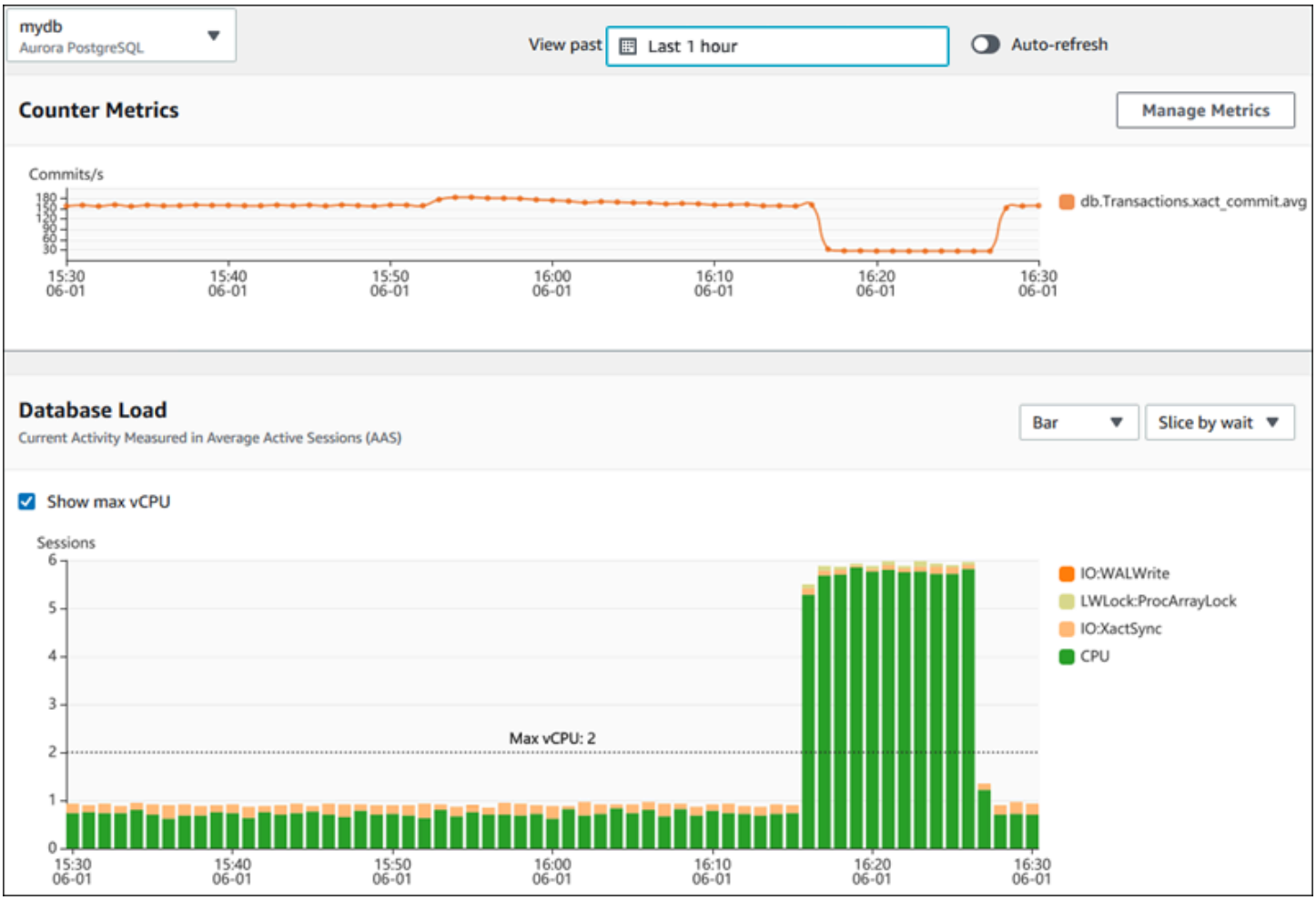
Performance Insights ダッシュボードには、パフォーマンスの問題を分析し、解決するのに役立つ、データベースのパフォーマンス情報が含まれます。ダッシュボードのメインページで、データベース負荷に関する情報を確認できます。待機イベントや SQL などのディメンションによって、DB のロードを「スライス」することが可能です。

Performance Insights ダッシュボード

- Performance Insights ダッシュボードの概要
- Performance Insights ダッシュボードにアクセスする
- 待機イベントによる DB 負荷の分析
- 一定期間のデータベースパフォーマンスの分析
- Performance Insights ダッシュボードのクエリの分析

Performance Insights ダッシュボードの概要

ダッシュボードは、Performance Insights を操作する最も簡単な方法です。次に、MySQL DB インスタンスのダッシュボードの例を示します。



トピック

- 時間範囲フィルター
- カウンターメトリクスグラフ

- データベースロードのグラフ
- 上位のディメンションテーブル

時間範囲フィルター

デフォルトでは、Performance Insights ダッシュボードには過去 60 分間の DB ロードが表示されます。この範囲は、最短で 5 分、最長で 2 年まで調整することができます。カスタム相対範囲を選択することもできます。

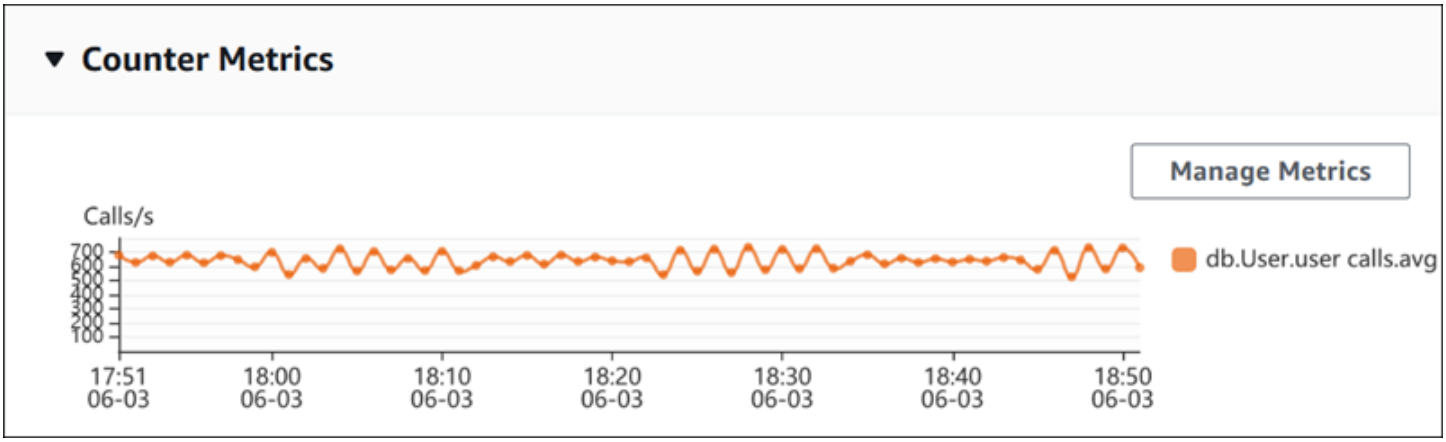
開始日時と終了日時の絶対範囲を選択できます。次の例は、22/4/11 の午前 0 時から 22/4/14 の午後 11 時 59 分までの時間範囲を示しています。

カウンターメトリクスグラフ

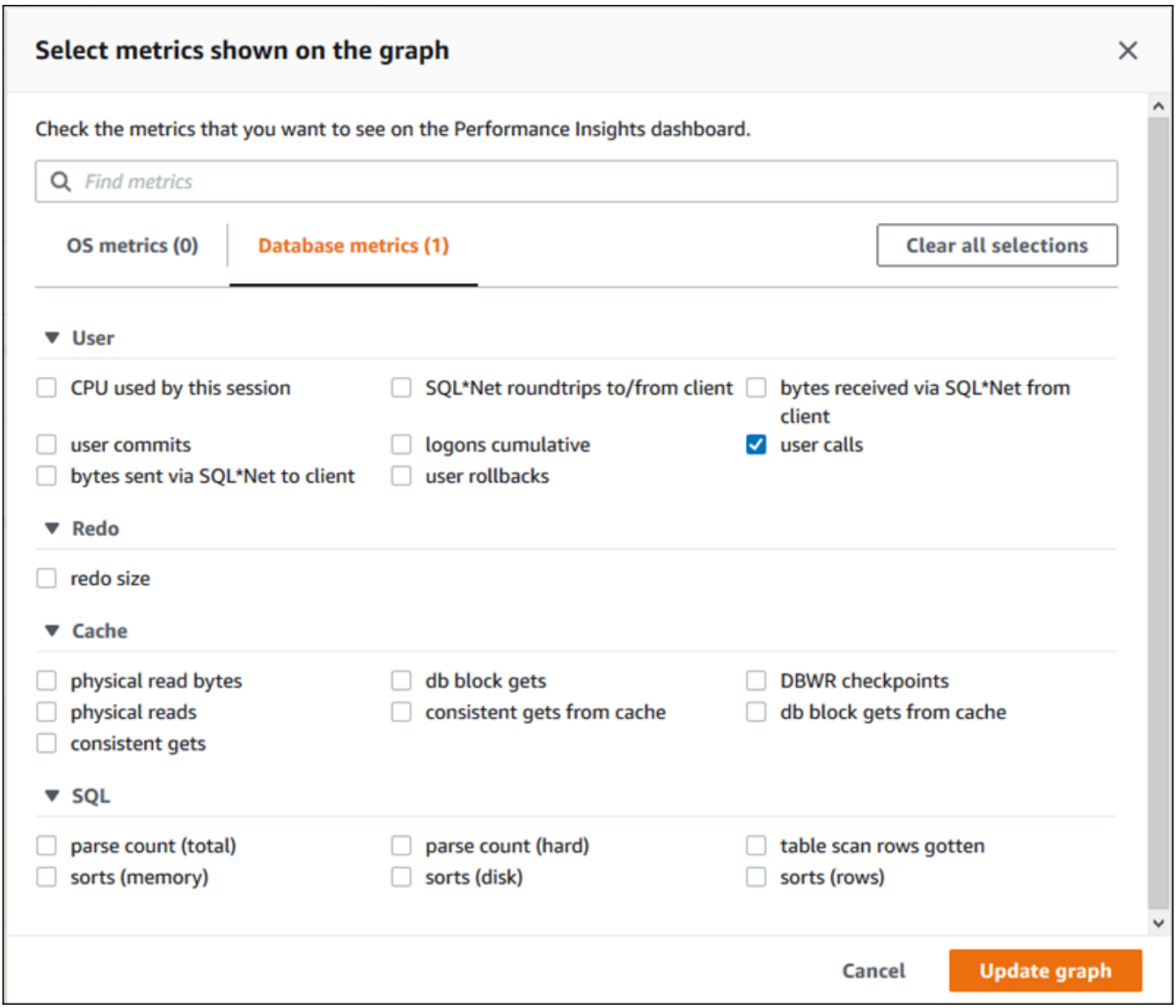
カウンターメトリクスを使用すると、Performance Insights ダッシュボードをカスタマイズして最大 10 個の追加グラフを含めることができます。これらのグラフは、数十種類のオペレーティングシステムとデータベースのパフォーマンスメトリクスの一部を示しています。この情報をデータベース負荷と関連付けることで、パフォーマンスの問題を特定して分析できます。

カウンターメトリクスグラフはパフォーマンスカウンターのデータを表示します。デフォルトのメトリクスは DB エンジンによって異なります。

- Aurora MySQL - `db.SQL.Innodb_rows_read.avg`
- Aurora PostgreSQL - `db.Transactions.xact_commit.avg`



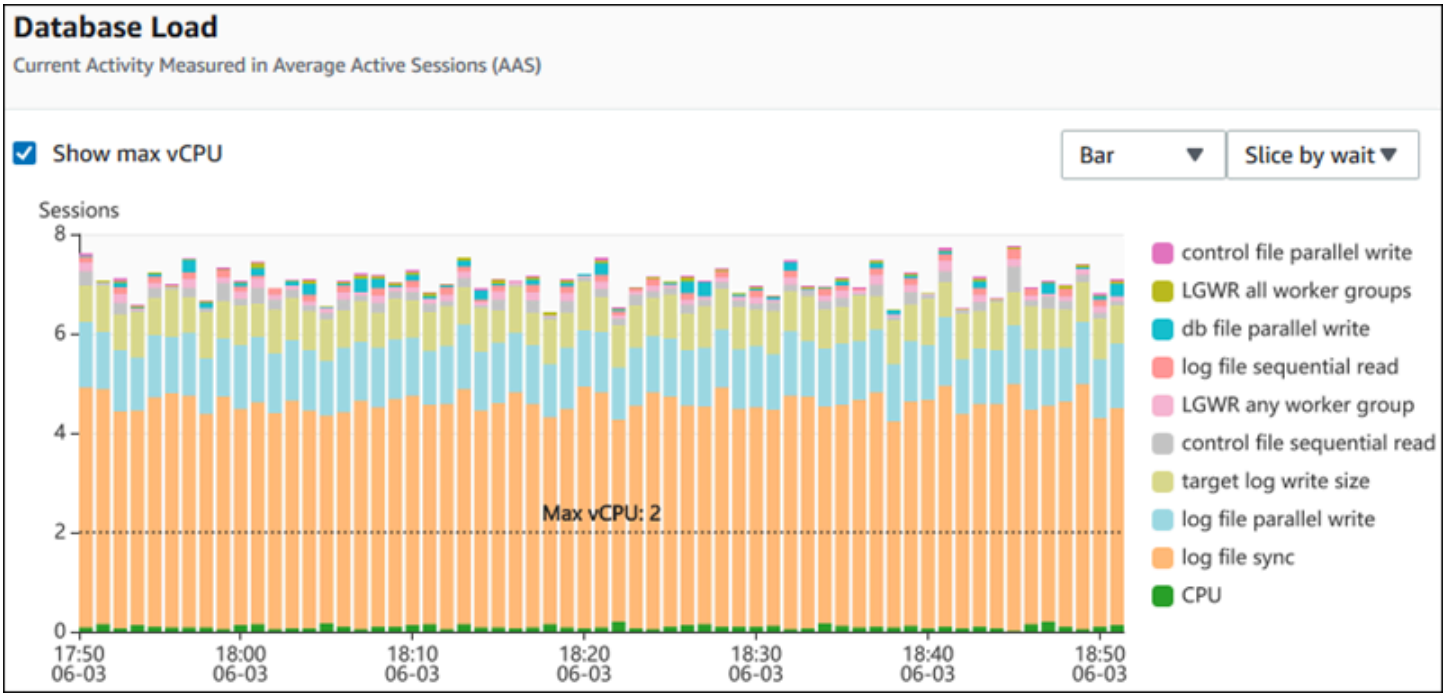
「メトリクスの管理」 を選択して、パフォーマンスカウンターを変更します。以下のスクリーンショットに示すように、複数の OS メトリクスまたはデータベースメトリクスを選択できます。メトリクスの詳細を表示するには、メトリクス名にカーソルを合わせます。



各 DB エンジンで追加できるカウンターメトリクスの詳細については、「Performance Insights カウンターメトリクス」を参照してください。

データベースロードのグラフ

データベースロードは、データベースアクティビティと DB インスタンス容量の比較結果が 最大 vCPU の折れ線グラフとして表示されます。デフォルトでは、折れ線グラフは DB ロードを単位時間あたりの平均アクティブセッションで表します。DB ロードは、待機状態でスライス (グループ化) されます。

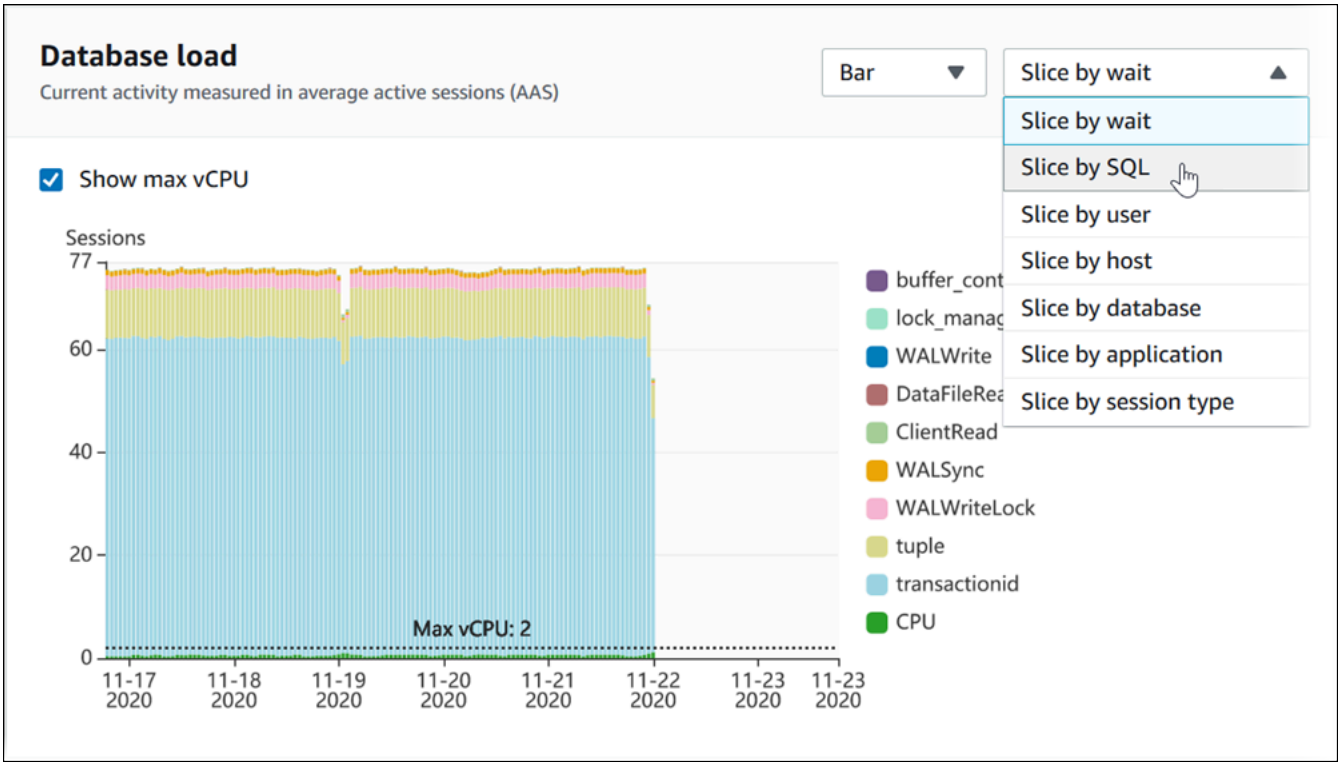


ディメンションでスライスされた DB の負荷

サポートされているディメンション別にグループ化された、アクティブなセッションとして負荷を表示するように選択できます。次の表に、各エンジンでサポートされているディメンションを示します。

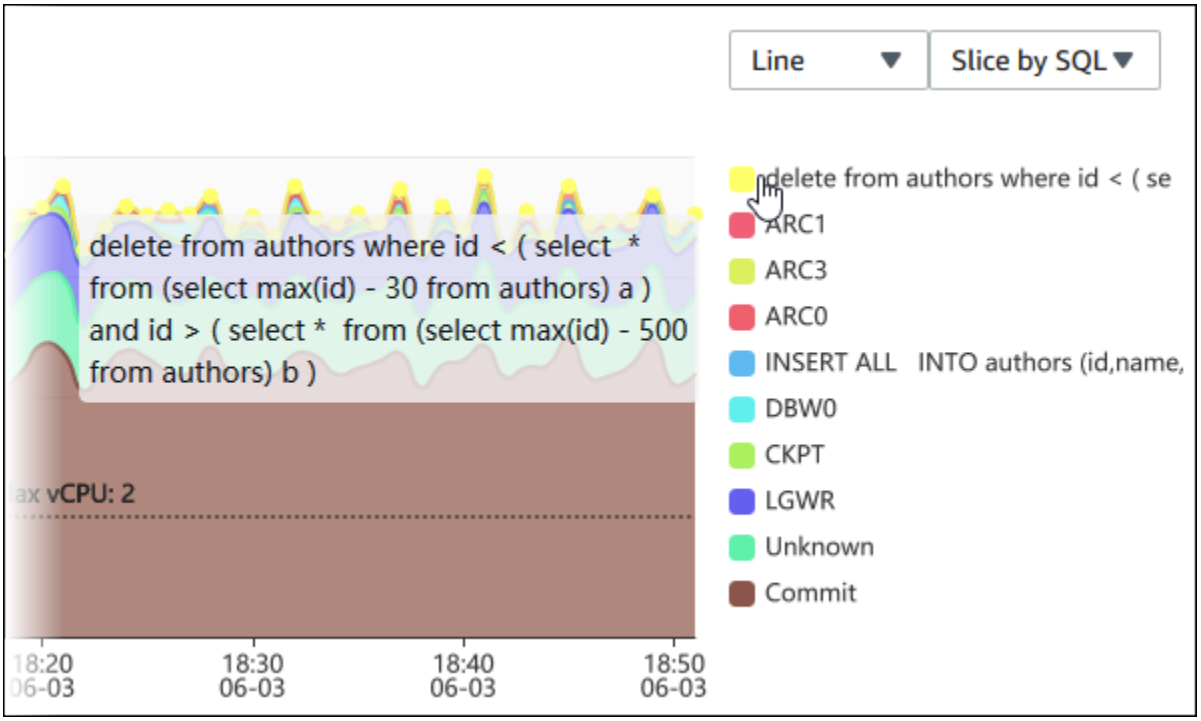
ディメンション	Aurora PostgreSQL	Aurora MySQL
ホスト	はい	はい
SQL	はい	はい
ユーザー	はい	はい
待機	はい	はい
アプリケーション	はい	いいえ
データベース	はい	はい
セッションタイプ	はい	いいえ

次の図に、PostgreSQL DB インスタンスのディメンションを示します。

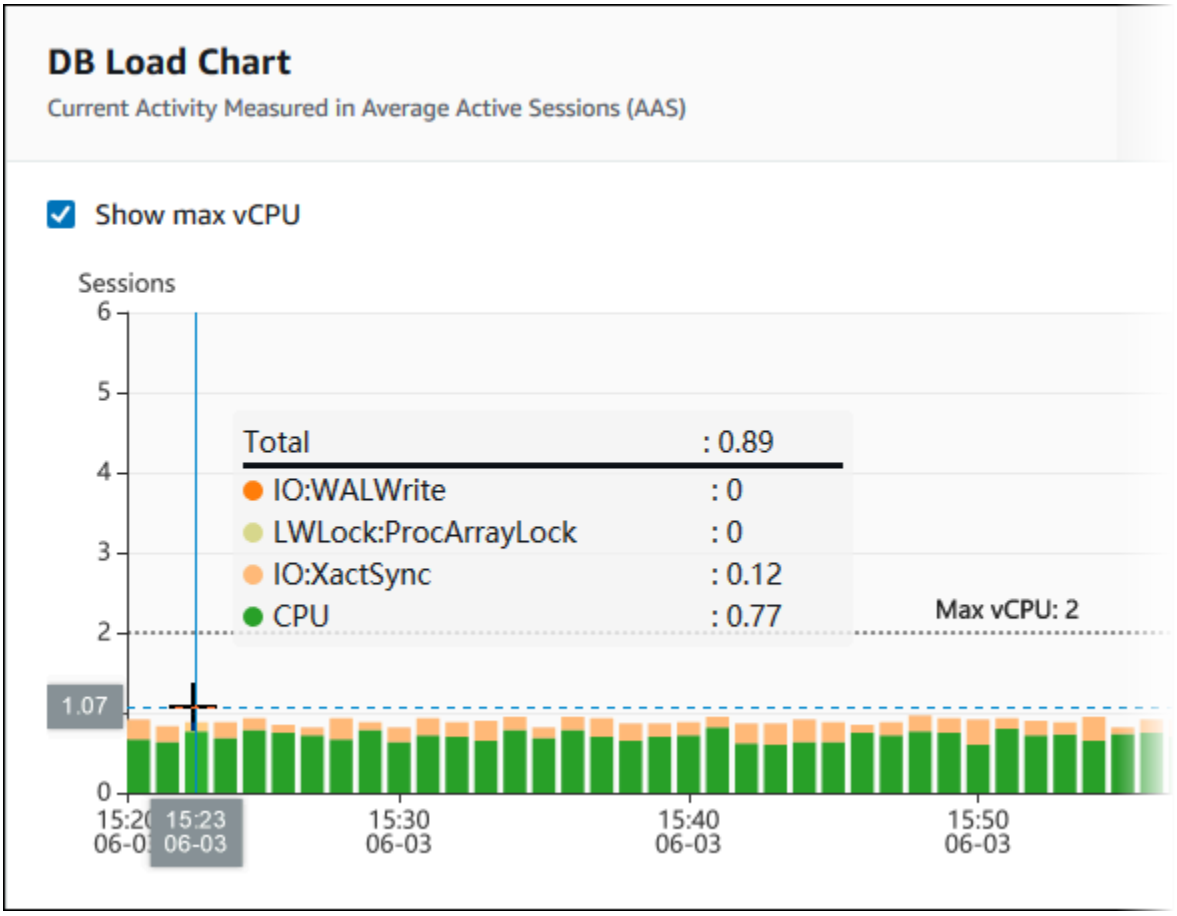


ディメンション項目に関する DB ロードの詳細

ディメンション内の DB 負荷項目の詳細を表示するには、項目名にカーソルを合わせます。次の図は、SQL ステートメントの詳細を示しています。

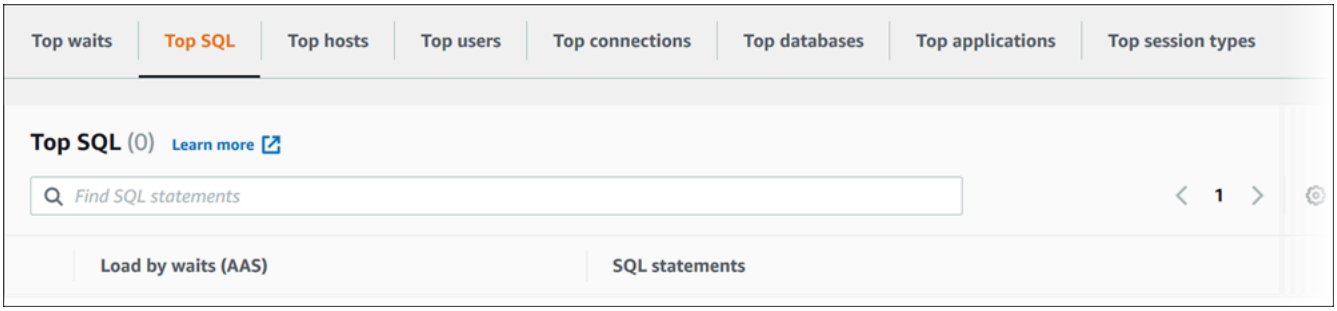


凡例で選択した期間に対する項目の詳細を表示するには、その項目にカーソルを合わせます。



上位のディメンションテーブル

上位ディメンションテーブルは、DB ロードを異なる次元でスライスします。ディメンションとは、DB ロードの異なる特性についてのカテゴリまたは「スライス化」のことです。ディメンションが SQL の場合、上位の SQLは、DB ロードに最も貢献している SQL ステートメントを表示します。



以下のディメンションタブのいずれかを選択します。

タブ	説明	サポートされているエンジン
上位の SQL	現在実行中の SQL ステートメント	すべて
上位待機	データベースバックエンドが待っているイベント	すべて
上位ホスト	接続されているクライアントのホスト名	すべて
上位ユーザー	データベースにログインしているユーザー	すべて
プロキシが接続しているデータベースユーザーの名前		
上位アプリケーション	データベースに接続されたアプリケーションの名前。	Aurora PostgreSQL と SQL Server のみ
上位セッションタイプ	現在のセッションのタイプ	Aurora PostgreSQL のみ

[上位の SQL] タブを使用してクエリを分析する方法を学習するには、「[トップ SQL] タブの概要」を参照してください。

Performance Insights ダッシュボードにアクセスする

Amazon RDS のPerformance Insights ダッシュボードでは、Performance Insights と CloudWatch メトリクスの統合ビューが提供されます。

Performance Insights ダッシュボードにアクセスするには、以下の手順を使用します。

AWS マネジメントコンソールで Performance Insights ダッシュボードを表示するには

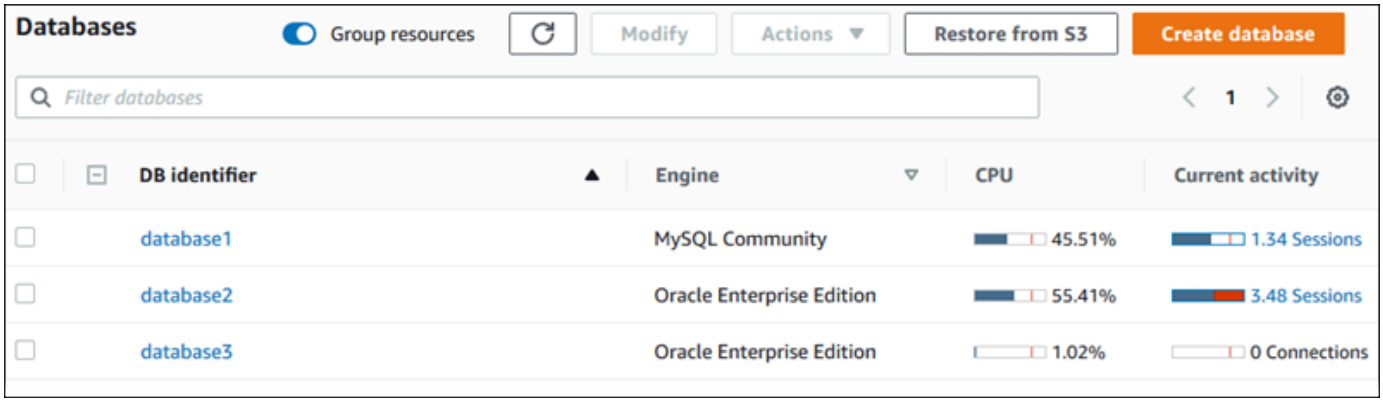
1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。
4. 表示されたウィンドウでデフォルトのモニタリングビューを選択します。

- [Performance Insights と CloudWatch メトリクスビュー (新規)] オプションを選択し、[続行] を選択してPerformance Insights と CloudWatch メトリクスを表示します。
- [Performance Insights ビュー] オプションを選択し、レガシーモニタリングビューに対して [続行] を選択します。その後、この手順に進みます。

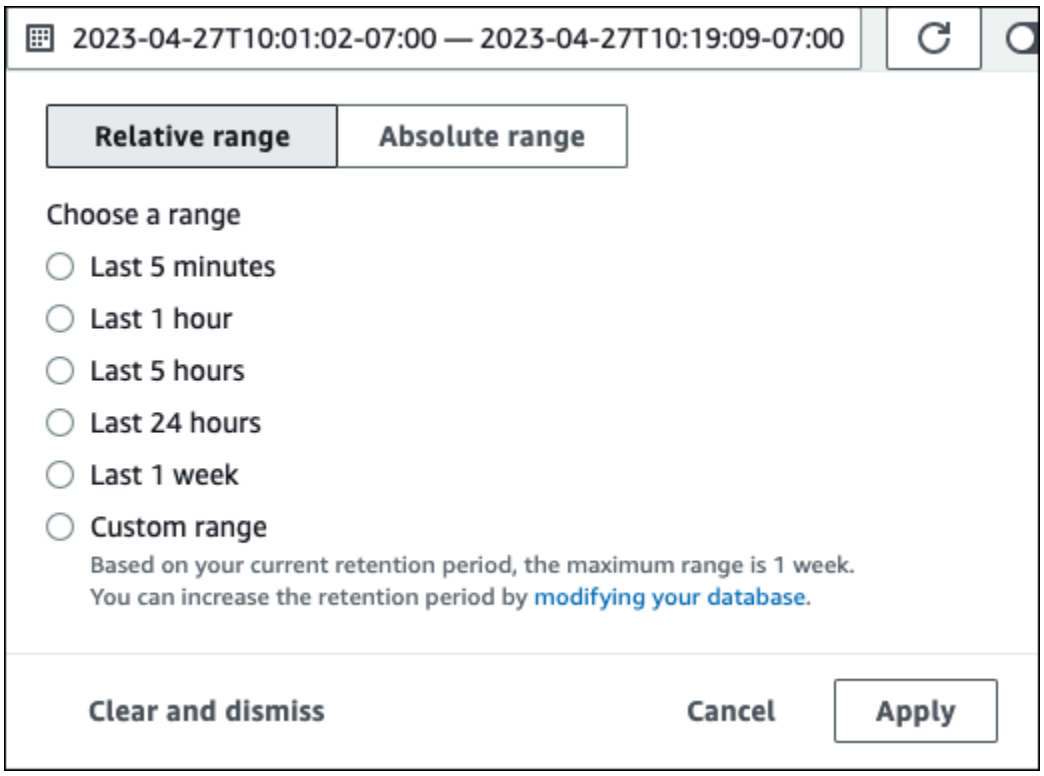
> **Note**
>
> このビューは 2023 年 12 月 15 日に廃止されます。

この DB インスタンスに Performance Insights ダッシュボードが表示されます。

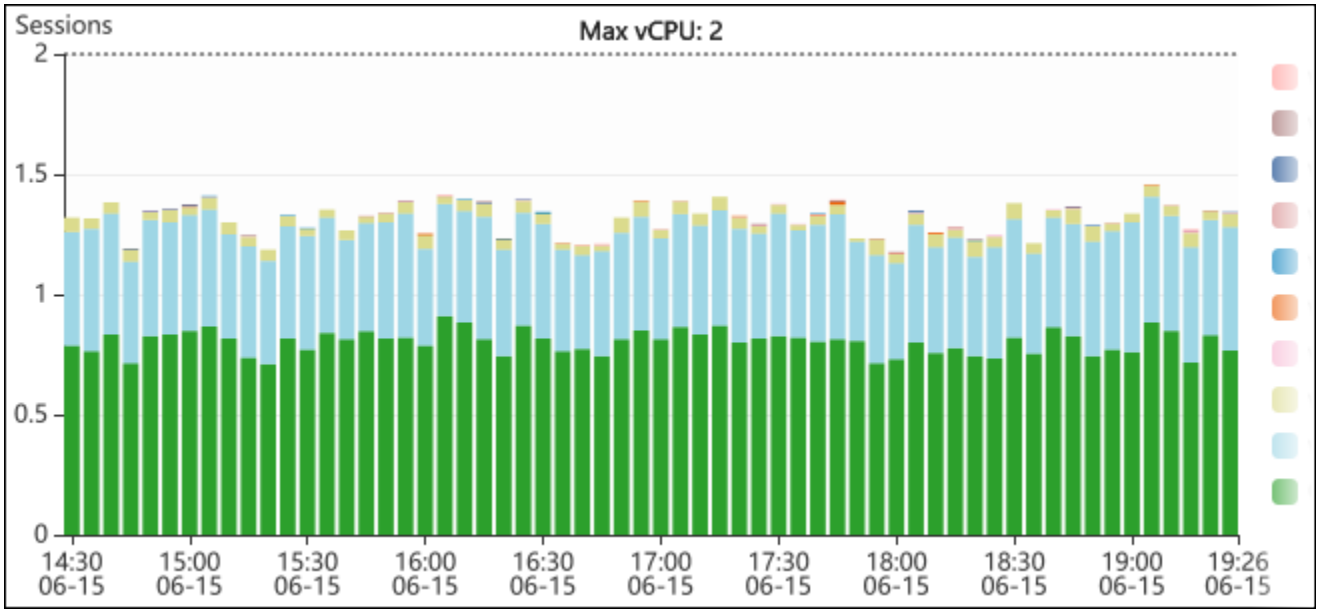
Performance Insights を有効にした DB インスタンスでは、DB インスタンスのリストで [セッション] 項目を選択してダッシュボードにアクセスすることもできます。[現在のアクティビティ] の [セッション] 項目には、直近 5 分間におけるアクティブなセッションの平均データベース負荷が表示されます。負荷はバーでグラフィカルに示されます。バーが空の場合、DB インスタンスはアイドル状態です。負荷が増加すると、バーが青色で塗りつぶされます。負荷が DB インスタンスクラスのおける仮想 CPU (vCPU) の数を超えると、バーが赤色になり、ボトルネックとなる可能性があることが示されます。



5. (オプション) 右上の日付または時間範囲を選択し、別の相対時間間隔または絶対時間間隔を指定します。これで、期間を指定して、データベースパフォーマンス分析レポートを生成できます。レポートには、特定されたインサイトと推奨事項が記載されています。詳細については、「パフォーマンス分析レポートの作成」を参照してください。

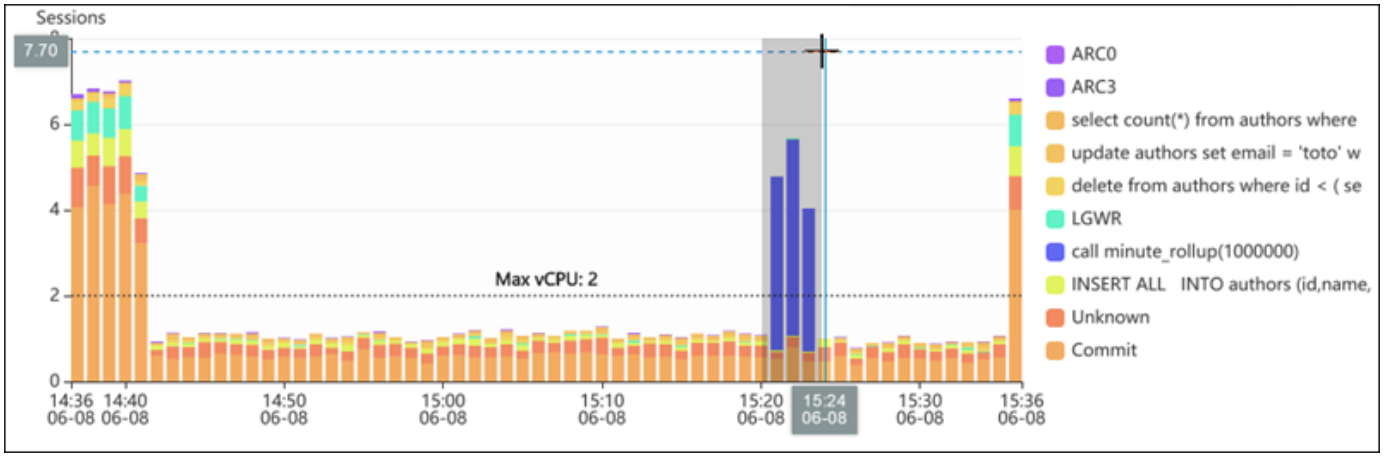


以下のスクリーンショットでは、DB 負荷の間隔は 5 時間です。

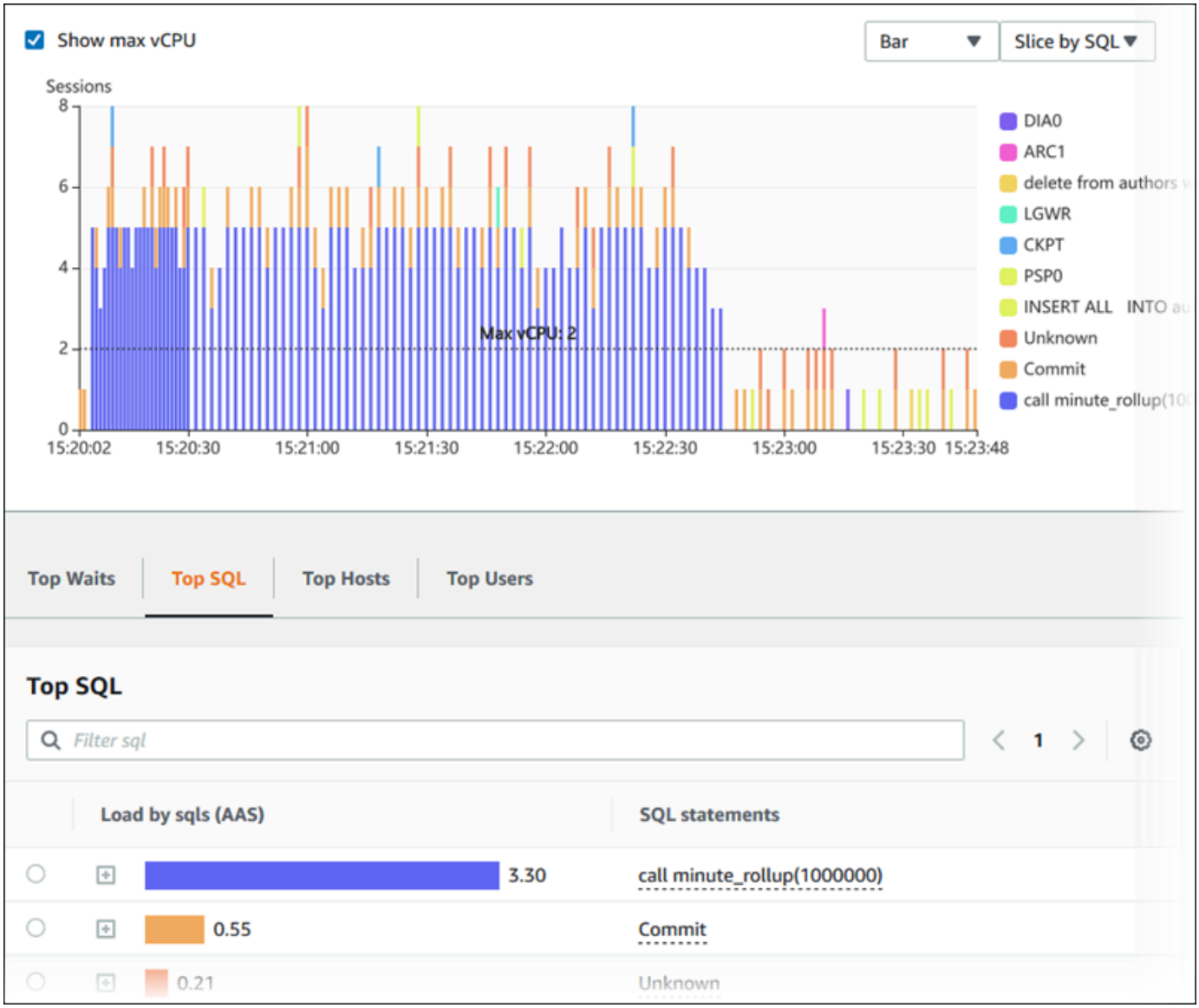


6. (オプション) DB ロードグラフの一部を拡大表示するには、スタート時間を選択し、目的の期間の最後までドラッグします。

   選択した領域が DB ロードチャートで強調表示されます。



マウスを離すと、選択した AWS リージョンの DB ロードグラフが拡大表示され、上位ディメンションのテーブルが再計算されます。



7. (オプション) データを自動的に更新するには、[自動更新] を選択します。

Performance Insights ダッシュボードが自動的に新しいデータで更新されます。更新の頻度は、表示されるデータの量によって異なります。

- 「5 分」は 10 秒ごとに更新されます。
- 「1 時間」は 5 分ごとに更新されます。
- 「5 時間」は 5 分ごとに更新されます。
- 「24 時間」は 30 分ごとに更新されます。
- 「1 週間」は 1 日ごとに更新されます。

- 「1 か月」は 1 日ごとに更新されます。

待機イベントによる DB 負荷の分析

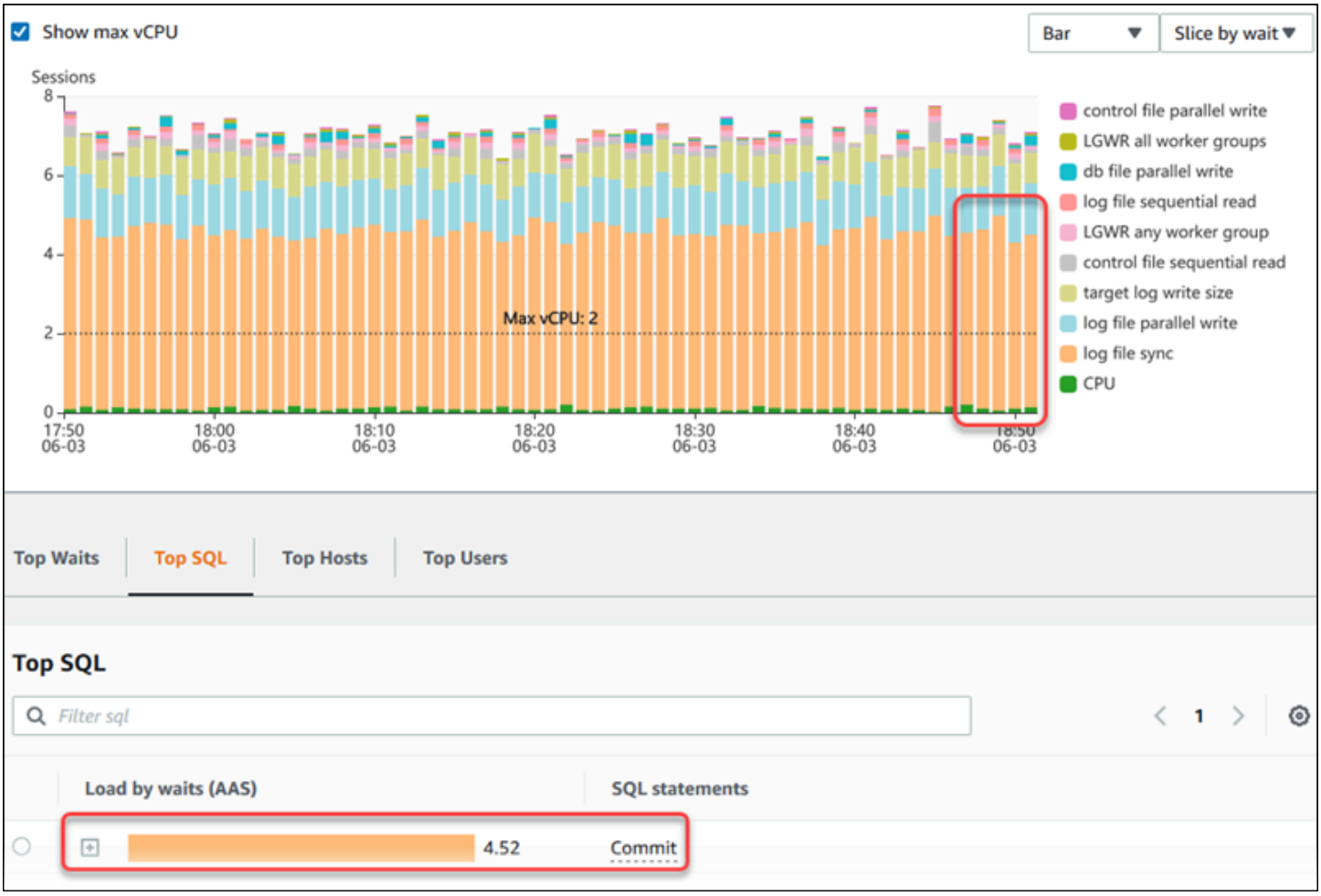
データベースロード のグラフにボトルネックが表示される場合、ロードの発生源を確認できます。これを実行するには、データベースロードグラフ下にある[上位ロード項目] テーブルを参照してください。SQL クエリやユーザーのような特定の項目を選択すると、その項目をドリルダウンして詳細を表示できます。

待機および上位 SQL クエリによってグループ分けされた DB 負荷は、Performance Insights ダッシュボードのデフォルトビューです。通常、この組み合わせは、パフォーマンス問題に関する最も正しい情報を提供します。待機でグループ化された DB 負荷は、データベースにリソースまたは同時のボトルネックがあるかどうかを示します。この場合、上位負荷項目のテーブルの SQL タブには、どのクエリがその負荷をかけているかが表示されます。

パフォーマンスの問題を診断するための一般的なワークフローは次のとおりです。

1. 「データベースロード」 グラフを確認し、最大 CPU ラインを超えているデータベースロードのインシデントがあるかどうかを確認します。
2. ある場合は、「データベースロード」 グラフを確認して、どの待機状態 (複数) が主に原因であるかを特定します。
3. 上位の負荷項目テーブルの SQL タブが待機状態に最も影響しているクエリを確認することによって、ロードを引き起こすダイジェストクエリを特定します。これらは [DB Load by Wait] 列で識別できます。
4. [SQL] タブでこれらのダイジェストクエリの 1 つを選択して展開し、構成されている子クエリを確認します。

例えば、以下のダッシュボードで、[ログファイルの同期] の待機はほとんどの DB 負荷の主な原因となっています。[LGWR すべてのワーカーグループ] の待機も高くなっています。「上位の SQL」 グラフでは、「ログファイルの同期」 の待機の発生元として、頻繁な`COMMIT` ステートメントが示されています。この場合、コミット頻度を下げると、DB 負荷が軽減されます。



一定期間のデータベースパフォーマンスの分析

一定期間のパフォーマンス分析レポートを作成して、オンデマンド分析でデータベースのパフォーマンスを分析します。パフォーマンス分析レポートで、リソースのボトルネックや DB インスタンスでのクエリの変更などのパフォーマンスの問題を確認します。Performance Insights ダッシュボードでは、期間を選択してパフォーマンス分析レポートを作成できます。レポートに 1 つ以上のタグを追加することもできます。

この機能を使用するには、有料利用枠の保持期間を使用している必要があります。詳細については、「Performance Insights の料金とデータ保持」を参照してください。

レポートは [パフォーマンス分析レポート - 新規] タブで選択して表示できます。レポートには、インサイト、関連するメトリクス、およびパフォーマンス問題を解決するための推奨事項が含まれています。レポートは、Performance Insights 保存期間中は表示できます。

レポート分析期間の開始時刻が保存期間外の場合、レポートは削除されます。保存期間が終了する前にレポートを削除することもできます。

パフォーマンス問題を検出し、DB インスタンスの分析レポートを生成するには、Performance Insights を有効にする必要があります。Performance Insights をオンにする方法の詳細については、「Performance Insights の有効化と無効化」を参照してください。

この機能のリージョン、DB エンジン、およびインスタンスクラスのサポート情報については、「Amazon Aurora DB エンジン、リージョン、およびインスタンスクラスでサポートされている Performance Insights 機能」を参照してください。

パフォーマンス分析レポートの作成

Performance Insights ダッシュボードで特定期間のパフォーマンス分析レポートを作成できます。期間を選択し、分析レポートに 1 つ以上のタグを追加できます。

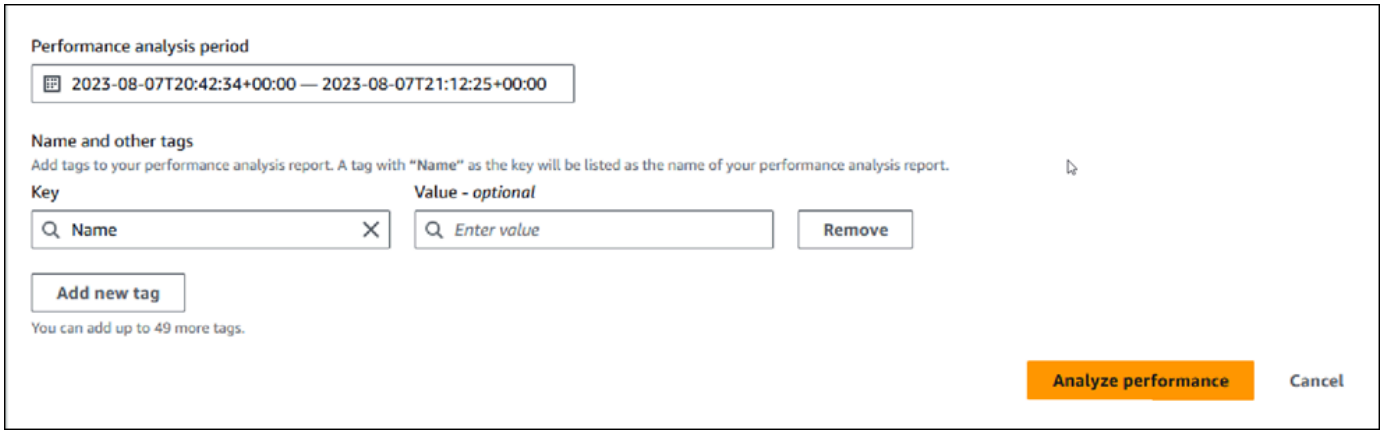
分析期間は 5 分から 6 日間までに設定できます。分析開始時刻の前に、少なくとも 24 時間のパフォーマンスデータが必要です。

一定期間のパフォーマンス分析レポートを作成するには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。

   この DB インスタンスに Performance Insights ダッシュボードが表示されます。
4. ダッシュボードの [データベース負荷] セクションで [パフォーマンスを分析] を選択します。

   期間を設定するフィールドと、パフォーマンス分析レポートに 1 つ以上のタグを追加するフィールドが表示されます。

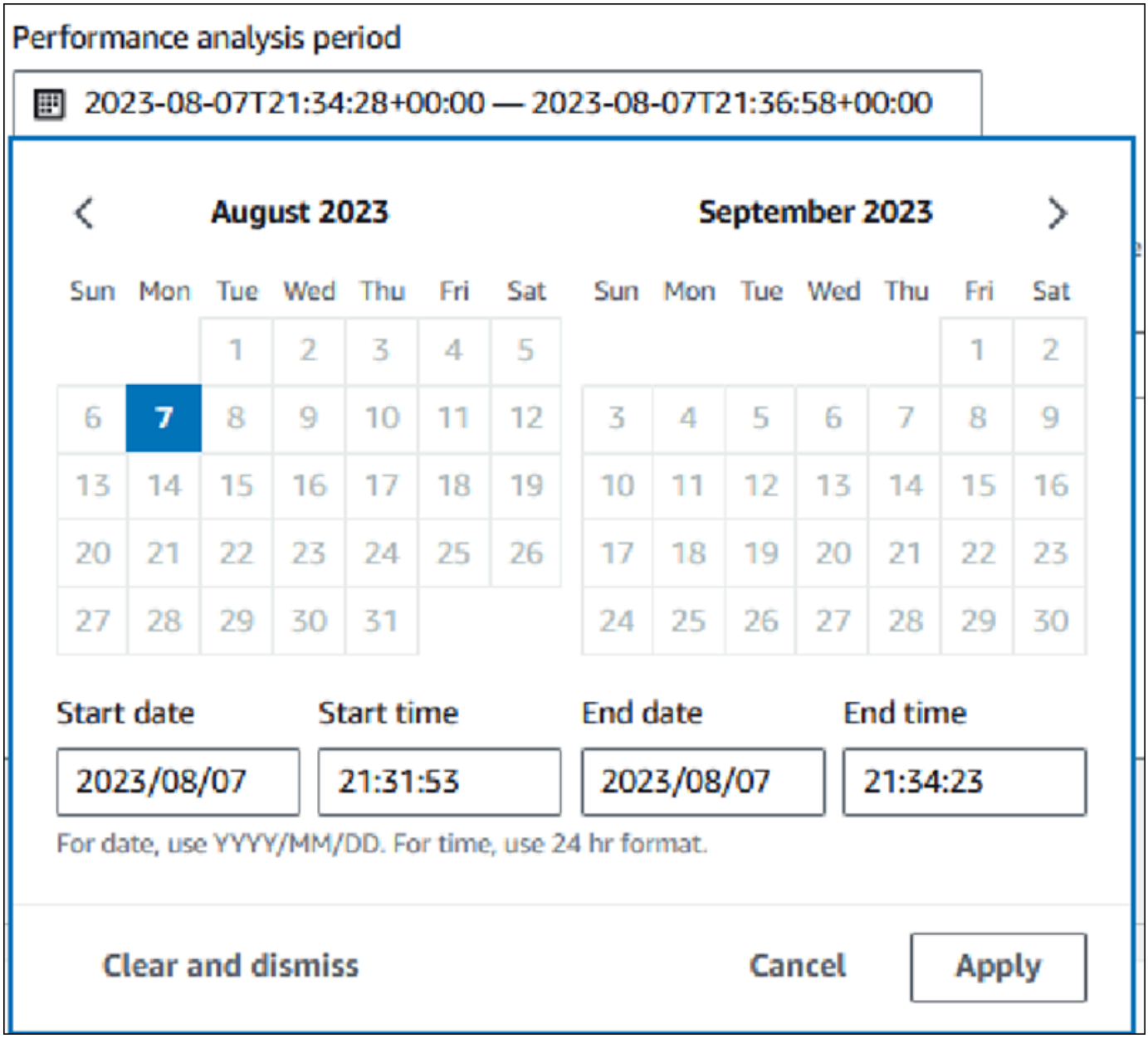


5. 期間を選択します。右上の [相対範囲] または [絶対範囲] で期間を設定した場合は、この期間内の分析レポートの日付と時刻のみを入力または選択できます。この期間以外の分析期間を選択すると、エラーメッセージが表示されます。

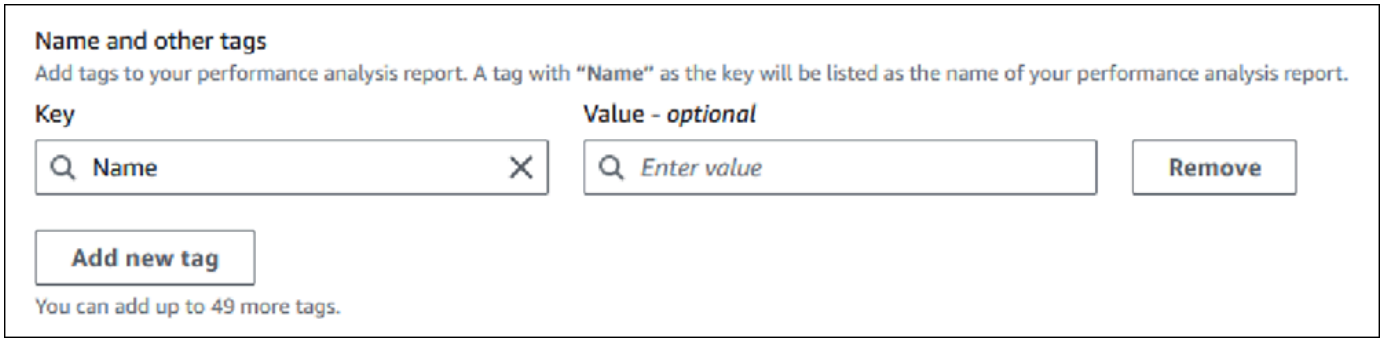
   期間を設定するには、次のいずれかを行います。

   - DB 負荷チャートのいずれかのスライダーを押してドラッグします。

     [パフォーマンス分析期間] ボックスに選択した期間が表示され、DB 負荷チャートに選択した期間が強調表示されます。
   - [パフォーマンス分析期間] ボックスで、[開始日]、[開始時間]、[終了日]、および [終了時間] を選択します。



6. (オプション) [キー] と [値- オプション] を入力して、レポートにタグを追加します。



7. [パフォーマンスを分析] を選択します。

   バナーに、レポート生成が成功したか失敗したかを示すメッセージが表示されます。メッセージには、レポートを表示するためのリンクも記載されています。

   次の例は、レポート作成成功メッセージを含むバナーを示しています。

   レポートは [パフォーマンス分析レポート - 新規] タブで表示できます。

パフォーマンス分析レポートは、AWS CLI を使用して作成できます。AWS CLI を使用してレポートを作成する方法の例については、「一定期間のパフォーマンス分析レポートの作成」を参照してください。

パフォーマンス分析レポートの表示

[パフォーマンス分析レポート - 新規] タブには、DB インスタンスについて作成されたすべてのレポートが表示されます。各レポートには、以下が表示されます。

- [ID]: レポートの一意識別子。
- [名前]: レポートに追加されたタグキー。
- [レポート作成時間]: レポートを作成した時刻。
- [分析開始時間]: レポート内の分析の開始時刻。
- [分析終了時間]: レポート内の分析の終了時刻。

パフォーマンス分析レポートを表示するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. 分析レポートを表示する DB インスタンスを選択します。

   この DB インスタンスに Performance Insights ダッシュボードが表示されます。
4. 下にスクロールして、パフォーマンス分析レポート - 新規] タブを選択します。

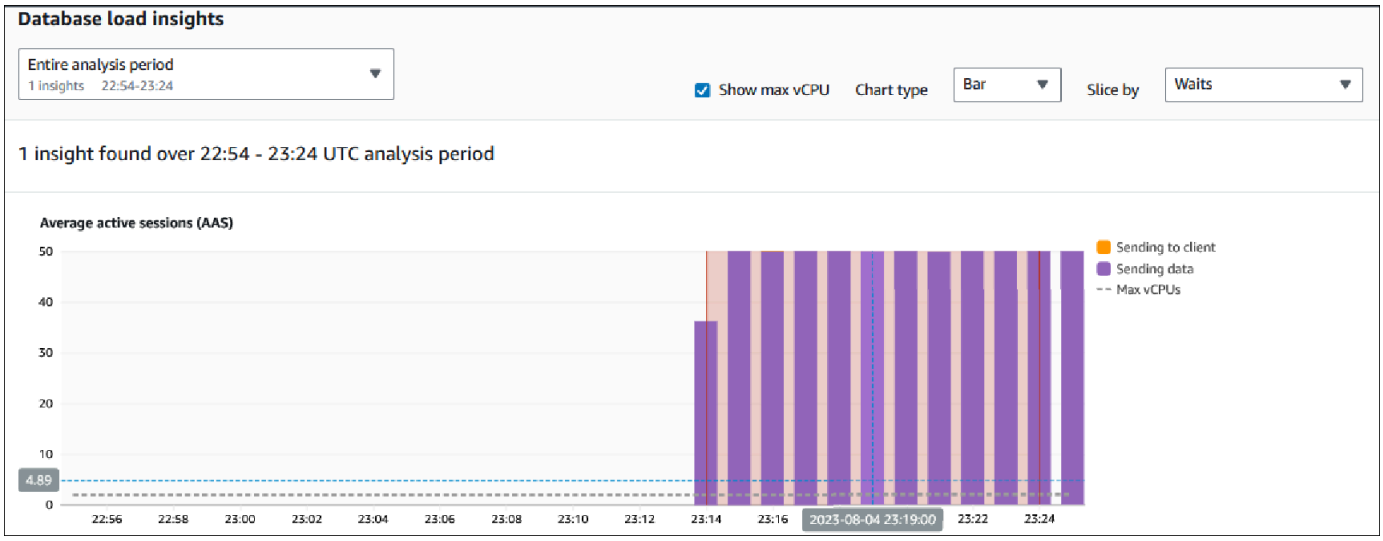
   異なる期間のすべての分析レポートが表示されます。

5. 表示するレポートの [ID] を選択します。

   複数のインサイトが特定された場合、DB 負荷チャートにはデフォルトで分析期間全体が表示されます。レポートで 1 つのインサイトが特定された場合、DB 負荷チャートにはデフォルトでそのインサイトが表示されます。

   ダッシュボードには、[タグ] セクションにレポートのタグも一覧表示されます。

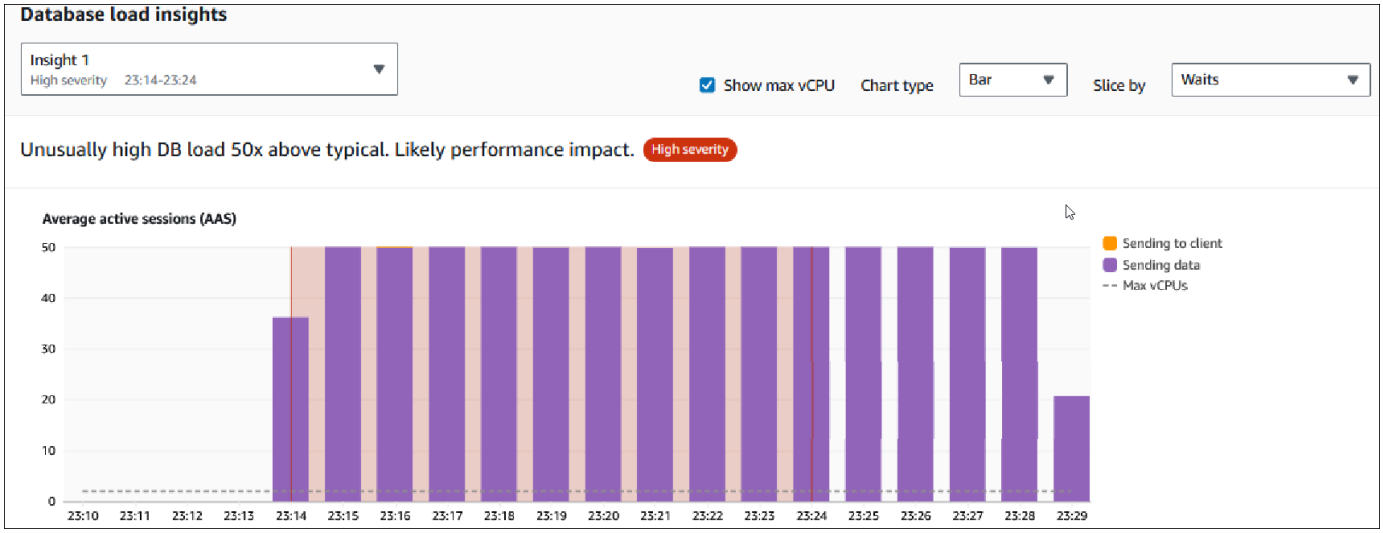
   次の例は、レポートの分析期間全体を示しています。

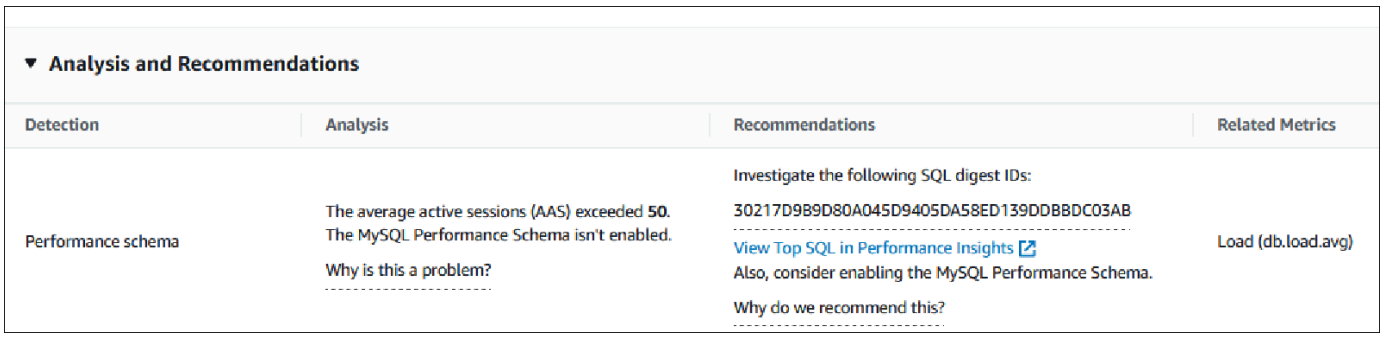


6. レポートで複数のインサイトが特定された場合、[データベース負荷に関するインサイト] リストで、表示するインサイトを選択します。

   ダッシュボードには、インサイトメッセージ、インサイトの期間が強調表示された DB 負荷チャート、分析と推奨事項、およびレポートタグのリストが表示されます。

   次の例は、レポートの DB 負荷インサイトを示しています。





パフォーマンス分析レポートにタグを追加する

レポートを作成または表示するときに、タグを追加できます。レポートには最大 50 個のタグを追加できます。

タグを追加するアクセス許可が必要です。Performance Insights のアクセスポリシーの詳細については、「Performance Insights 用のアクセスポリシーの設定」を参照してください。

レポートの作成時に 1 つ以上のタグを追加するには、手順 パフォーマンス分析レポートの作成 のステップ 6 を参照してください。

レポートを表示するときに 1 つ以上のタグを追加するには

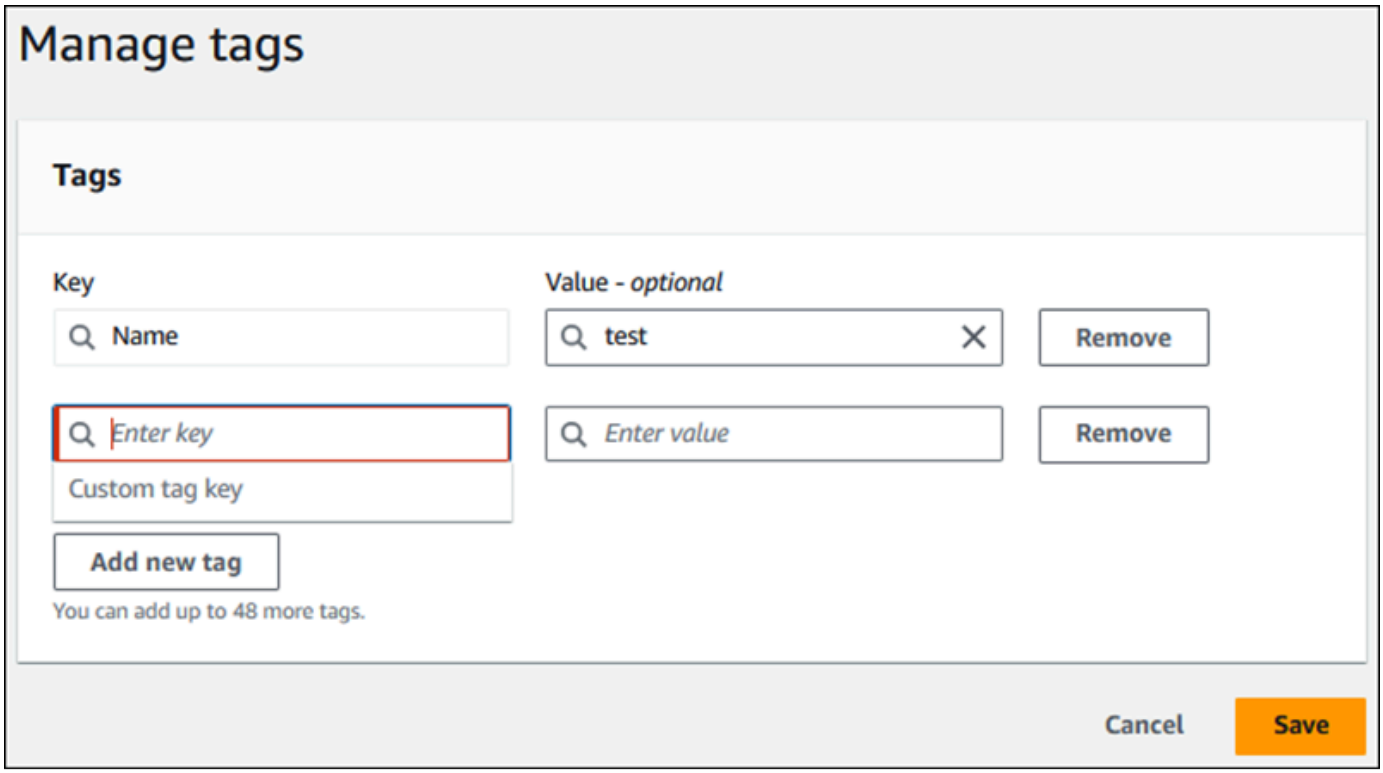
1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。

   この DB インスタンスに Performance Insights ダッシュボードが表示されます。

4. 下にスクロールして、パフォーマンス分析レポート - 新規] タブを選択します。
5. タグを追加するレポートを選択します。

   ダッシュボードにレポートが表示されます。
6. [タグ] までスクロールして、[タグを管理] を選択します。
7. [新しいタグを追加] をクリックします。
8. [キー] と [値 - オプション] を入力し、[新しいタグを追加] を選択します。

   次の例では、選択したレポートに新しいタグを追加するオプションを提供しています。



   レポート用に新しいタグが作成されます。

   ダッシュボードの [タグ] セクションにレポートのタグのリストが表示されます。レポートからタグを削除する場合は、タグの横の [削除] を選択します。

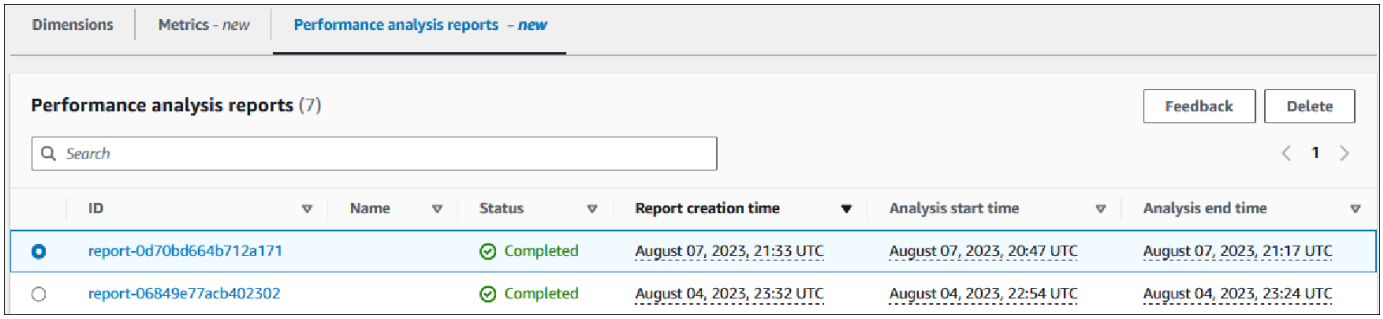
パフォーマンス分析レポートの削除

[パフォーマンス分析レポート] タブに表示されているレポートのリストから、またはレポートの表示中にレポートを削除できます。

レポートを削除するには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。

   この DB インスタンスに Performance Insights ダッシュボードが表示されます。
4. 下にスクロールして、パフォーマンス分析レポート - 新規] タブを選択します。
5. 削除するレポートを選択し、右上の [削除] を選択します。



   確認ウィンドウが表示されます。確認を選択すると、レポートは削除されます。
6. (オプション) 削除するレポートの ID を選択します。

   レポートページの右上にある [削除] を選択します。

   確認ウィンドウが表示されます。確認を選択すると、レポートは削除されます。

Performance Insights ダッシュボードのクエリの分析

Amazon RDS Performance Insights ダッシュボードでは、実行中のクエリや最近のクエリに関する情報を [Top dimensions] (上位ディメンション) テーブルの[Top SQL] (上位の SQL) タブで見ることができます。この情報を使用して、クエリをチューニングできます。

トピック

- [トップ SQL] タブの概要
- Performance Insights ダッシュボードでより多くの SQL テキストにアクセスする
- Performance Insights ダッシュボードでの SQL 統計の表示

[トップ SQL] タブの概要

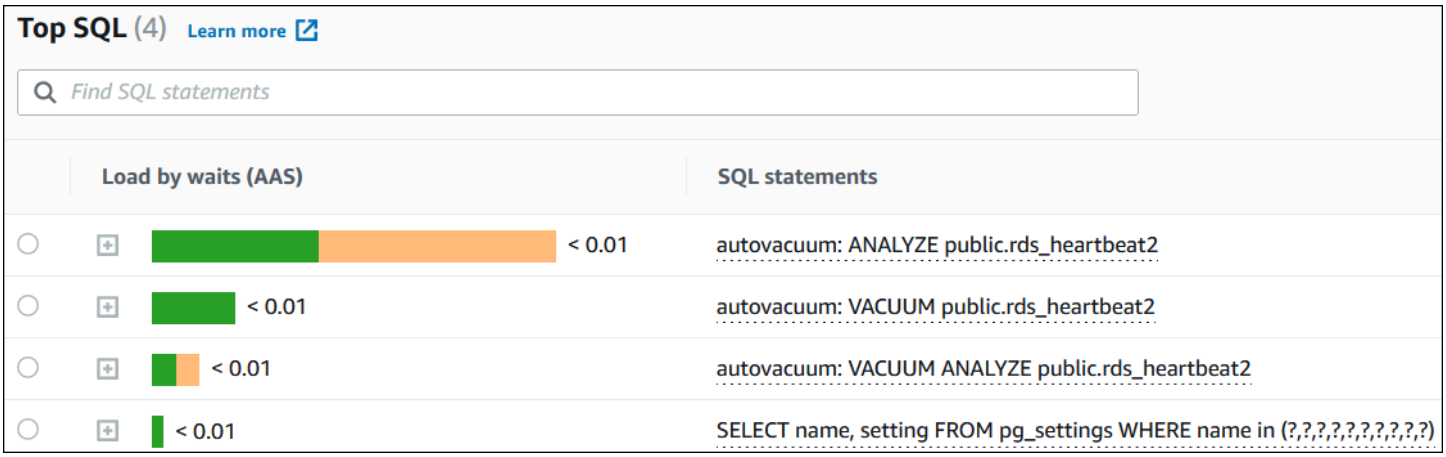
デフォルトでは、[Top SQL] (上位の SQL) タブはデータベースロードに最も貢献している 25 クエリを表示します。クエリをチューニングするために、クエリテキスト、SQL 統計などの情報を分析できます。また、[トップ SQL] タブに表示する統計を選択することもできます。

トピック

- SQL テキスト
- SQL 統計
- 待機によるロード (AAS)
- SQL 情報
- 設定

SQL テキスト

デフォルトでは、[Top SQL] (上位の SQL) テーブルの各行にはステートメントごとに 500 バイトのテキストが表示されます。



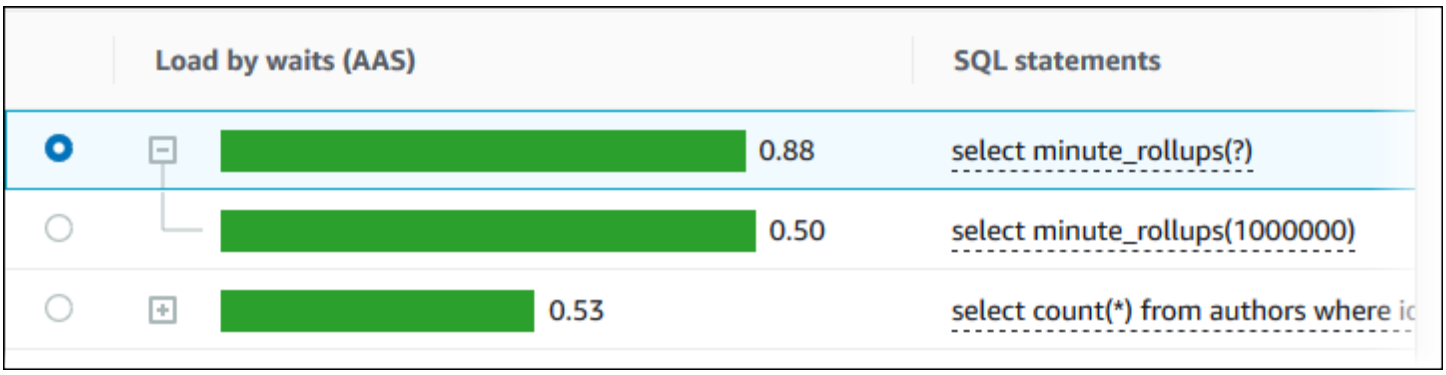
デフォルトの 500 バイト以上の SQL テキストを表示する方法については、「Performance Insights ダッシュボードでより多くの SQL テキストにアクセスする」を参照してください。

SQL ダイジェストは、構造的には類似しているが、異なるリテラル値を含む可能性の高い、複数の実際のクエリの複合体です。ダイジェストは、ハードコードされた値を疑問符に置き換えます。例えば、ダイジェストは`SELECT * FROM emp WHERE lname= ?`のことがあります。このダイジェストには、次の子クエリが含まれます。

```
SELECT * FROM emp WHERE lname = 'Sanchez'
```

```
SELECT * FROM emp WHERE lname = 'Olagappan'
SELECT * FROM emp WHERE lname = 'Wu'
```

ダイジェスト内でリテラル SQL ステートメントを表示するには、クエリを選択してからプラス記号 (+) を選択します。以下の例では、選択されたクエリはダイジェストです。

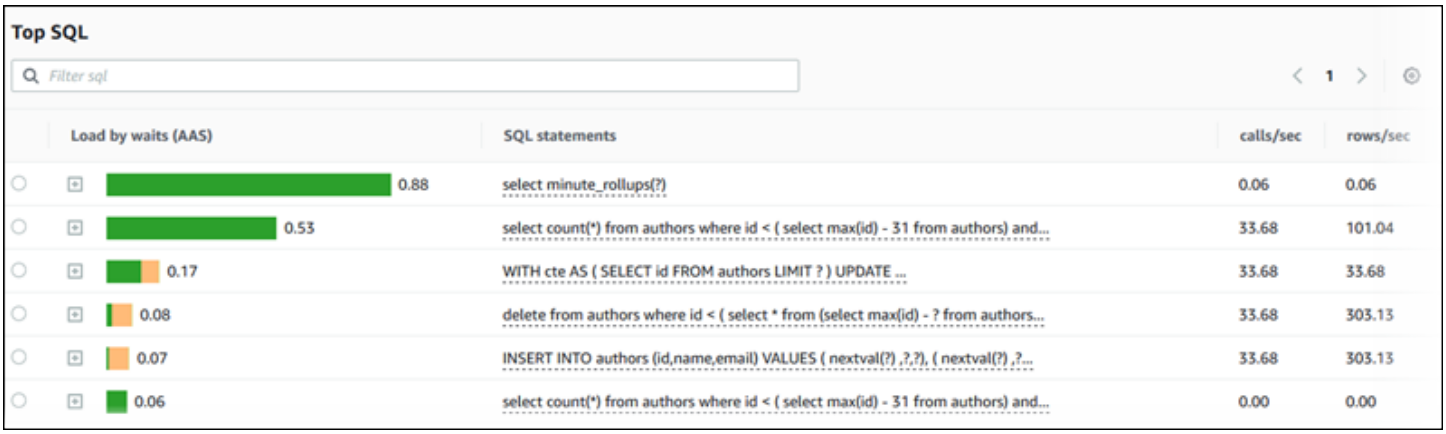


> **Note**
>
> SQL ダイジェストでは、類似した SQL ステートメントがグループ化されますが、機密情報は編集されません。

SQL 統計

SQL 統計は、SQL クエリに関するパフォーマンス関連のメトリックです。例えば、Performance Insights には 1 秒あたりの実行数や 1 秒あたりの処理行数が表示されることがあります。Performance Insights は、最も一般的なクエリのみの統計を収集します。通常、これらは Performance Insights ダッシュボードに負荷別に表示される上位のクエリと一致します。

[トップ SQL] テーブル内の各行は、次の例のように、SQL ステートメントまたはダイジェストに関連する統計を示します。



Performance Insights は、SQL 統計で `0.00` および - (不明) をレポートする可能性があります。この状況は、以下の条件で発生します。

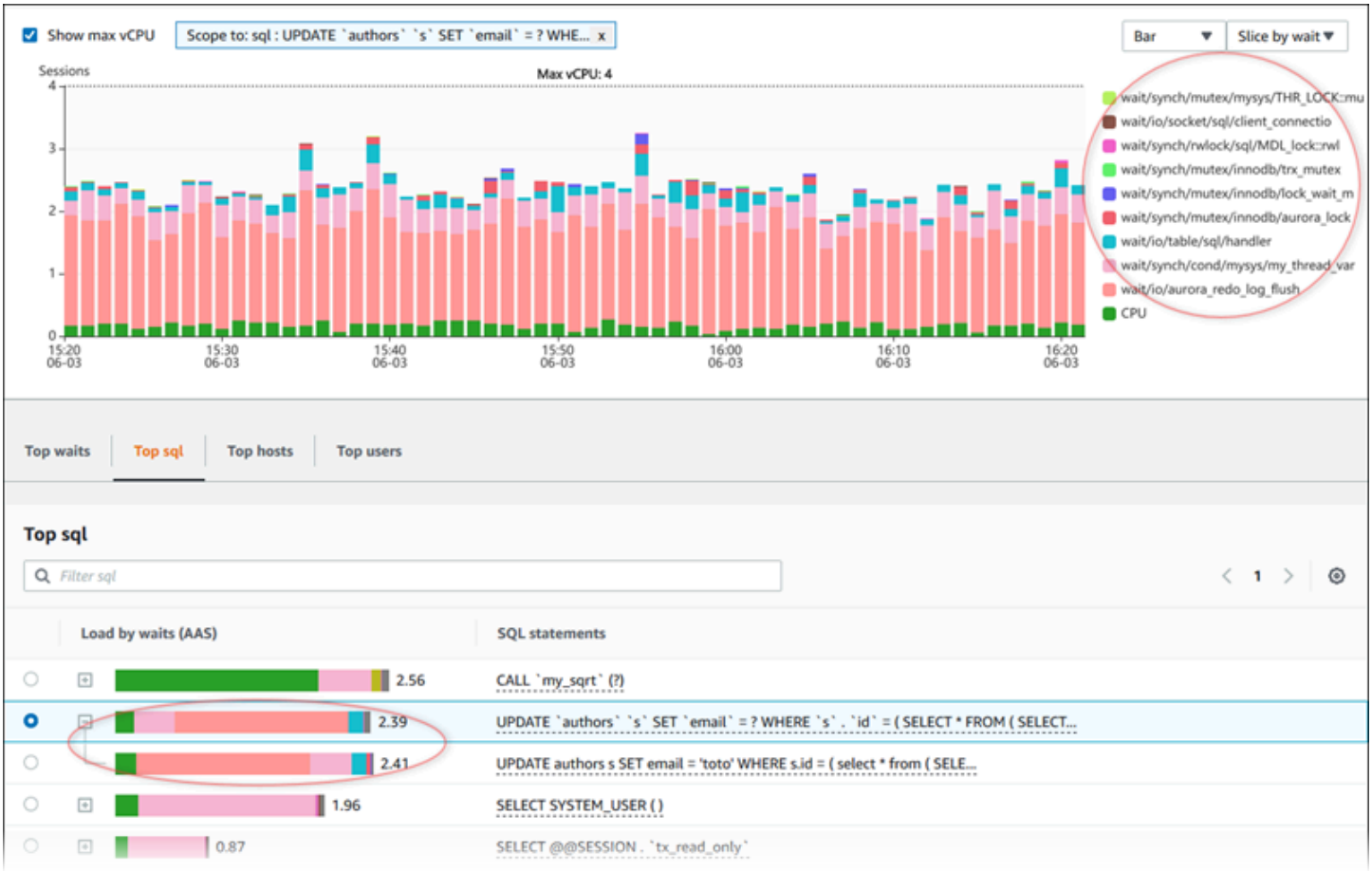
- サンプルが 1 つだけ存在する。例えば、Performance Insights は、`pg_stat_statements` ビューからの複数のサンプルに基づいて、Aurora PostgreSQL クエリの変更率を計算します。ワークロードが短時間実行されると、Performance Insights ではサンプルが 1 つしか収集されない場合があります。つまり、変更率を計算できません。不明な値はダッシュ (-) で表されます。
- 2 つのサンプルが同じ値を持っている。Performance Insights は、変更が発生していないため、変更率を計算することができず、変化率を `0.00` と報告します。
- Aurora PostgreSQL ステートメントに有効な識別子がない。PostgreSQL は、構文解析と分析の後にのみ、ステートメントの識別子を作成します。したがって、PostgreSQL の内部インメモリ構造に、識別子なしでステートメントが存在する可能性があります。Performance Insights は内部メモリ内構造を 1 秒に 1 回サンプリングするため、低レイテンシーのクエリは 1 つのサンプルに対してのみ表示されることがあります。このサンプルでクエリ識別子を使用できない場合、Performance Insights はこのステートメントを統計に関連付けることはできません。不明な値はダッシュ (-) で表されます。

Aurora エンジンの SQL 統計の説明については、「Performance Insights の SQL 統計」を参照してください。

待機によるロード (AAS)

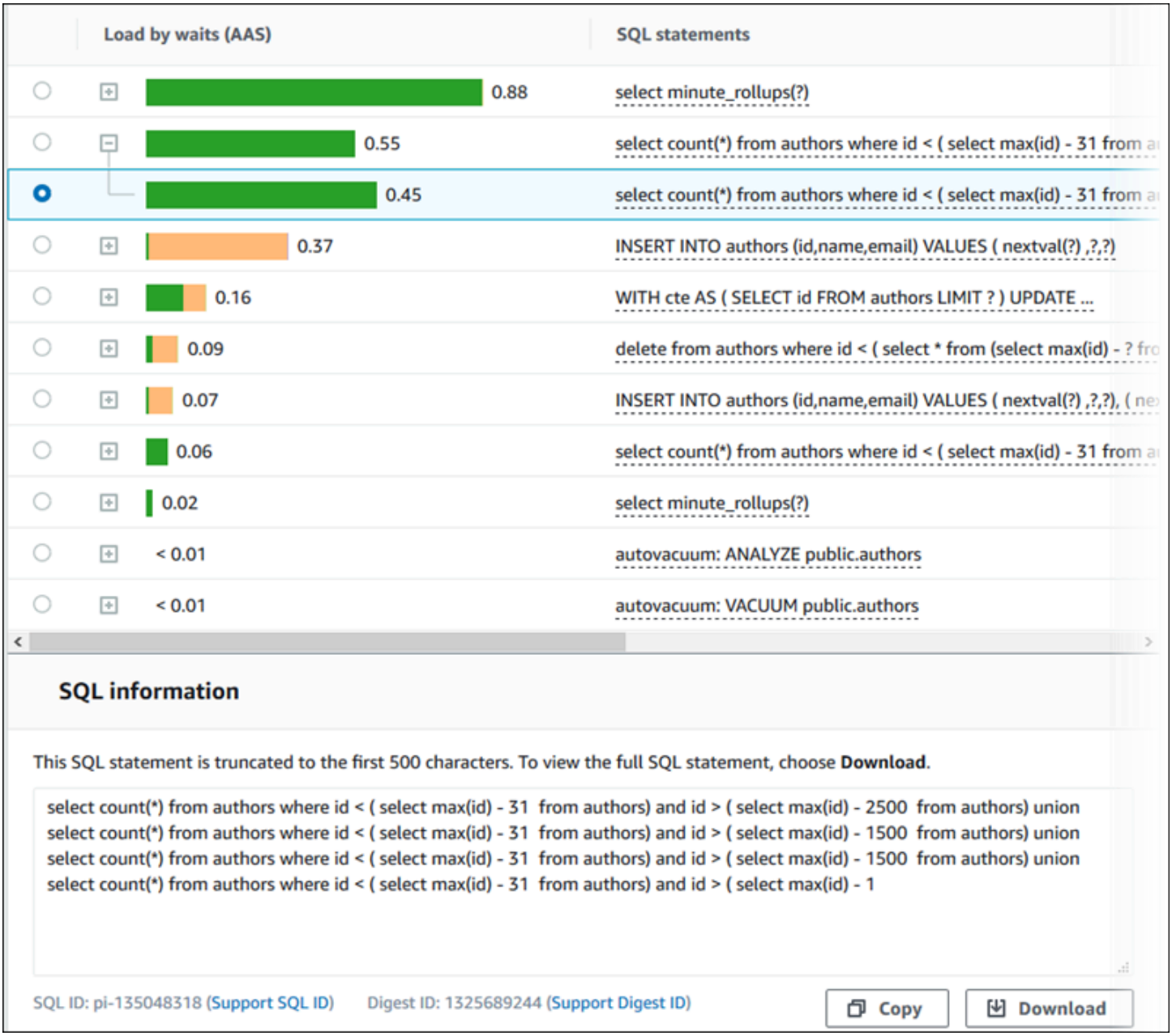
[トップ SQL] の [待機別の負荷 (AAS)] 列は、上位の各ロード項目に関連付けられているデータベースロードの割合を示しています。この列には、DB 負荷グラフで現在選択されているグループ化に応じて、その項目に対する負荷が反映されます。平均アクティブセッション (AAS) の詳細については、平均アクティブセッション を参照してください。

例えば、DB 負荷グラフを待機状態別にグループ化できます。上位負荷項目のテーブルで SQL クエリを調べます。この場合は、[待機別の DB 負荷] バーは、クエリが貢献している待機状態の量を示すために、サイズ、セグメント、および色で分けられています。また、選択したクエリに影響を与えている待機状態も示されます。



SQL 情報

[トップ SQL] テーブルで、ステートメントを開いてその情報を表示できます。下部のペインに情報が表示されます。

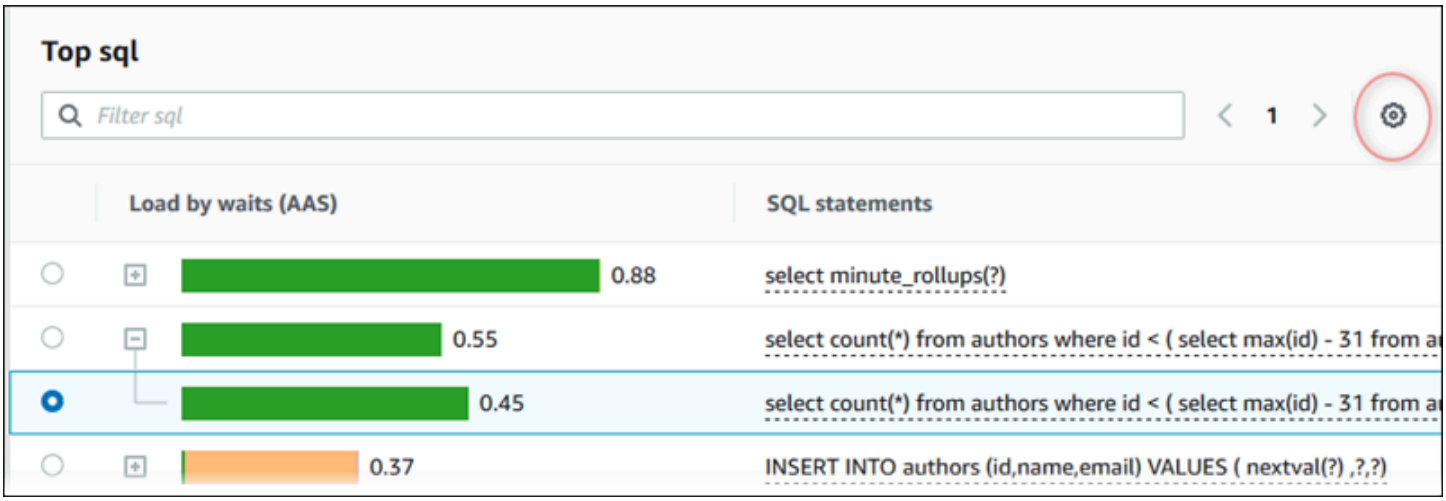


SQL ステートメントに関連付けられているタイプの識別子 (ID) は以下のとおりです。

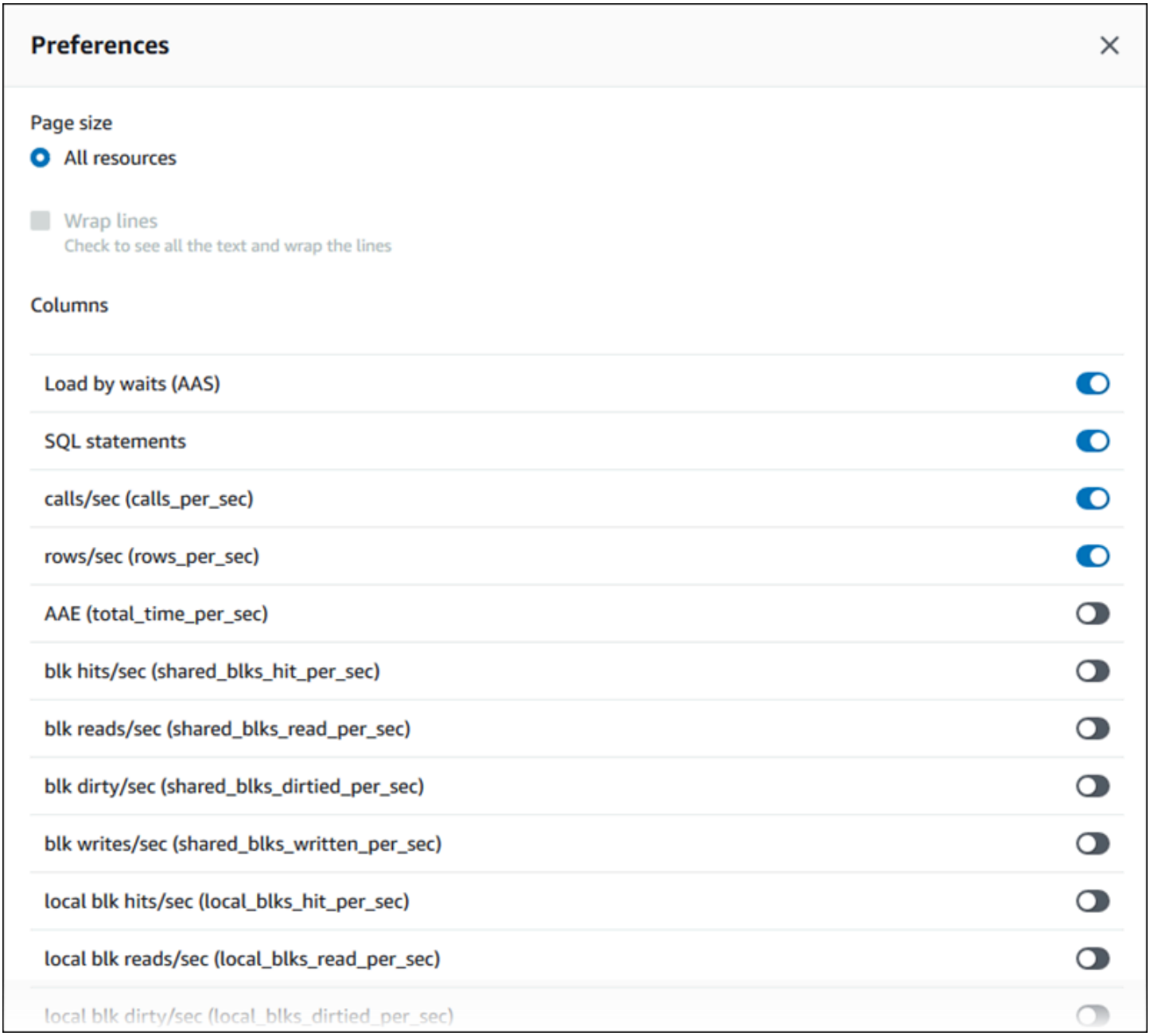
- Support SQL ID - SQL ID のハッシュ値。この値は、AWS サポートを利用しているときに SQL ID を参照するためだけのものです。AWS サポートが実際の SQL ID や SQL テキストにアクセスすることはできません。
- ダイジェスト ID のサポート - Digest ID のハッシュ値。この値は、AWS サポートを利用しているときにダイジェスト ID を参照するためだけのものです。AWSサポートが実際のダイジェスト ID や SQL テキストにアクセスすることはできません。

設定

「設定」 アイコンを選択すると、[トップ SQL] タブに表示される統計を制御できます。



[設定] アイコンを選択すると、[設定] ウィンドウが開きます。次のスクリーンショットは、[Preferences] (環境設定) ウィンドウの例です。



[トップ SQL] タブに表示させたい統計を有効にするには、マウスを使用してウィンドウの下部までスクロールし、[続行] を選択します。

Aurora エンジンの秒単位または呼び出し単位の統計の詳細については、Performance Insights の SQL 統計 の「エンジン固有の SQL 統計」セクションを参照してください。

Performance Insights ダッシュボードでより多くの SQL テキストにアクセスする

デフォルトでは、[トップ SQL] テーブルの各行には SQL ステートメントごとに 500 バイトの SQL テキストが表示されます。

SQL ステートメントのサイズが 500 バイトを超える場合、[Top SQL] (トップ SQL) テーブルの [SQL text] (SQL テキスト) セクションでテキストの表示量を増やすことができます。この場合、[SQL text] (SQL テキスト) に表示されるテキストの最大長は 4 KB です。コンソールで導入されるこの制限には、データベースエンジンによって設定された制限が適用されます。[SQL text] (SQL テキスト) に表示されているテキストを保存するには、[Download] (ダウンロード) を選択します。

トピック

- Aurora MySQL のテキストサイズの制限
- Aurora PostgreSQL DB インスタンスの SQL テキスト制限の設定
- Performance Insights ダッシュボードでの SQL テキストの表示とダウンロード

Aurora MySQL のテキストサイズの制限

SQL テキストをダウンロードするときに、データベースエンジンがテキストの最大長を決定します。エンジンごとのダウンロードできる SQL テキストの上限は次のとおりです。

DB エンジン	ダウンロードされるテキストの最大長
Aurora MySQL	4,096 バイト

Performance Insights コンソールの [SQL text] (SQL テキスト) では、エンジンが返すテキストが最大値まで表示できます。例えば、Aurora MySQL は、Performance Insights に対して最大 1 KB を返します。元のクエリが大きい場合でも、収集して表示できるのは 1 KB のみです。したがって、[SQL text] (SQL テキスト) でクエリを表示するか、ダウンロードすると、Performance Insights は同じバイト数を返します。

AWS CLI または API を使用する場合、Performance Insights には、コンソールで適用される 4 KB の制限がありません。`DescribeDimensionKeys` と `GetResourceMetrics` は、最大で 500 バイトを返します。

> **Note**
>
> `GetDimensionKeyDetails` はクエリ全体を返しますが、サイズにはエンジンの制限が適用されます。

Aurora PostgreSQL DB インスタンスの SQL テキスト制限の設定

Aurora PostgreSQL は、テキストを異なる方法で処理します。DB インスタンスパラメータ `track_activity_query_size` を使用して、テキストサイズの制限を設定できます。このパラメータには次の特徴があります。

デフォルトのテキストサイズ

Aurora PostgreSQL バージョン 9.6 では、`track_activity_query_size` パラメータのデフォルト設定は 1,024 バイトです。Aurora PostgreSQL バージョン 10 以降では、デフォルトは 4,096 バイトです。

最大テキストサイズ

Aurora PostgreSQL バージョン 12 以前の場合、`track_activity_query_size` の制限は 102,400 バイトです。バージョン 13 以降の場合、最大値は 1 MB です。

エンジンが Performance Insights に対して 1 MB を返す場合、コンソールでは最初の 4 KB のみが表示されます。クエリをダウンロードする場合、1 MB すべてを取得できます。この場合、表示する場合とダウンロードする場合では異なるバイト数が返されます。`track_activity_query_size` DB インスタンスパラメータの詳細については、PostgreSQL ドキュメントで「ランタイム統計」を参照してください。

SQL テキストのサイズを大きくするには、`track_activity_query_size` の制限を引き上げます。パラメータを変更するには、Aurora PostgreSQL DB インスタンスに関連付けられているパラメータグループのパラメータ設定を変更します。

インスタンスでデフォルトのパラメータグループが使用される際に設定を変更するには

1. 該当する DB エンジンおよび DB エンジンバージョンの新しい DB インスタンスパラメータグループを作成します。
2. 新しいパラメータグループにパラメータを設定します。
3. 新しいパラメータグループを DB インスタンスに関連付けます。

DB インスタンスパラメータの設定の詳細については、「DB パラメータグループのパラメータの変更」を参照してください。

Performance Insights ダッシュボードでの SQL テキストの表示とダウンロード

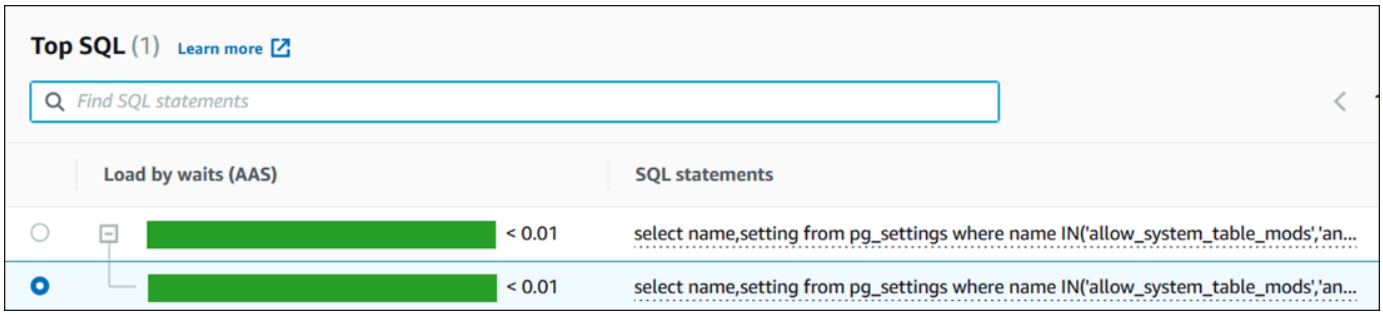
Performance Insights ダッシュボードで、SQL テキストを表示およびダウンロードできます。

Performance Insights ダッシュボードで SQL テキストの表示量を増やすには

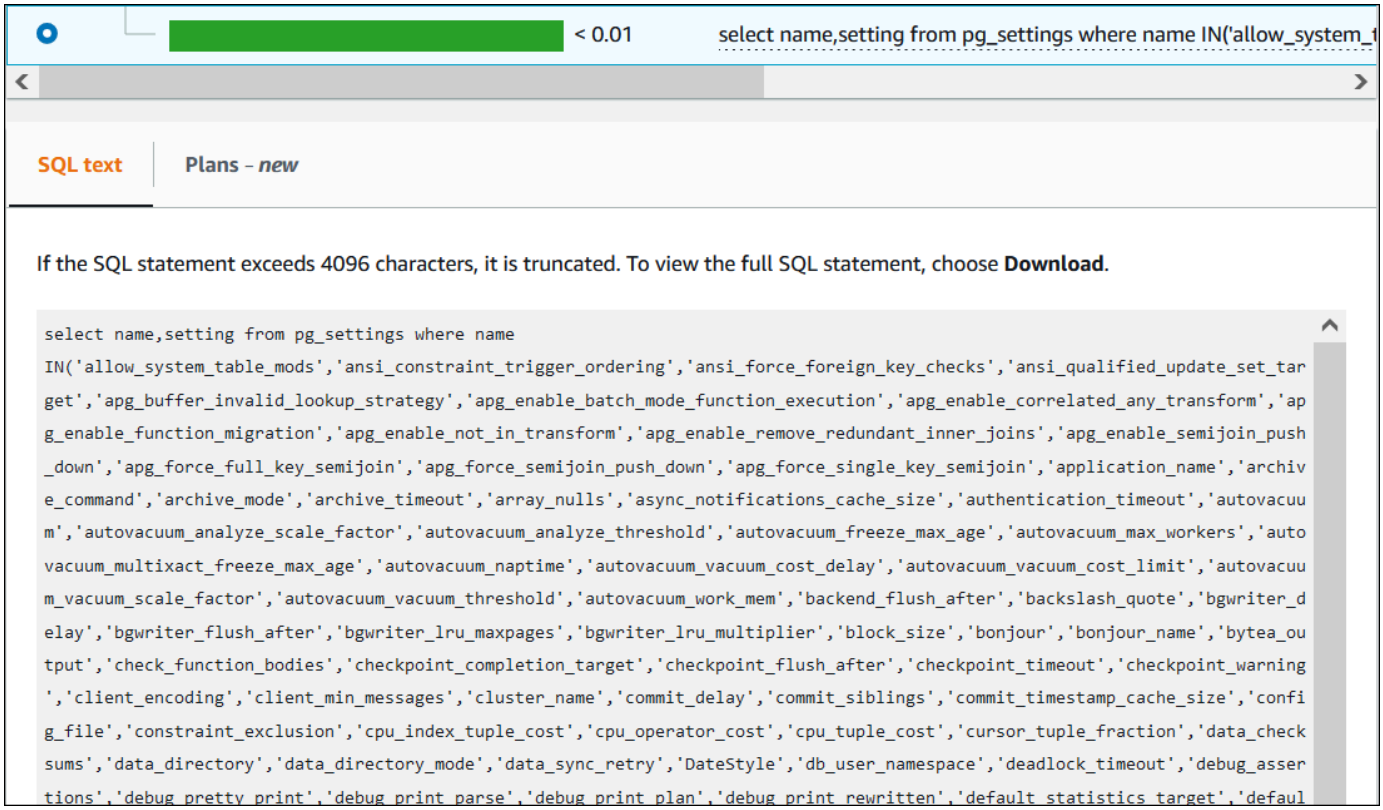
1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。

   DB インスタンスで Performance Insights ダッシュボードが表示されます。
4. 下部にある [トップ SQL] タブまでスクロールします。
5. プラス記号を選択して SQL ダイジェストを展開し、ダイジェストの子クエリのいずれかを選択します。

   500 バイトを超える SQL ステートメントは、次のイメージのように表示されます。



6. 下部にある [SQL テキスト] タブまでスクロールします。



Performance Insights ダッシュボードは、各 SQL ステートメントの最大 4,096 バイトまでを表示できます。

7. (オプション) [コピー] を選択して、表示された SQL ステートメントをコピーするか、[ダウンロード] を選択して、DB エンジンに応じた最大サイズの SQL ステートメントをダウンロードします。

> **Note**
>
> SQL ステートメントをコピーまたはダウンロードするには、ポップアップブロッカーを無効にします。

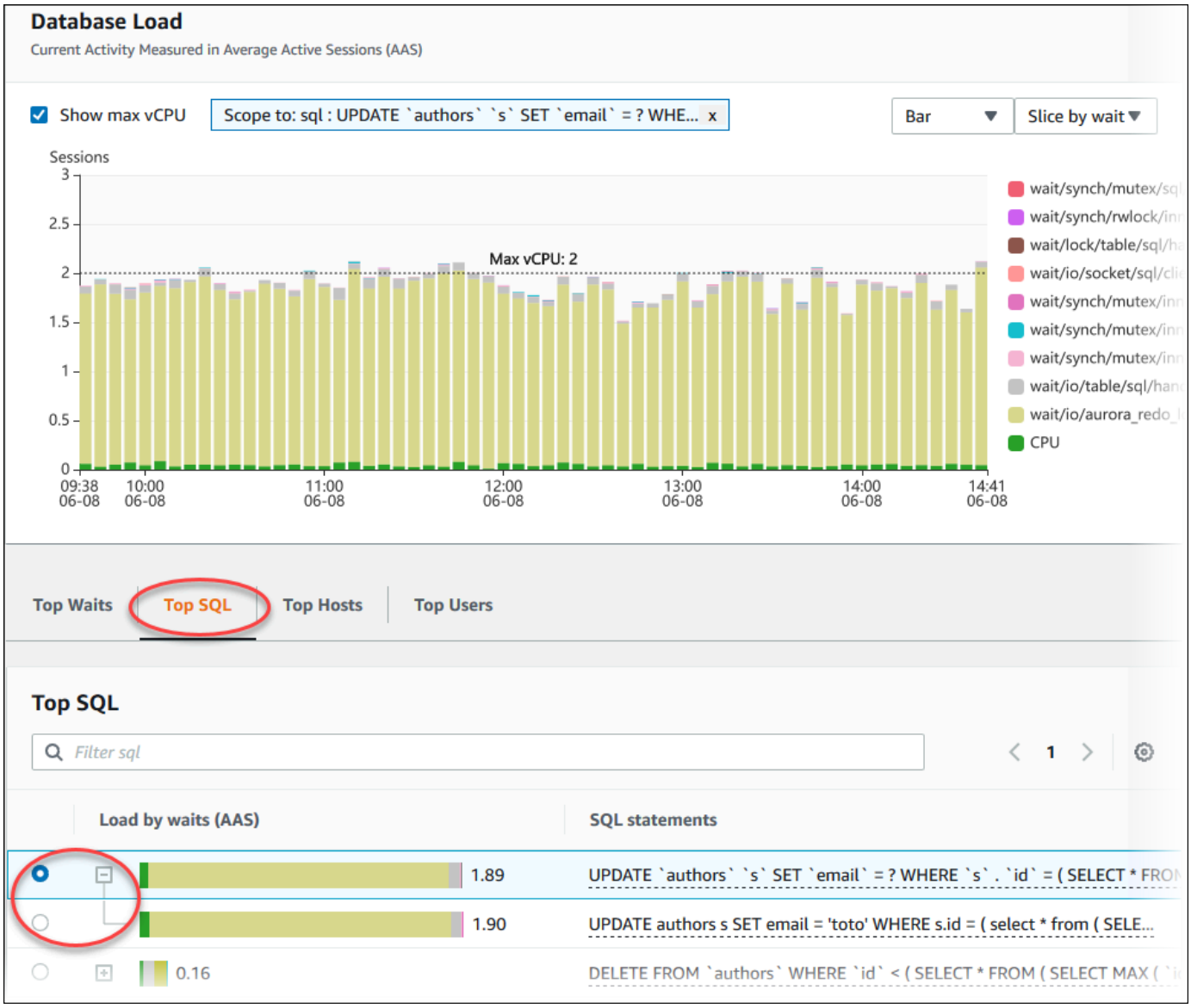
Performance Insights ダッシュボードでの SQL 統計の表示

Performance Insights ダッシュボードでは、SQL 統計を [データベース負荷] グラフの [トップ SQL] タブで見ることができます。

SQL 統計を表示するには

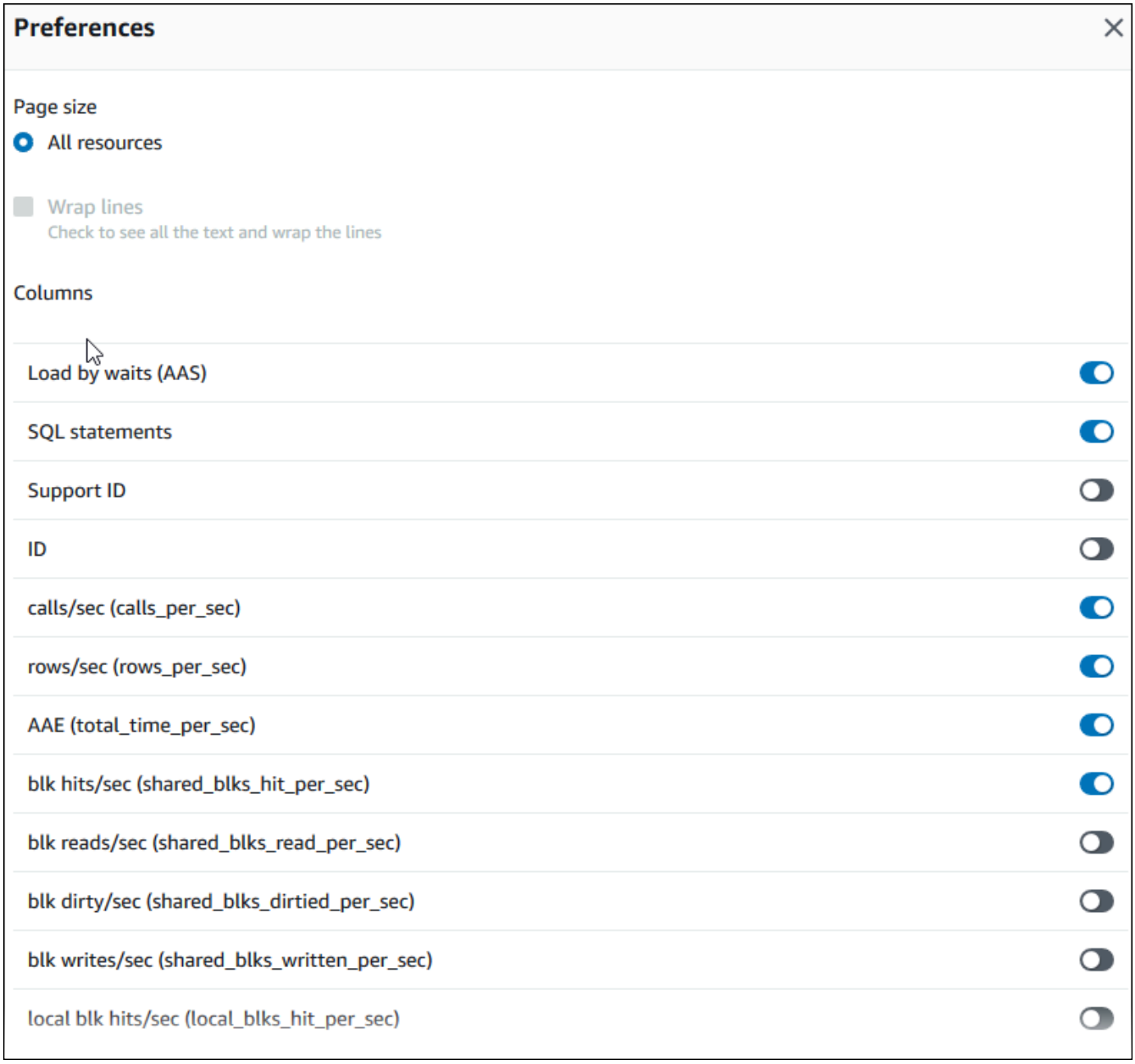
1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。

2. ナビゲーションペインで、[Performance Insights] を選択します。
3. ページの上部で、SQL 統計を表示するデータベースを選択します。
4. ページの下部までスクロールし、[トップ SQL] タブを選択します。
5. 個々のステートメント (Aurora MySQL のみ) またはダイジェストクエリを選択します。

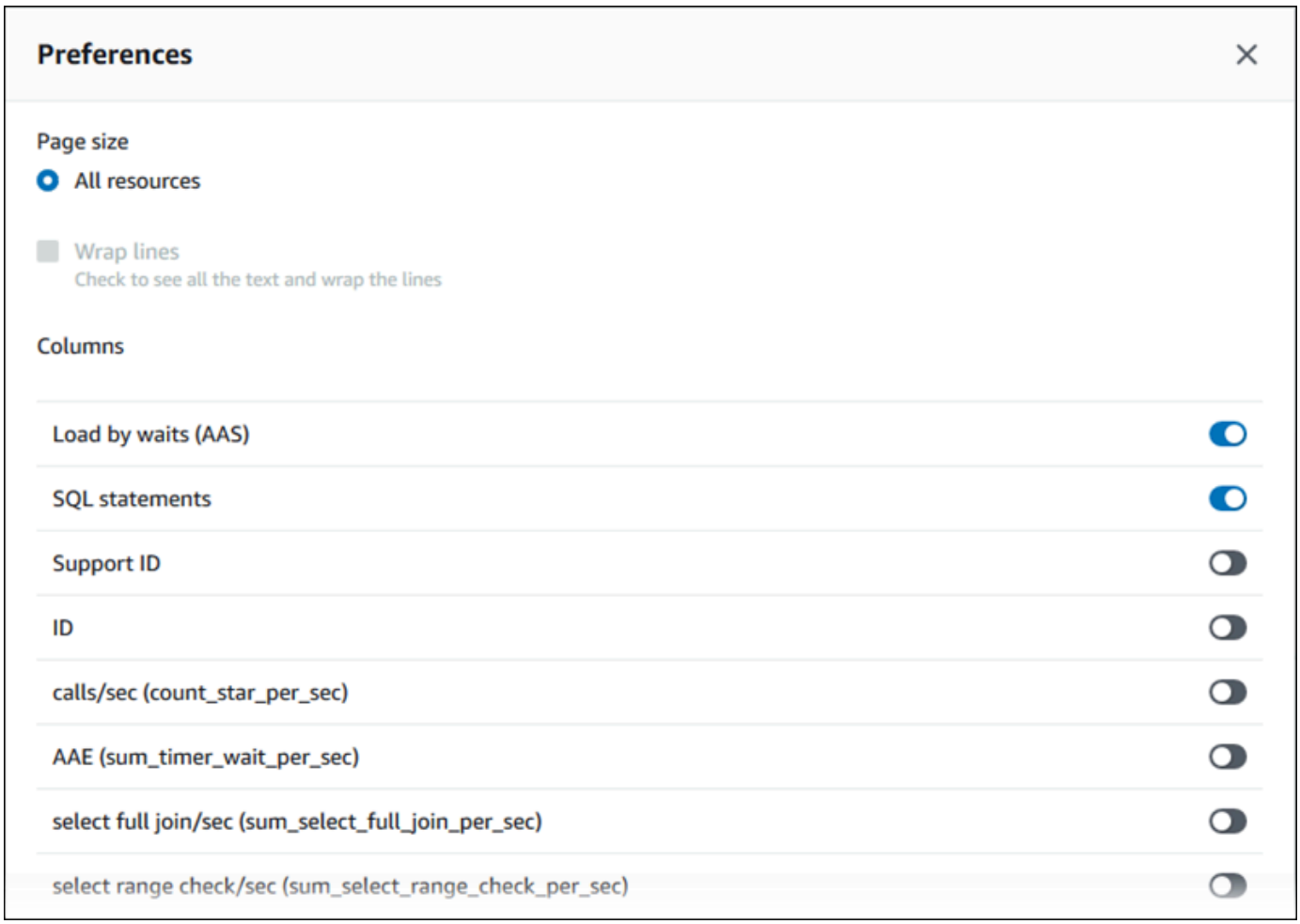


6. グラフの右上にある歯車のアイコンを選択して、表示する統計を選択します。Amazon RDSAurora エンジンの SQL 統計の説明については、「Performance Insights の SQL 統計」を参照してください。

   次の例は、Aurora PostgreSQL の設定を示しています。



次の例は、Aurora MySQL DB インスタンスの設定を示しています。



7. 設定を保存するには [保存] を選択します。

   [トップ SQL] テーブルが更新されます。

Performance Insights の事前対応型推奨事項の表示

Amazon RDS Performance Insights は特定のメトリクスを監視し、特定のリソースで潜在的に問題と見なされる可能性があるレベルを分析することで自動的にしきい値を作成します。新しいメトリクス値が事前定義されたしきい値を一定期間にわたって超えた場合に、Performance Insights は事前対応型推奨事項を生成します。この推奨事項は、将来のデータベースパフォーマンスへの影響を防ぐのに役立ちます。これらの事前対応型推奨事項を受け取るには、有料利用枠の保持期間を使用して Performance Insights を有効にする必要があります。

Performance Insights をオンにする方法の詳細については、「Performance Insights の有効化と無効化」を参照してください。Performance Insights の料金とデータ保持については、「Performance Insights の料金とデータ保持」を参照してください。

事前対応型推奨事項でサポートされているリージョン、DB エンジン、インスタンスクラスについては、「Amazon Aurora DB エンジン、リージョン、およびインスタンスクラスでサポートされている Performance Insights 機能」を参照してください。

事前対応型推奨事項の詳細な分析と推奨調査は、推奨事項の詳細ページで確認できます。

レコメンダーの詳細については、「Amazon Aurora の推奨事項の表示とこれらに対する対応」を参照してください。

事前対応型推奨事項の詳細な分析を表示するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、次のいずれかを実行します。
   - [レコメンデーション] を選択します。

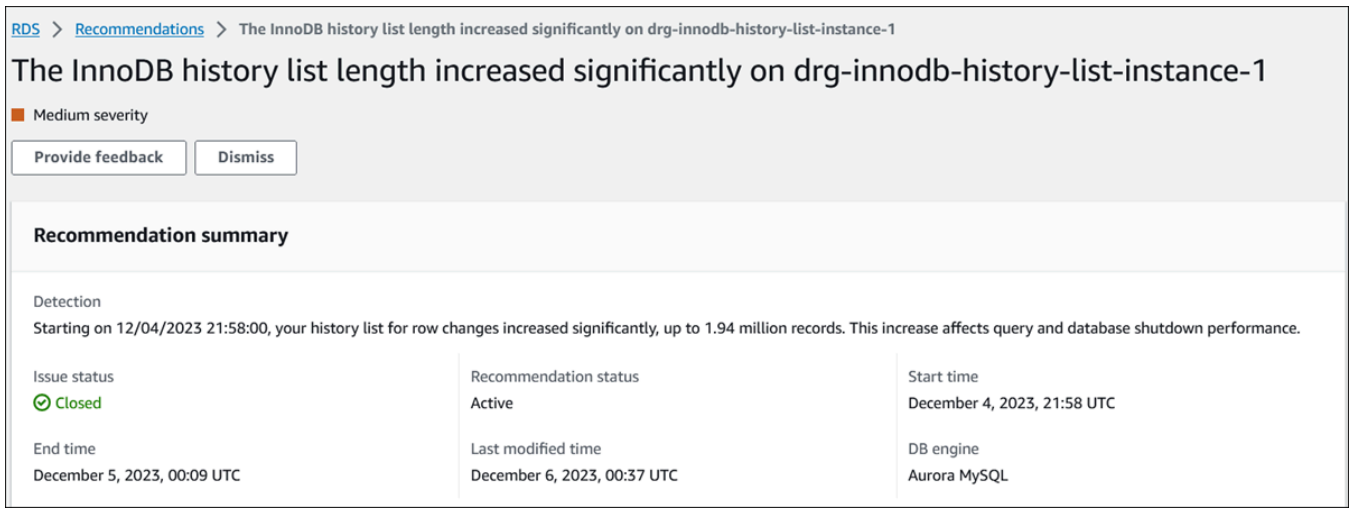
     [レコメンデーション] ページには、アカウント内のすべてのリソースの重要度でソートされた推奨事項のリストが表示されます。
   - [データベース] を選択し、データベースページでリソースの [レコメンデーション] を選択します。

     [レコメンデーション] タブには、選択したリソースの推奨事項とその詳細が表示されます。
3. 事前対応型推奨事項を検索し、[詳細の表示] を選択します。

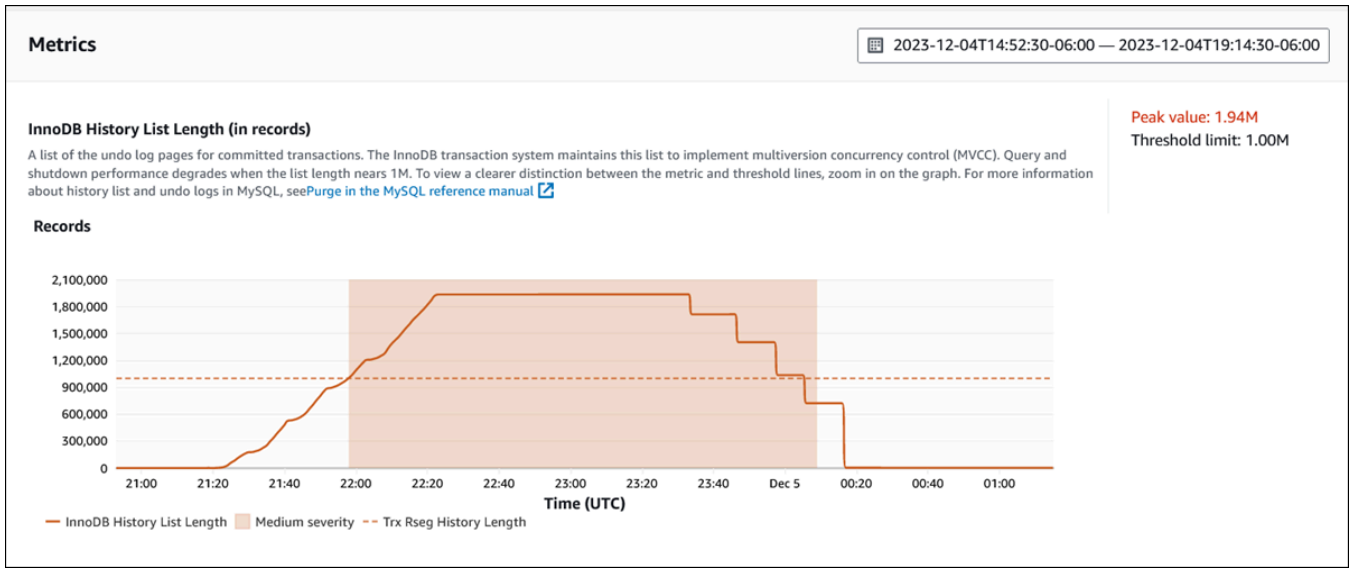
   推奨事項の詳細ページが表示されます。タイトルには、問題が検出されたリソースの名前と影響されるリソースが表示されます。

   推奨事項の詳細ページのコンポーネントは次のとおりです。

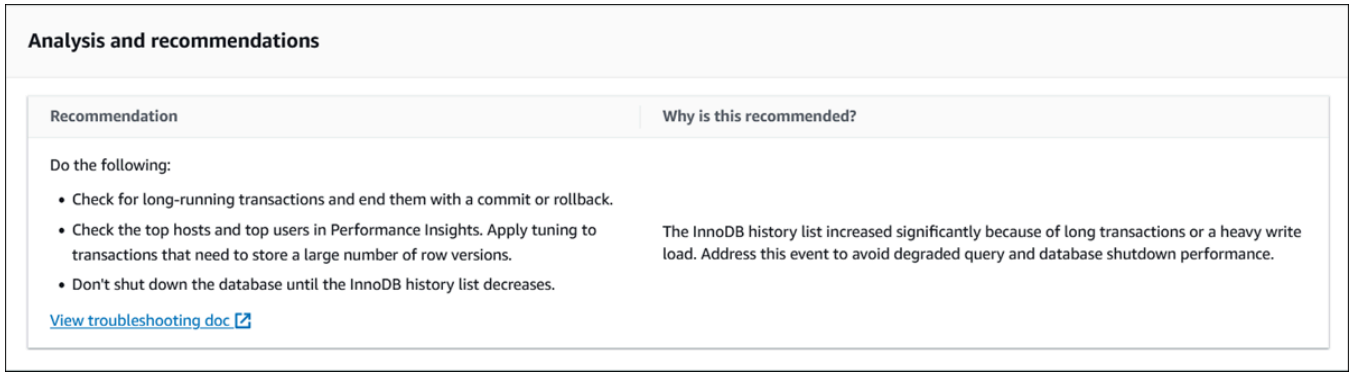
   - [レコメンデーションの概要] – 検出された問題、推奨事項と問題のステータス、問題の開始時刻と終了時刻、推奨事項の変更時刻、エンジンタイプ。



- [メトリクス] – 検出された問題のグラフ。各グラフには、リソースのベースライン動作によって決まるしきい値と、問題の開始時刻から報告されたメトリクスのデータが表示されます。



- [分析とレコメンデーション] — 推奨事項と推奨事項の理由。



問題の原因を確認し、推奨されるアクションを実行して問題を解決するか、右上の [閉じる] を選択して推奨事項を却下できます。

Performance Insights API によるメトリクスの取得

Performance Insights がオンになっている場合、API はインスタンスのパフォーマンスを可視化します。Amazon CloudWatch Logs は、AWS のサービスをモニタリングしたメトリクスの信頼性のある提供元です。

Performance Insightsは、平均アクティブ・セッション(AAS)として測定されるデータベースロードのドメイン固有のビューを提供します。このメトリクスはAPI利用者には2次元時系列データセットのように見えます。データの時間ディメンションは、クエリされた時間範囲内の各時点のDBロード・データを提供します。各時点で、その時点で計測された SQL、Wait-event、User、Host などのリクエストされたディメンションに関する負荷全体が分解されます。

Amazon RDS Performance Insights では、 Amazon Aurora DB クラスターをモニタリングし、データベースパフォーマンスの分析とトラブルシューティングを行うことができます。Performance Insights は、AWS Management Console で表示することができます。また、Performance Insights では独自のデータをクエリできるように、パブリック API も提供されています。API を使用して、次を実行できます。

- データベースにデータをオフロードする
- Performance Insights データを既存のモニタリングダッシュボードに追加する
- モニタリングツールを構築する

Performance Insights API を使用するには、いずれかの Amazon RDS DB インスタンスで Performance Insights を有効にします。Performance Insights の有効化については、「Performance Insights の有効化と無効化」を参照してください。Performance Insights API の詳細については、「Amazon RDS Performance Insights API リファレンス」を参照してください。

Performance Insights API は、以下のオペレーションを提供します。

Performance Insights でのアクション	AWS CLI コマンド	説明
CreatePerformanceAnalysisReport	aws pi create-performance-analysis-report	DB インスタンスの特定の期間のパフォーマンス分析レポートを作成します。結果は、レポートの固有識別子である AnalysisReportId です。
DeletePerformanceAnalysisReport	aws pi delete-performance-analysis-report	パフォーマンス分析レポートを削除します。
DescribeDimensionKeys	aws pi describe-dimension-keys	特定の期間に、メトリクスの上位 N 個のディメンションキーを取得します。
GetDimensionKeyDetails	aws pi get-dimension-key-details	DB インスタンスまたはデータソースの指定されたディメンショングループの属性を取得します。例えば、SQL ID を指定し、ディメンションの詳細が使用可能な場合、GetDimensionKeyDetails は、この ID に関連付けられているディメンション db.sql.statement の全文を取得します。このオペレーションは、GetResourceMetrics および DescribeDimensionKeys が大きな SQL ステートメントテキストの取得をサポートしないため、便利です。

Performance Insights でのアクション	AWS CLI コマンド	説明
GetPerformanceAnalysisReport	aws pi get-performance-analysis-report	レポートのインサイトを含むレポートを取得します。結果には、レポートのステータス、レポート ID、レポート時間の詳細、インサイト、および推奨事項が含まれます。
GetResourceMetadata	aws pi get-resource-metadata	さまざまな機能に関するメタデータを取得します。例えば、メタデータにより、特定の DB インスタンスで何等かの機能が有効化されているか無効化されているかを、示すことができます。
GetResourceMetrics	aws pi get-resource-metrics	期間中、データソースのセットに Performance Insights のメトリクスを取得します。特定のディメンショングループおよびディメンションを提供し、各グループの集約とフィルタリング条件を提供することができます。
ListAvailableResourceDimensions	aws pi list-available-resource-dimensions	指定したインスタンスで、指定したメトリクスタイプごとにクエリできるディメンションを取得します。
ListAvailableResourceMetrics	aws pi list-available-resource-metrics	DB インスタンスを指定しながら、指定されたメトリクスタイプでクエリが可能なメトリクスをすべて取得します。

Performance Insights でのアクション	AWS CLI コマンド	説明
ListPerformanceAnalysisReports	aws pi list-performance-analysis-reports	DB インスタンスについて入手可能なすべての分析レポートを取得します。レポートは、各レポートの開始時間に基づいて一覧表示されます。
ListTagsForResource	aws pi list-tags-for-resource	リソースに追加されたすべてのメタデータタグを一覧表示します。リストには、タグの名前と値が含まれます。
TagResource	aws pi tag-resource	Amazon RDS リソースにメタデータタグを追加します。タグには名前と値が含まれます。
UntagResource	aws pi untag-resource	メタデータタグをリソースから削除します。

トピック

- Performance Insights で AWS CLI を使用する
- 時系列メトリクスの取得
- Performance Insights での AWS CLI の例

Performance Insights で AWS CLI を使用する

Performance Insights は、AWS CLI を使用して表示することができます。Performance Insights の AWS CLI コマンドのヘルプを表示するには、コマンドラインで次のように入力します。

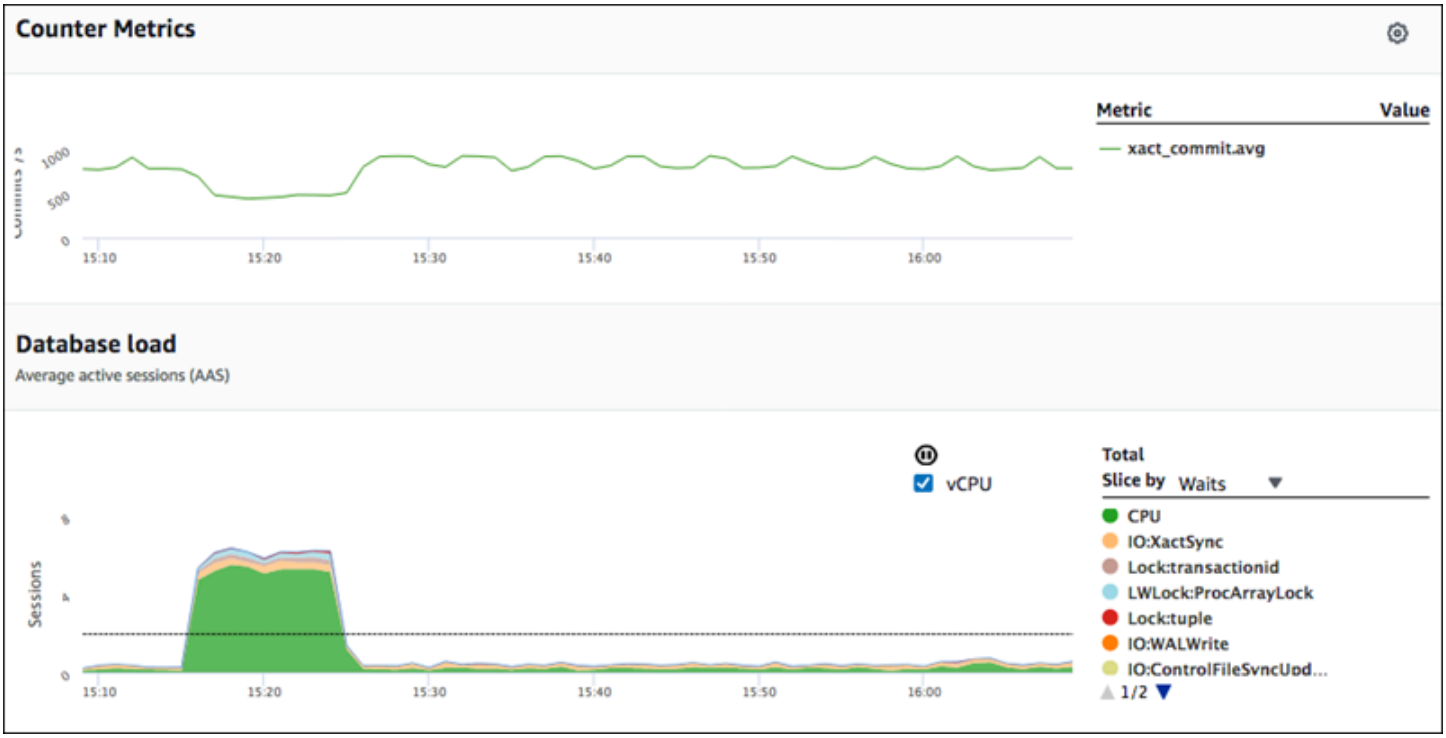
```
aws pi help
```

AWS CLI がインストールされていない場合は、AWS CLI ユーザーガイドの「AWS Command Line Interface のインストール」でインストールの方法を確認してください。

時系列メトリクスの取得

GetResourceMetrics オペレーションでは、1 つ以上の時系列メトリクスを Performance Insights データから取得します。GetResourceMetrics には、メトリクスおよび期間が必要であり、データポイントのリストを含むレスポンスが返ります。

例えば、AWS Management Console は、次のイメージのように、[カウンターメトリクス] チャートと [データベースロード] チャートの入力に GetResourceMetrics を使用します。



GetResourceMetrics によって返るメトリクスはすべて、db.load の例外を除き、スタンダードの時系列メトリクスです。このメトリクスは、[データベースロード] グラフに表示されます。この db.load メトリクスは、ディメンションと呼ばれるサブコンポーネントに分割できるため、他の時系列メトリクスとは異なります。前のイメージでは、db.load は分割され、db.load を構成する待機状態によってグループ化されています。

> **Note**
>
> GetResourceMetrics は、db.sampleload メトリクスを返すこともできますが、通常 db.load メトリクスが適切です。

GetResourceMetrics により返されるカウンターメトリクスに関する情報は、「Performance Insights カウンターメトリクス」を参照してください。

以下の計算は、メトリクスにサポートされています。

- 平均 - 期間中のメトリクスの平均値。.avg をメトリクス名に追加します。
- 最小 - 期間中のメトリクスの最小値。.min をメトリクス名に追加します。
- 最大 - 期間中のメトリクスの最大値。.max をメトリクス名に追加します。
- 合計 - 期間中のメトリクス値の合計。.sum をメトリクス名に追加します。
- サンプル数 - 期間中にメトリクスが収集された回数。.sample_count をメトリクス名に追加します。

例えば、メトリクスが 300 秒 (5 分) 収集され、メトリクスが 1 分に 1 回収集されたものと見なします。毎分の値は、1、2、3、4、5 です。この場合、以下の計算が返されます。

- 平均 - 3
- 最小 - 1
- 最大 - 5
- 合計 - 15
- サンプル数 - 5

get-resource-metrics AWS CLI コマンドの使用の詳細については、「get-resource-metrics」を参照してください。

--metric-queries オプションでは、結果を取得する 1 つ以上のクエリを指定します。各クエリは、必須の Metric と、オプションの GroupBy および Filter パラメータから構成されます。--metric-queries オプションの指定の例を次に示します。

```
{
   "Metric": "string",
   "GroupBy": {
     "Group": "string",
     "Dimensions": ["string", ...],
     "Limit": integer
   },
   "Filter": {"string": "string"
     ...}
```

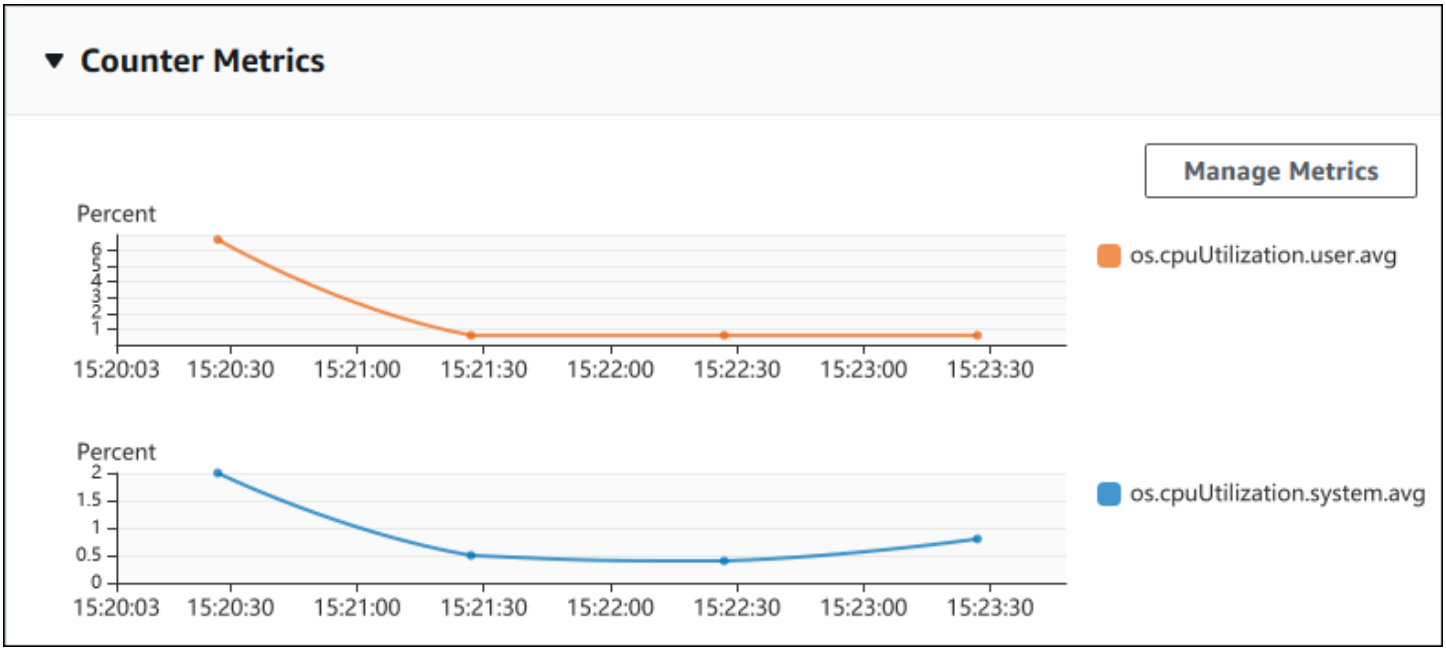
Performance Insights での AWS CLI の例

次の例は、Performance Insights のための AWS CLI の使用方法を示しています。

トピック

カウンターメトリクスの取得

以下のスクリーンショットは、AWS Management Console における 2 つのカウンターメトリクスグラフを示します。



以下の例では、2 つのカウンターメトリクスグラフを生成するために AWS Management Console で使用するデータと同じデータを生成する方法を示します。

Linux、macOS、Unix の場合:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

Windows の場合:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

また、コマンドを作成しやすくするために、--metrics-query オプションにファイルを指定します。以下の例では、このオプション用に query.json と呼ばれるファイルを使用します。ファイルの内容は次のとおりです。

```
[
    {
        "Metric": "os.cpuUtilization.user.avg"
    },
    {
        "Metric": "os.cpuUtilization.idle.avg"
    }
]
```

ファイルを使用するには、次のコマンドを実行します。

Linux、macOS、Unix の場合:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

Windows の場合:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

前述の例では、各オプションに次の値を指定します。

- --service-type - RDS for Amazon RDS
- --identifier - DB インスタンスのリソース ID
- --start-time および --end-time - クエリを実行する期間の ISO 8601 DateTime 値 (サポートされている複数の形式)

クエリは 1 時間の範囲で実行されます。

- `--period-in-seconds` - `60` (1 分ごとのクエリ)
- `--metric-queries` - 2 つのクエリの配列。それぞれ 1 つのメトリクスに対して使用されます。

  メトリクス名ではドットを使用してメトリクスを有用なカテゴリに分類します。最終の要素は関数になります。この例では、関数は、クエリの `avg` です。Amazon CloudWatch と同様に、サポートされている関数は、`min`、`max`、`total`、および `avg` です。

レスポンスは次の例のようになります。

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1540857600.0,
    "AlignedEndTime": 1540861200.0,
    "MetricList": [
        { //A list of key/datapoints
            "Key": {
                "Metric": "os.cpuUtilization.user.avg" //Metric1
            },
            "DataPoints": [
                //Each list of datapoints has the same timestamps and same number of
 items
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 4.0
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 4.0
                },
                {
                    "Timestamp": 1540857780.0, //Minute 3
                    "Value": 10.0
                }
                //... 60 datapoints for the os.cpuUtilization.user.avg metric
            ]
        },
        {
            "Key": {
                "Metric": "os.cpuUtilization.idle.avg" //Metric2
            },
```

```
            "DataPoints": [
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 12.0
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 13.5
                },
                //... 60 datapoints for the os.cpuUtilization.idle.avg metric
            ]
        }
    ] //end of MetricList
} //end of response
```

レスポンスには、Identifier、AlignedStartTime、AlignedEndTime があります。--period-in-seconds 値が 60 の場合、スタート時間および終了時間は、時間 (分) に調整されます。--period-in-seconds が 3600 の場合、スタート時間および終了時間は、時間 (時) に調整されます。

レスポンスの MetricList には、多数のエントリを含み、それぞれに Key および DataPoints エントリがあります。DataPoint にはそれぞれ、Timestamp および Value を含みます。クエリは 1 分ごとのデータが 1 時間以上実行されるため、Datapoints の各リストには、60 個のデータポイントがあります。これには、Timestamp1/Minute1 や Timestamp2/Minute2 から、Timestamp60/Minute60 まで含まれます。

クエリは 2 つの異なるカウンターメトリクスを対象としているため、レスポンス MetricList には 2 つの要素があります。

上位の待機イベントに関する DB 平均負荷の取得

以下の例は、スタックされたエリアチャートを生成するために AWS Management Console で使用されるのと同じクエリです。この例では、上位 7 つの待機イベントに応じて負荷を分割し、最後の 1 時間で db.load.avg を取得します。コマンドは カウンターメトリクスの取得 と同じコマンドです。ただし、query.json ファイルには、次の内容が含まれます。

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_event", "Limit": 7 }
    }
```

```
]
```

以下のコマンドを実行します。

Linux、macOS、Unix の場合:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

Windows の場合:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

この例では、上位 7 つの待機イベントのうち db.load.avg と GroupBy のメトリクスを指定しています。この例の有効な値の詳細については、Performance Insights の API リファレンスの「DimensionGroup」を参照してください。

レスポンスは次の例のようになります。

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1540857600.0,
    "AlignedEndTime": 1540861200.0,
    "MetricList": [
        { //A list of key/datapoints
            "Key": {
                //A Metric with no dimensions. This is the total db.load.avg
                "Metric": "db.load.avg"
            },
            "DataPoints": [
```

```
                //Each list of datapoints has the same timestamps and same number of
 items
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 0.5166666666666667
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 0.38333333333333336
                },
                {
                    "Timestamp": 1540857780.0, //Minute 3
                    "Value": 0.26666666666666666
                }
                //... 60 datapoints for the total db.load.avg key
            ]
        },
        {
            "Key": {
                //Another key. This is db.load.avg broken down by CPU
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.name": "CPU",
                    "db.wait_event.type": "CPU"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 0.35
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 0.15
                },
                //... 60 datapoints for the CPU key
            ]
        },
        //... In total we have 8 key/datapoints entries, 1) total, 2-8) Top Wait Events
    ] //end of MetricList
} //end of response
```

このレスポンスでは、MetricList の 8 つのエントリがあります。合計の db.load.avg のエントリが 1 つあり、上位 7 つの待機イベントのいずれかに従って分割された db.load.avg のエントリが 7 つあります。初期の例とは異なり、グループ化ディメンションがあったため、メトリクスのグループ化ごとに 1 つのキーが必要です。基本的なカウンターメトリクスのユースケースのように、メトリクスごとに 1 つのキーのみ使用することはできません。

上位の SQL に関する DB 平均負荷の取得

以下の例では、上位 10 個の SQL ステートメント別に db.wait_events をグループ化します。SQL ステートメントには 2 つの異なるグループがあります。

- db.sql - SQL のフルステートメント (例:select * from customers where customer_id = 123)
- db.sql_tokenized - トークン分割された SQL ステートメント select * from customers where customer_id = ?()

データベースのパフォーマンスを分析するときは、パラメータが異なるだけの SQL ステートメントを 1 つの論理的な項目として検討すると便利です。そのため、クエリを実行する際、db.sql_tokenized を使用することができます。ただし、特に Explain Plan を確認したい場合は、パラメータ付きの完全な SQL ステートメントと、db.sql によるクエリのグループ化を調べる方が便利な場合があります。トークン分割化された SQL と完全 SQL の間には親子関係があり、複数の完全 SQL (子) が同じトークン分割化された SQL (親) の下にグループ化されています。

この例のコマンドは、上位の待機イベントに関する DB 平均負荷の取得 のコマンドに似ています。ただし、query.json ファイルには、次の内容が含まれます。

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.sql_tokenized", "Limit": 10 }
    }
]
```

次の例では db.sql_tokenized を使用しています。

Linux、macOS、Unix の場合:

```
aws pi get-resource-metrics \
   --service-type RDS \
```

```
    --identifier db-ID \
    --start-time 2018-10-29T00:00:00Z \
    --end-time   2018-10-30T00:00:00Z \
    --period-in-seconds 3600 \
    --metric-queries file://query.json
```

Windows の場合:

```
aws pi get-resource-metrics ^
    --service-type RDS ^
    --identifier db-ID ^
    --start-time 2018-10-29T00:00:00Z ^
    --end-time   2018-10-30T00:00:00Z  ^
    --period-in-seconds 3600 ^
    --metric-queries file://query.json
```

この例では、1 時間の間隔 (秒) で 24 時間以上のクエリを実行します。

この例では、上位 7 つの待機イベントのうち db.load.avg と GroupBy のメトリクスを指定しています。この例の有効な値の詳細については、Performance Insights の API リファレンスの「DimensionGroup」を参照してください。

レスポンスは次の例のようになります。

```
{
    "AlignedStartTime": 1540771200.0,
    "AlignedEndTime": 1540857600.0,
    "Identifier": "db-XXX",

    "MetricList": [ //11 entries in the MetricList
        {
            "Key": { //First key is total
                "Metric": "db.load.avg"
            }
            "DataPoints": [ //Each DataPoints list has 24 per-hour Timestamps and a
 value
                {
                    "Value": 1.6964980544747081,
                    "Timestamp": 1540774800.0
                },
                //... 24 datapoints
            ]
        },
```

```
        {
            "Key": { //Next key is the top tokenized SQL
                "Dimensions": {
                    "db.sql_tokenized.statement": "INSERT INTO authors (id,name,email)
 VALUES\n( nextval(?)  ,?,?)",
                    "db.sql_tokenized.db_id": "pi-2372568224",
                    "db.sql_tokenized.id": "AKIAIOSFODNN7EXAMPLE"
                },
                "Metric": "db.load.avg"
            },
            "DataPoints": [ //... 24 datapoints
            ]
        },
        // In total 11 entries, 10 Keys of top tokenized SQL, 1 total key
    ] //End of MetricList
} //End of response
```

このレスポンスの MetricList には 11 のエントリがあり (合計が 1 つと、トークン分割された上位 10 項目の SQL)、各エントリには、1 時間あたり 24 の DataPoints があります。

トークン分割された SQL の場合は、各ディメンションリストに 3 つのエントリがあります。

- db.sql_tokenized.statement - トークン分割された SQL ステートメント。
- db.sql_tokenized.db_id - SQL の参照に使用されていたネイティブデータベース ID、または Performance Insights によって生成される合成 ID (ネイティブデータベース ID が利用できない場合)。この例では、pi-2372568224 合成 ID が返ります。
- db.sql_tokenized.id - Performance Insights 内のクエリの ID。

  AWS Management Console で、この ID はサポート ID と呼ばれます。ID は、データベースに関する問題のトラブルシューティングに役立つ、AWS サポートが調査できるデータであるため、この名前が付けられています。AWS は、データのセキュリティとプライバシーを非常に真剣に受け止め、ほとんどすべてのデータが AWS KMS カスタマーマスターキー (CMK) で暗号化されて保存されます。そのため、このデータを AWS 内で見ることはできません。前の例では、tokenized.statement と tokenized.db_id の両方が暗号化されて保存されます。データベースに問題がある場合は、AWS サポートがサポート ID を参照して問題を解決できるようお手伝いします。

クエリを実行する際、Group で GroupBy を指定した方が便利な場合があります。ただし、返るデータを詳細に制御できるように、ディメンションのリストを指定します。例えば、必要なデータが

db.sql_tokenized.statement のみの場合は、Dimensions 属性を query.json ファイルに追加することができます。

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": {
            "Group": "db.sql_tokenized",
            "Dimensions":["db.sql_tokenized.statement"],
            "Limit": 10
        }
    }
]
```

SQL によってフィルタリングされた平均 DB ロードの取得

上のイメージは、特定のクエリが選択されていることを示しています。上位の平均アクティブセッションのスタックされたエリアグラフはそのクエリを対象としています。クエリは、依然として上位 7 つの全体的な待機イベントを対象としていますが、レスポンスの値はフィルタリングされます。フィルタでは、特定のフィルタに一致するセッションのみが考慮されます。

この例に対応する API クエリは、上位の SQL に関する DB 平均負荷の取得 のコマンドに似ています。ただし、query.json ファイルには、次の内容が含まれます。

```
[
 {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_event", "Limit": 5  },
        "Filter": { "db.sql_tokenized.id": "AKIAIOSFODNN7EXAMPLE" }
    }
]
```

Linux、macOS、Unix の場合:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

Windows の場合:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

レスポンスは次の例のようになります。

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1556215200.0,
    "MetricList": [
        {
            "Key": {
                "Metric": "db.load.avg"
            },
            "DataPoints": [
```

```
                {
                    "Timestamp": 1556218800.0,
                    "Value": 1.4878117913832196
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 1.192823803967328
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "io",
                    "db.wait_event.name": "wait/io/aurora_redo_log_flush"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 1.1360544217687074
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 1.058051341890315
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "io",
                    "db.wait_event.name": "wait/io/table/sql/handler"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.16241496598639457
                },
                {
                    "Timestamp": 1556222400.0,
```

```
                    "Value": 0.05163360560093349
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "synch",
                    "db.wait_event.name": "wait/synch/mutex/innodb/
aurora_lock_thread_slot_futex"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.11479591836734694
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.013127187864644107
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "CPU",
                    "db.wait_event.name": "CPU"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.05215419501133787
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.05805134189031505
                }
            ]
        },
        {
```

```
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "synch",
                    "db.wait_event.name": "wait/synch/mutex/innodb/lock_wait_mutex"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.017573696145124718
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.002333722287047841
                }
            ]
        }
    ],
    "AlignedEndTime": 1556222400.0
} //end of response
```

このレスポンスでは、query.json ファイルで指定されているトークン分割化された SQL AKIAIOSFODNN7EXAMPLE の割合に従って、値はすべてフィルタリングされます。キーは、フィルタなしのクエリとは異なる順序で表示されることもあります。これは、フィルタ処理された SQL に影響を与えるのは上位 5 つの待機イベントであるためです。

SQL ステートメントの全文の取得

次の例では、DB インスタンス db-10BCD2EFGHIJ3KL4M5NO6PQRS5 の SQL ステートメントの全文を取得します。`--group` は `db.sql` であり、`--group-identifier` は `db.sql.id` です。この例では、*`my-sql-id`* は、`pi get-resource-metrics` または `pi describe-dimension-keys` を呼び出して取得した SQL ID を表します。

以下のコマンドを実行します。

Linux、macOS、Unix の場合:

```
aws pi get-dimension-key-details \
   --service-type RDS \
   --identifier db-10BCD2EFGHIJ3KL4M5NO6PQRS5 \
```

```
    --group db.sql \
    --group-identifier my-sql-id \
    --requested-dimensions statement
```

Windows の場合:

```
aws pi get-dimension-key-details ^
    --service-type RDS ^
    --identifier db-10BCD2EFGHIJ3KL4M5NO6PQRS5 ^
    --group db.sql ^
    --group-identifier my-sql-id ^
    --requested-dimensions statement
```

この例では、ディメンションの詳細を使用できます。したがって、Performance Insights は、SQL ステートメントを切り捨てることなく、その全文を取得します。

```
{
    "Dimensions":[
    {
        "Value": "SELECT e.last_name, d.department_name FROM employees e, departments d
 WHERE e.department_id=d.department_id",
        "Dimension": "db.sql.statement",
        "Status": "AVAILABLE"
    },
    ...
    ]
}
```

一定期間のパフォーマンス分析レポートの作成

次の例では、db-loadtest-0 データベースの 1682969503 開始時間と 1682979503 終了時間を使用してパフォーマンス分析レポートを作成します。

```
        aws pi-test create-performance-analysis-report \
        --service-type RDS \
        --identifier db-loadtest-0 \
        --start-time 1682969503 \
        --end-time 1682979503 \
        --endpoint-url https://api.titan.pi.a2z.com \
        --region us-west-2
```

レスポンスは、レポートの一意識別子 report-0234d3ed98e28fb17 です。

```
      {
         "AnalysisReportId": "report-0234d3ed98e28fb17"
      }
```

パフォーマンス分析レポートの取得

次の例では、report-0d99cc91c4422ee61 レポートについて分析レポートの詳細を取得します。

```
        aws pi-test get-performance-analysis-report \
--service-type RDS \
--identifier db-loadtest-0 \
--analysis-report-id report-0d99cc91c4422ee61 \
--endpoint-url https://api.titan.pi.a2z.com \
--region us-west-2
```

レスポンスには、レポートのステータス、ID、時間の詳細、およびインサイトが表示されます。

```
        {
    "AnalysisReport": {
        "Status": "Succeeded",
        "ServiceType": "RDS",
        "Identifier": "db-loadtest-0",
        "StartTime": 1680583486.584,
        "AnalysisReportId": "report-0d99cc91c4422ee61",
        "EndTime": 1680587086.584,
        "CreateTime": 1680587087.139,
        "Insights": [
           ... (Condensed for space)
        ]
    }
}
```

DB インスタンスのすべてのパフォーマンス分析レポートを一覧表示する

次の例は、db-loadtest-0 データベースについて入手可能なすべてのパフォーマンス分析レポートを一覧表示します。

```
           aws pi-test list-performance-analysis-reports \
--service-type RDS \
--identifier db-loadtest-0 \
--endpoint-url https://api.titan.pi.a2z.com \
--region us-west-2
```

レスポンスには、すべてのレポートがレポートの ID、ステータス、および時間の詳細とともに一覧表示されます。

```
           {
    "AnalysisReports": [
        {
            "Status": "Succeeded",
            "EndTime": 1680587086.584,
            "CreationTime": 1680587087.139,
            "StartTime": 1680583486.584,
            "AnalysisReportId": "report-0d99cc91c4422ee61"
        },
        {
            "Status": "Succeeded",
            "EndTime": 1681491137.914,
            "CreationTime": 1681491145.973,
            "StartTime": 1681487537.914,
            "AnalysisReportId": "report-002633115cc002233"
        },
        {
            "Status": "Succeeded",
            "EndTime": 1681493499.849,
            "CreationTime": 1681493507.762,
            "StartTime": 1681489899.849,
            "AnalysisReportId": "report-043b1e006b47246f9"
        },
        {
            "Status": "InProgress",
            "EndTime": 1682979503.0,
```

```
            "CreationTime": 1682979618.994,
            "StartTime": 1682969503.0,
            "AnalysisReportId": "report-01ad15f9b88bcbd56"
        }
    ]
}
```

パフォーマンス分析レポートの削除

次の例では、db-loadtest-0 データベースの分析レポートを削除します。

```
        aws pi-test delete-performance-analysis-report \
--service-type RDS \
--identifier db-loadtest-0 \
--analysis-report-id report-0d99cc91c4422ee61 \
--endpoint-url https://api.titan.pi.a2z.com \
--region us-west-2
```

パフォーマンス分析レポートにタグを追加する

次の例では、キー name と値 test-tag のタグを report-01ad15f9b88bcbd56 レポートに追加します。

```
            aws pi-test tag-resource \
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--tags Key=name,Value=test-tag \
--endpoint-url https://api.titan.pi.a2z.com \
--region us-west-2
```

パフォーマンス分析レポートのすべてのタグを一覧表示する

次の例では、report-01ad15f9b88bcbd56 レポートのすべてのタグを一覧表示します。

```
        aws pi-test list-tags-for-resource \
```

```
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--endpoint-url https://api.titan.pi.a2z.com \
--region us-west-2
```

レスポンスには、レポートに追加されたすべてのタグの値とキーが一覧表示されます。

```
        {
    "Tags": [
        {
            "Value": "test-tag",
            "Key": "name"
        }
    ]
}
```

パフォーマンス分析レポートからタグを削除する

次の例では、report-01ad15f9b88bcbd56 レポートから name タグを削除します。

```
        aws pi-test untag-resource \
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--tag-keys name \
--endpoint-url https://api.titan.pi.a2z.com \
--region us-west-2
```

タグを削除した後、list-tags-for-resource API を呼び出しても、このタグは一覧に表示されません。

AWS CloudTrail を使用した Performance Insights 呼び出しの口グ記録

Performance Insights は AWS CloudTrail サービスと連携して動作します。このサービスは、Performance Insights でユーザー、ロール、または AWS のサービスによって実行されたアクションを記録します。CloudTrail は、Performance Insights のすべての API コールをイベントとして

キャプチャします。このキャプチャには、Amazon RDS コンソールからのコールと、Performance Insights API オペレーションへのコードコールが含まれます。

証跡を作成すると、Performance Insights のイベントも含めて、Amazon S3 バケットへの CloudTrail イベントの継続的配信を有効にすることができます。追跡を設定しない場合でも、CloudTrail コンソールの Event history (イベント履歴)で最新のイベントを表示できます。CloudTrail によって収集されたデータを使用して、多くの情報を判断できます。この情報には、Performance Insights に対するリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時が含まれます。追加の詳細も含まれています。

CloudTrail の詳細については、AWS CloudTrail ユーザーガイドを参照してください。

CloudTrail での Performance Insights 情報の使用

AWS アカウントを作成すると、そのアカウントに対して CloudTrail が有効になります。Performance Insights でアクティビティが発生すると、そのアクティビティは、CloudTrail イベントとして AWS の他のサービスのイベントとともに、CloudTrail コンソールの [イベント履歴] に記録されます。AWS アカウントで最近のイベントを表示、検索、ダウンロードできます。詳細については、AWS CloudTrail ユーザーガイドの「CloudTrail イベント履歴でのイベントの表示」を参照してください。

Performance Insights のイベントなど、AWS アカウントのイベントを継続的に記録する場合は、証跡を作成します。証跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コンソールで追跡を作成するときに、追跡がすべての AWS リージョンに適用されます。追跡では、AWS パーティション内のすべての AWS リージョンからのイベントをログに記録し、指定した Simple Storage Service (Amazon S3)バケットにログファイルを配信します。さらに、CloudTrail· ログで収集したイベントデータをより詳細に分析し、それに基づく対応するためにその他の AWS のサービスを設定できます。詳細については、AWS CloudTrail ユーザーガイドの次のトピックを参照してください。

- 追跡を作成するための概要
- CloudTrail のサポート対象サービスと統合
- Amazon SNS の CloudTrail の通知の設定
- 「複数のリージョンから CloudTrail ログファイルを受け取る」および「複数のアカウントから CloudTrail ログファイルを受け取る」

すべての Performance Insights オペレーションは CloudTrail によってログに記録されます。また、Performance Insights API リファレンスに記載されています。例え

ば、DescribeDimensionKeys オペレーションと GetResourceMetrics オペレーションへのコールに伴って、CloudTrail ログファイルにエントリが生成されます。

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。同一性情報は次の判断に役立ちます。

- リクエストが、ルートと IAM ユーザー認証情報のどちらを使用して送信されたか。
- リクエストが、ロールとフェデレーティッドユーザーのどちらの一時的なセキュリティ認証情報を使用して送信されたか.
- リクエストが、別の AWS のサービスによって送信されたかどうか。

詳細については、[CloudTrail userIdentity Element] (CloudTrail ユーザーアイデンティティ要素) を参照してください。

Performance Insights のログファイルのエントリ

[トレイル] は、指定した Simple Storage Service (Amazon S3) バケットにイベントをログファイルとして配信するように設定できます。CloudTrail ログファイルには、1 つ以上のログエントリがあります。イベントは、任意の送信元からの単一のリクエストを表します。各イベントには、リクエストされたオペレーション、オペレーションの日時、リクエストパラメータなどに関する情報が含まれます。CloudTrail ログファイルは、パブリック API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

GetResourceMetrics オペレーションを示す CloudTrail ログエントリの例は、次のとおりです。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
         "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T19:28:46Z",
    "eventSource": "pi.amazonaws.com",
    "eventName": "GetResourceMetrics",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "72.21.198.67",
```

```
    "userAgent": "aws-cli/1.16.240 Python/3.7.4 Darwin/18.7.0 botocore/1.12.230",
    "requestParameters": {
        "identifier": "db-YTDU5J5V66X7CXSCVDFD2V3SZM",
        "metricQueries": [
            {
                "metric": "os.cpuUtilization.user.avg"
            },
            {
                "metric": "os.cpuUtilization.idle.avg"
            }
        ],
        "startTime": "Dec 18, 2019 5:28:46 PM",
        "periodInSeconds": 60,
        "endTime": "Dec 18, 2019 7:28:46 PM",
        "serviceType": "RDS"
    },
    "responseElements": null,
    "requestID": "9ffbe15c-96b5-4fe6-bed9-9fccff1a0525",
    "eventID": "08908de0-2431-4e2e-ba7b-f5424f908433",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

Amazon DevOps Guru for Amazon RDS でパフォーマンスの異常を分析する

Amazon DevOps Guru は、開発者およびオペレーターがアプリケーションのパフォーマンスと可用性を向上させるためのフルマネージド型オペレーションサービスです。DevOps Guru は、運用上の問題の特定に関連するタスクをオフロードし、アプリケーションを改善するための推奨事項を迅速に実装できるようにします。詳細については、「Amazon DevOps Guru ユーザーガイド」の「Amazon DevOps Guru とは」を参照してください。

DevOps Guru は、すべての Amazon RDS DB エンジンの既存の運用上の問題を検出し、分析し、推奨を行います。DevOps Guru for RDS は、Amazon Aurora データベースの Performance Insights メトリクスに機械学習を適用することで、この機能を拡張します。これらのモニタリング機能により、DevOps Guru for RDS はパフォーマンスのボトルネックを検出して診断し、具体的な修正措置を推奨できます。DevOps Guru for RDS は、Aurora データベース で問題が発生する前に問題条件を検出することもできます。

これらの推奨事項を RDS コンソールで表示できるようになりました。詳細については、「Amazon Aurora の推奨事項の表示とこれらに対する対応」を参照してください。

次の動画は DevOps Guru for RDS の概要です。

この主題の詳細については、Amazon DevOps Guru for RDS の内部を参照してください。

トピック

- DevOps Guru for RDS の利点
- DevOps Guru for RDSはどのように機能しますか
- RDS 用の DevOps Guru のセットアップ

DevOps Guru for RDS の利点

Amazon Aurora データベースを担当していて、そのデータベースに影響を与えるイベントやリグレッションの発生を知らないことがあります。問題を知っても、なぜそれが発生しているのか、どう対処すべきかわからないこともあります。データベース管理者 (DBA) に問い合わせたり、サードパーティーツールに頼ったりするのではなく、DevOps Guru for RDS のレコメンデーションに従ってください。

DevOps Guru for RDS の詳細な分析により、次の利点が得られます。

高速診断

DevOps Guru for RDS は、データベースのテレメトリを継続的にモニタリングおよび分析します。Performance Insights、拡張モニタリング、および Amazon CloudWatch は、データベースクラスターのテレメトリーデータを収集します。DevOps Guru for RDS は、統計的な機械学習の技術を使用してこのデータをマイニングし、異常を検出します。テレメトリーデータの詳細については、Amazon Aurora ユーザーガイドの「Amazon Aurora での Performance Insights を使用した DB 負荷のモニタリング」および「拡張モニタリングを使用した OS メトリクスのモニタリング」、を参照してください。

高速解像度

各異常はパフォーマンスの問題を特定し、調査または修正措置の方法を提案します。例えば、DevOps Guru for RDS では、特定の待機イベントの調査をお勧めすることがあります。または、データベース接続数を制限するよう、アプリケーションプールの設定のチューニングをお勧めすることもあります。これらのレコメンデーションに基づいて、マニュアルでトラブルシューティングを実行するよりも迅速にパフォーマンスの問題を解決できます。

事前対応型インサイト

DevOps Guru for RDS は、リソースからのメトリクスを使用して、問題となる可能性のある動作を大きな問題になる前に検出します。例えば、データベースが使用するディスク上の一時テーブルの数が増え、パフォーマンスに影響を与え始めたことを検出できます。その場合、DevOps Guru は問題が大きくなる前に対処するのに役立つ推奨事項を提供します。

Amazon エンジニアの深い知識と機械学習

パフォーマンス問題を検出し、ボトルネックの解決を支援するために、DevOps Guru for RDS は機械学習 (ML) と高度な数式に依存しています。Amazon データベースエンジニアは、数十万のデータベース管理の長年の経験をカプセル化した DevOps Guru for RDS の知見の開発に貢献しました。この集合的な知識を活かすことで、DevOps Guru for RDS はベストプラクティスを伝えることができます。

DevOps Guru for RDSはどのように機能しますか

DevOps Guru for RDS は、Amazon RDS Performance Insights から Aurora データベースに関するデータを収集します。最も重要なメトリクスは`DBLoad`です。DevOps Guru for RDS は、Performance Insights メトリクスを消費し、機械学習を使用して分析し、ダッシュボードにインサイトを公開します。

インサイトは、DevOps Guru によって検出された関連する異常のコレクションです。

DevOps Guru for RDS では、異常とは、Amazon Aurora データベースの通常のパフォーマンスから逸脱したパターンのことです。

事前対応型インサイト

プロアクティブインサイトでは、問題のある動作を発生前に知ることができます。異常と共に推奨事項と関連メトリクスも含まれるため、Amazon Aurora データベースの問題が大きな問題になる前に対処できます。これらのインサイトは、DevOps Guru ダッシュボードに発行されます。

例えば、DevOps Guru は、Aurora PostgreSQL データベースがディスク上に多数のテンポラリテーブルを作成していることを検出する場合があります。対処しなければ、この傾向はパフォーマンス問題につながる可能性があります。各プロアクティブインサイトには、是正措置に関する推奨事項と、Amazon DevOps Guru のプロアクティブインサイトによる Aurora MySQL のチューニング または Amazon DevOps Guru のプロアクティブインサイトによる Aurora PostgreSQL のチューニング 内の関連トピックへのリンクが含まれています。詳細については、「Amazon DevOps Guru ユーザーガイド」の「Working with insights in DevOps Guru」を参照してください。

事後対応型インサイト

リアクティブインサイトは、異常な動作を発生時に識別します。DevOps Guru for RDS が Amazon Aurora DB インスタンスでパフォーマンス問題を発見すると、DevOps Guru ダッシュボードにリアクティブインサイトが発行されます。詳細については、「Amazon DevOps Guru ユーザーガイド」の「Working with insights in DevOps Guru」(Amazon DevOps Guru でのインサイトの操作) を参照してください。

因果異常

因果異常は、リアクティブインサイト内のトップレベルの異常です。データベースロード (DB ロード)DevOps Guru for RDS の因果異常です。

異常は、深刻度レベルを高、中、低のいずれかに割り当てて、パフォーマンスへの影響を測定します。詳細については、「Amazon DevOps Guru ユーザーガイド」の「DevOps Guru for RDS の主要な概念」を参照してください。

DevOps Guru が DB インスタンスで現在の異常を検出すると、RDS コンソールの[Databases] (データベース) ページにアラートが表示されます。コンソールは、過去 24 時間に発生した異常についても警告します。RDS コンソールから異常ページに移動するには、アラートメッセージ内のリンクを選択します。RDS コンソールでは、Amazon Aurora DB クラスター のページでもアラートが表示されます。

コンテキスト異常

コンテキスト異常は、事後対応型インサイトに関連するデータベースロード (DB ロード)での所見です。各コンテキスト異常は、調査が必要な特定の Amazon Aurora のパフォーマンス問題を記述します。例えば、DevOps Guru for RDS は、CPU 容量の増加を検討したり、DB ロードに寄与している待機イベントを調査するよう推奨することがあります。

> **Important**
>
> 本稼働インスタンスの修正前に、各変更の影響を完全に把握できるように、テストインスタンスでの変更のテストをお勧めします。このようにして、変更の影響を理解します。

詳細については、Amazon DevOps Guru ユーザーガイドの「Analyzing anomalies in Amazon RDS」(Amazon RDS の異常を分析する) を参照してください。

RDS 用の DevOps Guru のセットアップ

DevOps Guru for Amazon RDS が Amazon Aurora データベースのインサイトを発行できるようにするには、以下のタスクを実行します。

トピック

- DevOps Guru for RDS の IAM アクセスポリシーの設定
- Aurora DB インスタンスの Performance Insights をオンにする
- DevOps Guru をオンにしてリソースカバレッジを指定する

DevOps Guru for RDS の IAM アクセスポリシーの設定

DevOps Guru からのアラートを RDS コンソールに表示するには、AWS Identity and Access Management (IAM) ユーザーまたはロールに次のいずれかのポリシーが必要です。

- AWS マネージドポリシー `AmazonDevOpsGuruConsoleFullAccess`
- AWS 管理ポリシー `AmazonDevOpsGuruConsoleReadOnlyAccess` および次のいずれかのポリシーが必要です。
  - AWS マネージドポリシー `AmazonRDSFullAccess`
  - `pi:GetResourceMetrics` と `pi:DescribeDimensionKeys` を含むカスタマー管理ポリシー

詳細については、「Performance Insights 用のアクセスポリシーの設定」を参照してください。

Aurora DB インスタンスの Performance Insights をオンにする

DevOps Guru for RDS は、データの Performance Insights に依存しています。Performance Insights がなければ、DevOps Guru は異常を公開しますが、詳細な分析と推奨事項は含まれません。

Aurora DB クラスターを作成またはクラスターインスタンスを変更するときに、Performance Insights をオンにすることができます。詳細については、「Performance Insights の有効化と無効化」を参照してください。

DevOps Guru をオンにしてリソースカバレッジを指定する

DevOps Guru をオンにして、次のいずれかの方法で Amazon Aurora データベースをモニタリングできます。

トピック

- RDS コンソールで DevOps Guru をオンにする
- Aurora リソースを DevOps Guru コンソールに追加する
- AWS CloudFormation を使用して Aurora リソースを追加する

RDS コンソールで DevOps Guru をオンにする

Amazon RDS コンソールで複数のパスを取得して DevOps Guru をオンにすることができます。

トピック

- Aurora データベースを作成するときに DevOps Guru をオンにする
- 通知バナーから DevOps Guru をオンにする
- DevOps Guru をオンにしたときのアクセス許可エラーへの応答

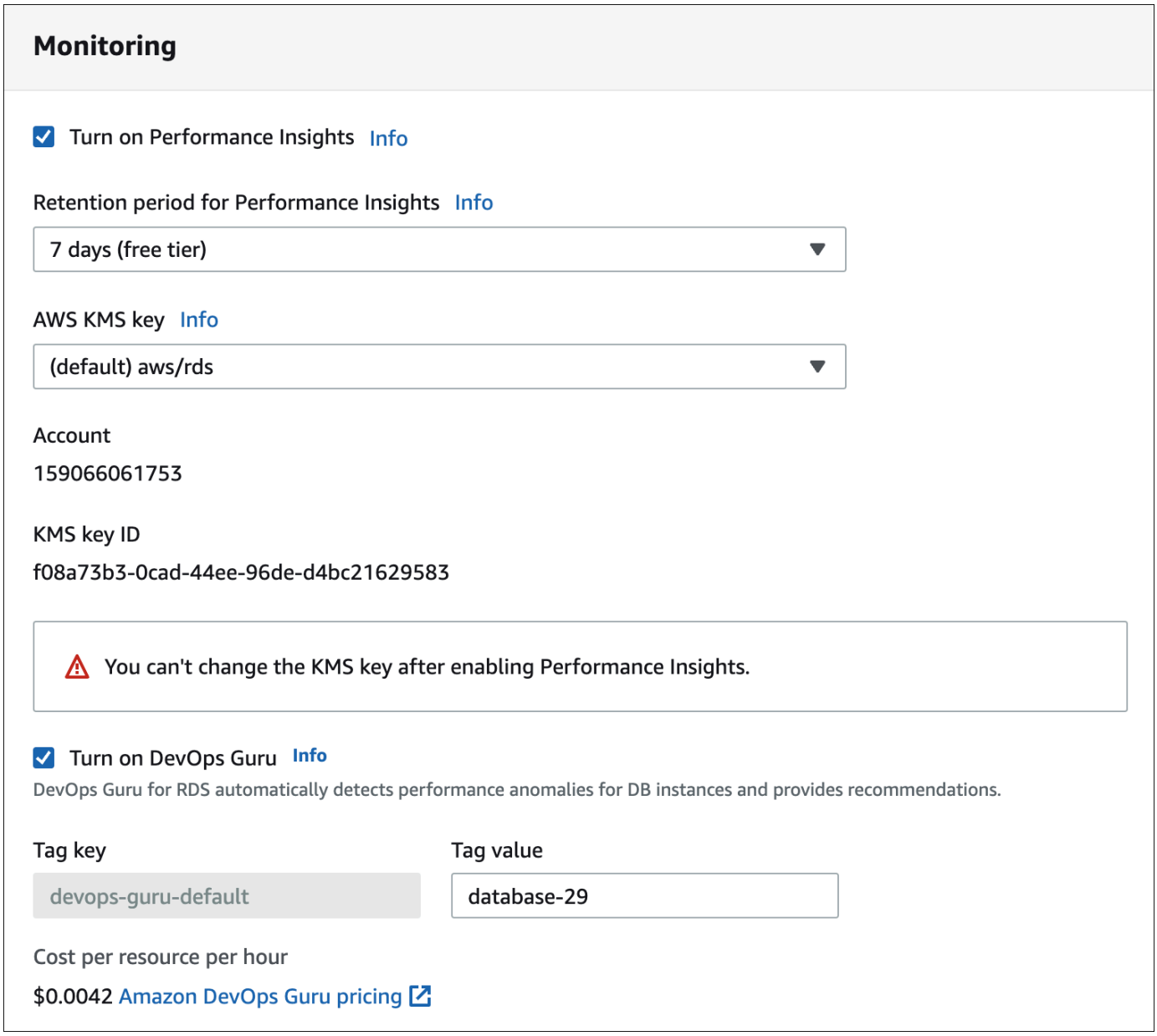
Aurora データベースを作成するときに DevOps Guru をオンにする

作成ワークフローには、データベースの DevOps Guru カバレッジを有効にする設定が含まれています。本番稼働用テンプレートを選択した場合、この設定はデフォルトでオンになっています。

Aurora データベースを作成するときに DevOps Guru をオンにするには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。

2. 「DB クラスターの作成」で、モニタリング設定を選択する手順まで (ただしその手順は含まない)の手順を行います。
3. [Monitoring] (モニタリング) で、[Turn on Performance Insights] (Performance Insights をオンにする) を選択します。DevOps Guru for RDS で、パフォーマンスの異常の詳細な分析を提供するには、[Performance Insights] (パフォーマンスインサイト) をオンにする必要があります。
4. [Turn on DevOps Guru] (DevOps Guru をオンにする) を選択します。



5. DevOps Guru がそれをモニタリングできるように、データベースのタグを作成します。以下の操作を実行します。
   - [Tag key] (タグキー) のテキストフィールドで、**`Devops-Guru-`** で始まる名前を入力します。

- [Tag value] (タグ値) のテキストフィールドで、任意の値を入力します。例えば、Aurora データベースの名前として **rds-database-1** と入力した場合、タグ値として **rds-database-1** も入力することができます。

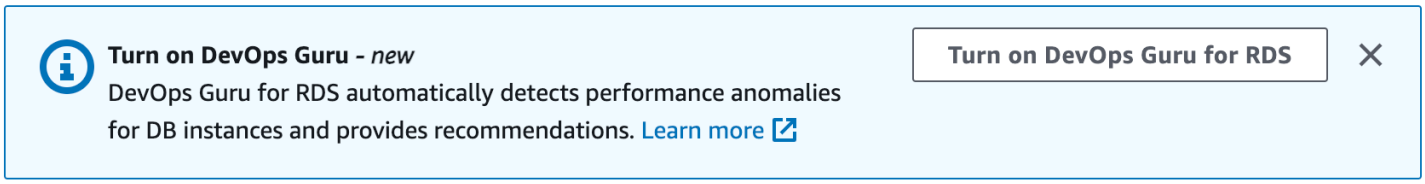
タグの詳細については、「Amazon DevOps Guru ユーザーガイド」の「タグを使用して DevOps Guru アプリケーションのリソースを特定する」を参照してください。

6. 「DB クラスターの作成」の残りのステップを行います。

通知バナーから DevOps Guru をオンにする

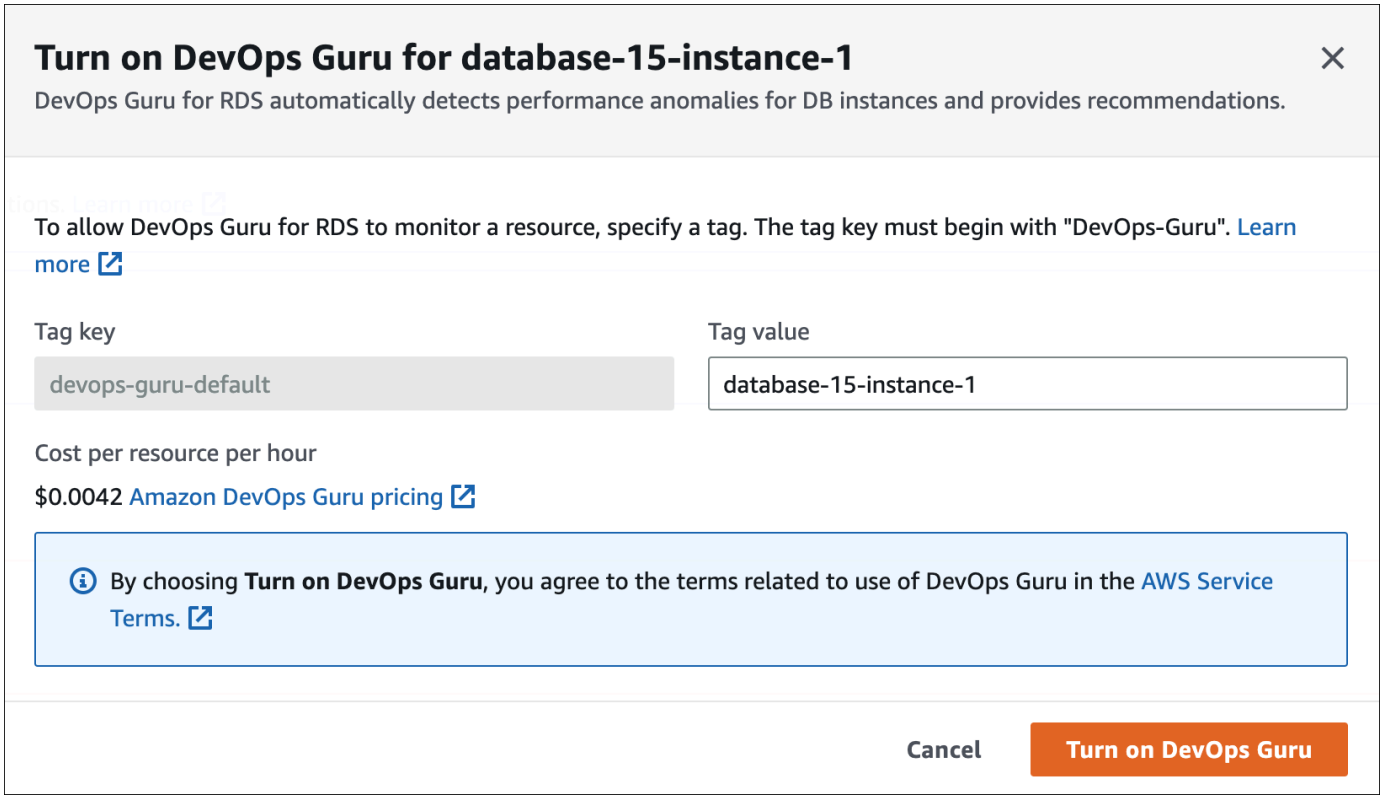
リソースが DevOps Guru の対象外である場合、Amazon RDS は次の場所のバナーで通知します。

- DB クラスターインスタンスの[Monitoring] (モニタリング) タブ
- Performance Insights ダッシュボード



Aurora データベースについて DevOps Guru をオンにするには

1. バナーで、[Turn on DevOps Guru for RDS] (DevOps Guru for RDS をオンにする) を選択します。
2. タグのキー名と値を入力します。タグの詳細については、「Amazon DevOps Guru ユーザーガイド」の「タグを使用して DevOps Guru アプリケーションのリソースを特定する」を参照してください。



3. [Turn on DevOps Guru] (DevOps Guru をオンにする) を選択します。

DevOps Guru をオンにしたときのアクセス許可エラーへの応答

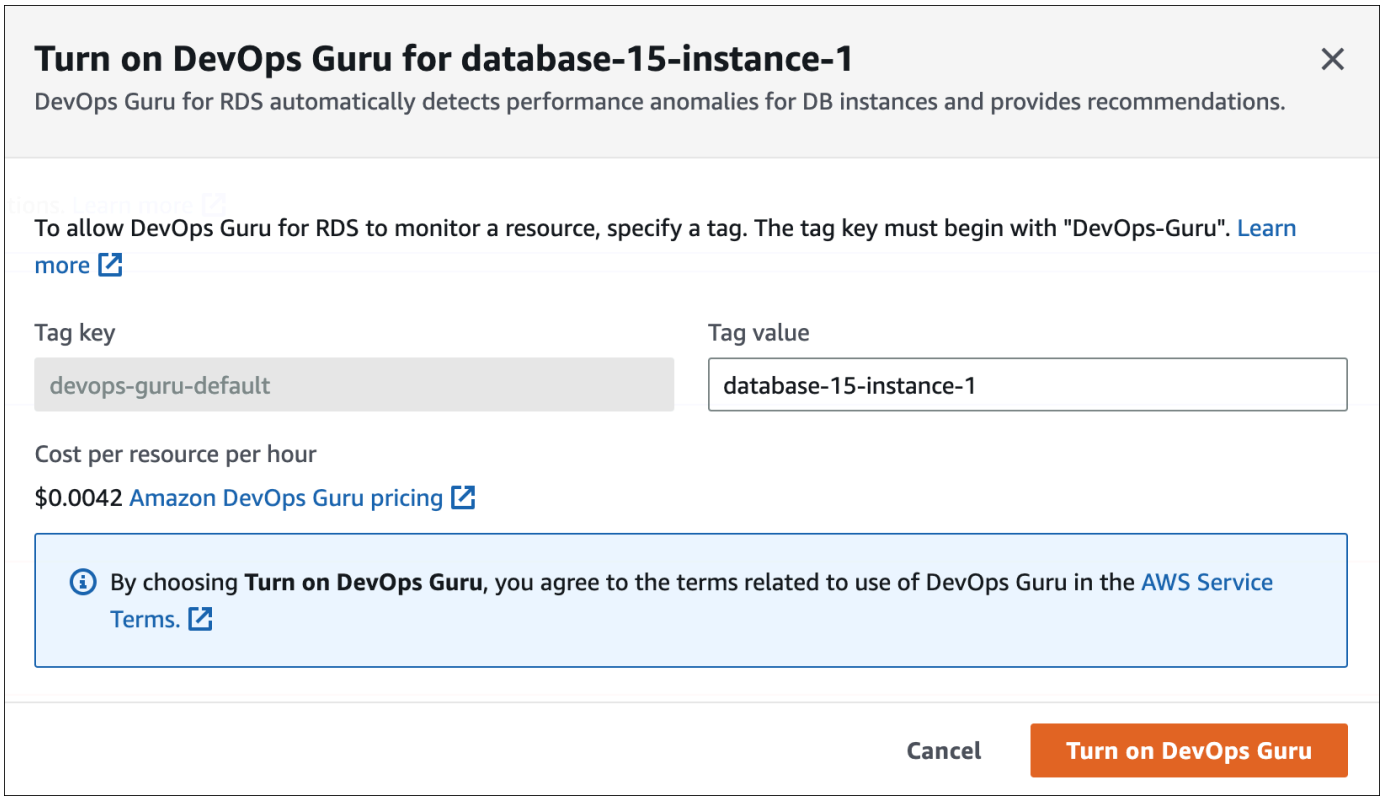
データベースを作成するときに RDS コンソールから DevOps Guru をオンにすると、RDS にアクセス許可がないことについて次のバナーが表示されることがあります。

アクセス許可エラーに応答するには

1. IAM ユーザーまたはロールにユーザー管理ロール `AmazonDevOpsGuruConsoleFullAccess` を付与します。詳細については、「DevOps Guru for RDS の IAM アクセスポリシーの設定」を参照してください。
2. RDS コンソールを開きます。
3. ナビゲーションペインで、[Performance Insights] を選択します。
4. 先ほど作成したクラスターで DB インスタンスを選択します。
5. スイッチを選択して、[DevOps Guru for RDS] をオンにします。

DevOps Guru for RDS

6. タグ値を選択します。詳細については、「Amazon DevOps Guru ユーザーガイド」の「タグを使用して DevOps Guru アプリケーションのリソースを特定する」を参照してください。



7. [Turn on DevOps Guru] (DevOps Guru をオンにする) を選択します。

Aurora リソースを DevOps Guru コンソールに追加する

DevOps Guru コンソールで DevOps Guru リソースカバレッジを指定できます。「Amazon DevOps Guru ユーザーガイド」の「DevOps Guru リソースカバレッジを指定する」で説明されている手順に従います。分析リソースを編集する場合は、以下のオプションのいずれかを選択します。

- Aurora データベースも含め、AWS アカウント とリージョンでサポートされているすべてのリソースを分析するには、[すべてのアカウントリソース] を選択します。
- 選択したスタック内の Aurora データベースを分析するには、[CloudFormation スタック] を選択します。詳細については、「Amazon DevOps Guru ユーザーガイド」の「AWS CloudFormation スタックを使用して DevOps Guru アプリケーション内のリソースを識別する」を参照してください。

- タグ付けした Aurora データベースを分析するには、[タグ] を選択します。詳細については、「Amazon DevOps Guru ユーザーガイド」の「タグを使用して DevOps Guru アプリケーションのリソースを特定する」を参照してください。

詳細については、「Amazon DevOps Guru ユーザーガイド」の 「Amazon DevOps Guru の有効化」を参照してください。

AWS CloudFormation を使用して Aurora リソースを追加する

タグを使用して、Aurora リソースのカバレッジを CloudFormation テンプレートに追加できます。次の手順では、Aurora DB インスタンスと DevOps Guru スタックの両方についての CloudFormation テンプレートがあることを前提としています。

CloudFormation タグを使用して Aurora DB インスタンスを指定するには

1. DB インスタンスの CloudFormation テンプレートで、キーと値のペアを使用してタグを定義します。

   次の例では、値 `my-aurora-db-instance1` を Aurora DB インスタンスの `Devops-guru-cfn-default` に割り当てます。

   ```
   MyAuroraDBInstance1:
     Type: "AWS::RDS::DBInstance"
     Properties:
       DBClusterIdentifier: my-aurora-db-cluster
       DBInstanceIdentifier: my-aurora-db-instance1
       Tags:
         - Key: Devops-guru-cfn-default
           Value: devopsguru-my-aurora-db-instance1
   ```

2. DevOps Guru スタックの CloudFormation テンプレートで、リソースコレクションフィルターに同じタグを指定します。

   次の例では、タグ値 `my-aurora-db-instance1` を持つリソースをカバーするように DevOps Guru を設定します。

   ```
   DevOpsGuruResourceCollection:
     Type: AWS::DevOpsGuru::ResourceCollection
     Properties:
       ResourceCollectionFilter:
         Tags:
   ```

```
        - AppBoundaryKey: "Devops-guru-cfn-default"
          TagValues:
          - "devopsguru-my-aurora-db-instance1"
```

次の例では、アプリケーション境界 Devops-guru-cfn-default 内のすべてのリソースを対象としています。

```
DevOpsGuruResourceCollection:
  Type: AWS::DevOpsGuru::ResourceCollection
  Properties:
    ResourceCollectionFilter:
      Tags:
        - AppBoundaryKey: "Devops-guru-cfn-default"
          TagValues:
          - "*"
```

詳細については、「AWS CloudFormation ユーザーガイド」の「AWS::DevOpsGuru::ResourceCollection」と「AWS::RDS::DBInstance」を参照してください。

拡張モニタリングを使用した OS メトリクスのモニタリング

Enhanced Monitoring を使用すると、DB インスタンスのオペレーティングシステムをリアルタイムでモニタリングできます。さまざまなプロセスまたはスレッドが CPU をどのように使用しているかを確認するには、Enhanced Monitoring メトリクスが役立ちます。

トピック

Enhanced Monitoring の概要

Amazon RDS には、DB インスタンスが実行されているオペレーティングシステム (OS) のリアルタイムのメトリクスが用意されています。RDS DB インスタンスのすべてのシステムメトリクスとプロセス情報をコンソールに表示できます。各インスタンスでモニタリングするメトリクスを管理し、要件に応じてダッシュボードをカスタマイズできます。拡張モニタリングメトリクスの説明については、「拡張モニタリングの OS メトリクス」を参照してください。

RDS は、拡張モニタリングのメトリクスを Amazon CloudWatch Logs アカウントに配信します。CloudWatch Logs から CloudWatch のメトリクスフィルタを作成し、CloudWatch ダッシュボードにグラフを表示できます。選択したモニタリングシステムで CloudWatch Logs からの拡張モニタリング JSON 出力を使用できます。詳細については、Amazon RDS に関するよくある質問の「拡張モニタリング」を参照してください。

トピック

CloudWatch と拡張モニタリングのメトリクスの相違点

ハイパーバイザーは、仮想マシン (VM) を作成して実行します。ハイパーバイザーを使用すると、メモリと CPU を仮想的に共有することで、1 つのインスタンスで複数のゲスト VM をサポートできま

す。CloudWatch は DB インスタンスのハイパーバイザーから CPU 使用率のメトリクスを収集します。対照的に、Enhanced Monitoring は DB インスタンス上のエージェントからメトリクスを収集します。

ハイパーバイザーレイヤーで少量の処理が実行されるため、CloudWatch と Enhanced Monitoring 測定値の間に違いが見つかることがあります。DB インスタンスがより小さなインスタンスクラスを使用している場合、その違いはより大きくなる可能性があります。このシナリオでは、1 つの物理インスタンス上のハイパーバイザー層によってより多くの仮想マシン (VM) が管理されている可能性があります。

拡張モニタリングメトリクスの説明については、「拡張モニタリングの OS メトリクス」を参照してください。Amazon CloudWatch メトリクスの詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。

Enhanced Monitoring メトリクスの保持

デフォルトでは、Enhanced Monitoring メトリクスは CloudWatch Logs で 30 日間保存されます。この保持期間は、通常の CloudWatch メトリクスとは異なります。

メトリクスが CloudWatch Logs に保存される時間の長さを変更するには、CloudWatch コンソールの `RDSOSMetrics` ロググループの保存期間を変更します。詳細については、Amazon CloudWatch Logs User Guideの「CloudWatch ログでのログデータ保管期間の変更」を参照してください。

拡張モニタリングのコスト

拡張モニタリングのメトリクスは、CloudWatch メトリクスではなく CloudWatch Logs に保存されます。拡張モニタリングのコストは次の要因によって異なります。

- 拡張モニタリングの料金は、Amazon CloudWatch Logs に示された無料利用枠を超えた場合にのみ課金されます。CloudWatch Logs のデータ転送料金とストレージ料金に基づいて料金が決まります。
- RDS インスタンスに対して転送される情報の量は、拡張モニタリング機能に対して定義された詳細度に正比例します。モニタリング間隔を短くすると、OS メトリクスのレポート回数が増え、モニタリングコストが高くなります。コストを管理するには、アカウント内のインスタンスごとに異なる詳細度を設定します。
- 拡張モニタリングの使用コストは、拡張モニタリングが有効になっている各 DB インスタンスに適用されます。多数の DB インスタンスをモニタリングすると、少数の DB インスタンスをモニタリングするよりもコストが高くなります。

- 複数のコンピューティング集中型のワークロードをサポートする DB インスタンスでは、レポートする OS プロセスアクティビティが増え、拡張モニタリングのコストがより高くなります。

料金の詳細については、「Amazon CloudWatch の料金」を参照してください。

拡張モニタリングの設定と有効化

拡張モニタリングを使用するには、IAM ロールを作成し、拡張モニタリングを有効にする必要があります。

トピック

- 拡張モニタリング用の IAM ロールの作成
- 拡張モニタリングのオンとオフを切り替える
- 「混乱した代理」問題からの保護

拡張モニタリング用の IAM ロールの作成

拡張モニタリングには、CloudWatch Logs に OS メトリクスの情報を送るためのアクセス権限が必要です。AWS Identity and Access Management (IAM) ロールを使用して、Enhanced Monitoring に必要なアクセス許可を付与します。このロールは、拡張モニタリングを有効にする際に作成することも、事前に作成しておくこともできます。

トピック

- 拡張モニタリングを有効にしたときの IAM ロールの作成
- 拡張モニタリングを有効にする前の IAM ロールの作成

拡張モニタリングを有効にしたときの IAM ロールの作成

RDS コンソールで拡張モニタリングを有効にすると、Amazon RDS は必要な IAM ロールを作成できます。ロールの名前は `rds-monitoring-role` です。RDS は、指定済み DB インスタンス、リードレプリカ、またはマルチ AZ DB クラスターに対してこのロールを使用します。

拡張モニタリングを有効にするときに、IAM ロールを作成するには

1. 拡張モニタリングのオンとオフを切り替える の手順を行います。
2. ロールを選択する手順で、[モニタリングロール] を [デフォルト] に設定します。

拡張モニタリングを有効にする前の IAM ロールの作成

拡張モニタリングを有効にする前に、必要なロールを作成できます。拡張モニタリングを有効にする場合は、新しいロールの名前を指定します。AWS CLI または RDS API を使用して拡張モニタリングを有効にする場合は、この必要なロールを作成する必要があります。

拡張モニタリングを有効にするユーザーには、`PassRole` アクセス許可を付与する必要があります。詳細については、IAM ユーザーガイドの「AWS サービスにロールを渡すアクセス許可をユーザーに許可する」の「例 2」を参照してください。

Amazon RDS 拡張モニタリング用の IAM ロールを作成するには

1. IAM コンソール (https://console.aws.amazon.com) を開きます。
2. ナビゲーションペインで [ロール] を選択します。
3. [ロールの作成] を選択します。
4. [AWS のサービス] タブを選択し、サービスのリストから [RDS] を選択します。
5. [RDS - Enhanced Monitoring] (RDS - 拡張モニタリング)、[Next] (次へ) の順に選択します。
6. [Permissions policies] (アクセス許可ポリシー) に [AmazonRDSEnhancedMonitoringRole] が表示されていることを確認し、[Next] (次へ) を選択します。
7. [ロール名] に、ロールの名前を入力します。例えば、「**`emaccess`**」と入力します。

   ロールの信頼されたエンティティは、AWS サービスmonitoring.rds.amazonaws.com です。
8. [ロールの作成] を選択します。

拡張モニタリングのオンとオフを切り替える

拡張モニタリングのオンとオフは、AWS Management Console、AWS CLI、または RDS API を使用して切り替えることができます。拡張モニタリングをオンにする RDS インスタンスを選択します。DB インスタンスごとに、メトリクス収集の詳細度を別々に設定できます。

コンソール

DB クラスターもしくはリードレプリカの作成時、または DB インスタンスの変更時に、拡張モニタリングをオンにすることができます。拡張モニタリングをオンにするように DB インスタンスを変更した場合は、変更を有効にするために DB インスタンスを再起動する必要はありません。

RDS コンソールの [データベース] ページで、次のいずれかのアクションを行うと、拡張モニタリングをオンにすることができます。

- DB クラスターを作成する – [データベースを作成] を選択します。
- [リードレプリカの作成] – [アクション]、[リードレプリカの作成] の順にクリックします。
- DB インスタンスを変更する – [変更] を選択します。

RDS コンソールで拡張モニタリングのオンとオフを切り替えるには

1. [その他の設定] までスクロールします。
2. [モニタリング] で、DB インスタンスまたはリードレプリカに対し、[拡張モニタリングを有効にする] をクリックします。拡張モニタリングを無効にする場合は、[拡張モニタリングを無効にする] をクリックします。
3. [ロールの作成] プロパティで、Amazon CloudWatch Logs との通信を Amazon RDS に許可するために作成した IAM ロールに設定するか、[デフォルト] を選択して、RDS によって `rds-monitoring-role` という名前でロールが作成されるようにします。
4. [詳細度] プロパティを、DB インスタンスまたはリードレプリカのメトリクスが収集される間隔 (秒単位) に設定します。[詳細度] プロパティは、`1`、`5`、`10`、`15`、`30`、`60` のいずれかの値に設定できます。

   RDS コンソールは最速で 5 秒ごとに更新されます。RDS コンソールで詳細度を 1 秒に設定しても、メトリクスはやはり 5 秒ごとに更新されます。1 秒ごとにメトリクスの更新を取得するには、CloudWatch Logs を使用します。

AWS CLI

AWS CLI を使用して拡張モニタリングをオンにするには、次のコマンドで `--monitoring-interval` オプションを `0` 以外の値に設定し、`--monitoring-role-arn` オプションを 拡張モニタリング用の IAM ロールの作成 で作成したロールに設定します。

- create-db-instance
- create-db-instance-read-replica
- modify-db-instance

`--monitoring-interval` オプションにより、拡張モニタリングのメトリクスが収集される時間点の間隔 (秒単位) が指定されます。オプションの有効な値は、`0`、`1`、`5`、`10`、`15`、`30`、および `60` です。

AWS CLI を使用して拡張モニタリングをオフにするには、これらのコマンドで `--monitoring-interval` オプションを `0` に設定します。

Example

次の例では、DB インスタンスの拡張モニタリングをオンに切り替えています。

Linux、macOS、Unix の場合:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Windows の場合:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --monitoring-interval 30 ^
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Example

次の例では、マルチ AZ DB クラスターの拡張モニタリングをオンに切り替えています。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Windows の場合:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --monitoring-interval 30 ^
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

RDS API

RDS API を使用して拡張モニタリングをオンにするには、MonitoringInterval パラメータに 0 以外の値を設定し、MonitoringRoleArn パラメータを 拡張モニタリング用の IAM ロールの作成 で作成したロールに設定します。次のアクションでこれらのパラメータを設定します。

- CreateDBInstance
- CreateDBInstanceReadReplica
- ModifyDBInstance

MonitoringInterval パラメータにより、拡張モニタリングのメトリクスが収集される時間点の間隔 (秒単位) が指定されます。有効な値は、0、1、5、10、15、30、60 です。

RDS API を使用して拡張モニタリングをオフにするには、MonitoringInterval に 0 を設定します。

「混乱した代理」問題からの保護

混乱した代理問題は、アクションを実行する許可を持たないエンティティが、より特権のあるエンティティにアクションを実行するように強制できるセキュリティの問題です。AWS では、サービス間でのなりすましによって、混乱した代理問題が発生する場合があります。サービス間でのなりすましは、1 つのサービス (呼び出し元サービス) が、別のサービス (呼び出し対象サービス) を呼び出すときに発生する可能性があります。呼び出し元サービスは、本来ならアクセスすることが許可されるべきではない方法でその許可を使用して、別のお客様のリソースに対する処理を実行するように操作される場合があります。これを防ぐために、AWS には、アカウント内のリソースへのアクセス権が付与されたサービスプリンシパルですべてのサービスのデータを保護するために役立つツールが用意されています。詳細については、「「混乱した代理」問題」を参照してください。

リソースに関して、Amazon RDS から別のサービスに付与できるアクセス許可を制限する場合は、拡張モニタリングロール用の信頼ポリシー内で、aws:SourceArn および aws:SourceAccount のグローバル条件コンテキストキーを使用することをお勧めします。これら両方のグローバル条件コンテキストキーを使用する場合、アカウント ID は同じものを使用する必要があります。

混乱した代理問題から保護するための最も効果的な方法は、リソースの完全な ARN を指定して aws:SourceArn グローバル条件コンテキストキーを使用することです。Amazon RDS の場合は、aws:SourceArn に arn:aws:rds:*Region*:*my-account-id*:db:*dbname* を設定します。

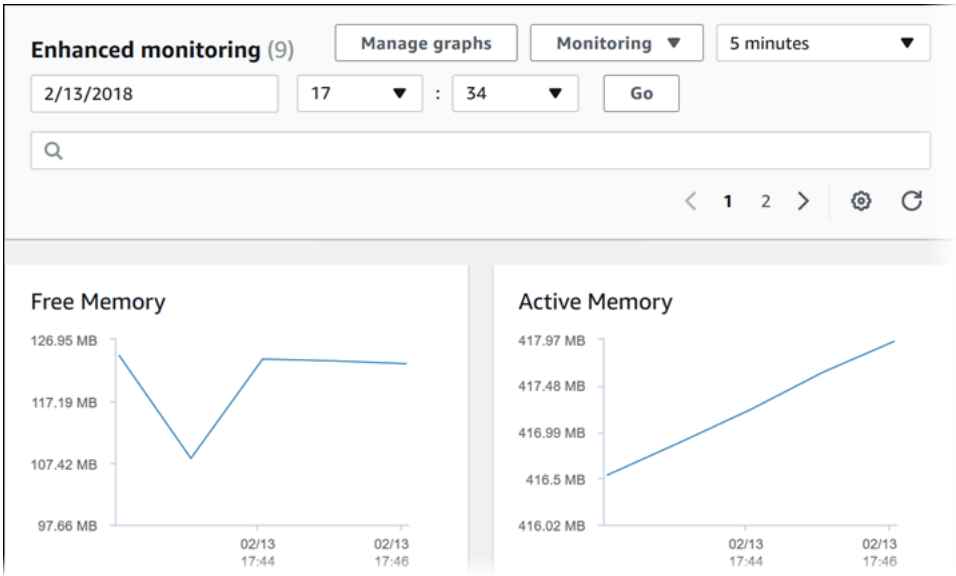
次の例では、「混乱した代理」問題を防ぐために、信頼ポリシー内で aws:SourceArn および aws:SourceAccount のグローバル条件コンテキストを使用しています。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "monitoring.rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringLike": {
          "aws:SourceArn": "arn:aws:rds:Region:my-account-id:db:dbname"
        },
        "StringEquals": {
          "aws:SourceAccount": "my-account-id"
        }
      }
    }
  ]
}
```

RDS コンソールでの OS メトリクスの表示

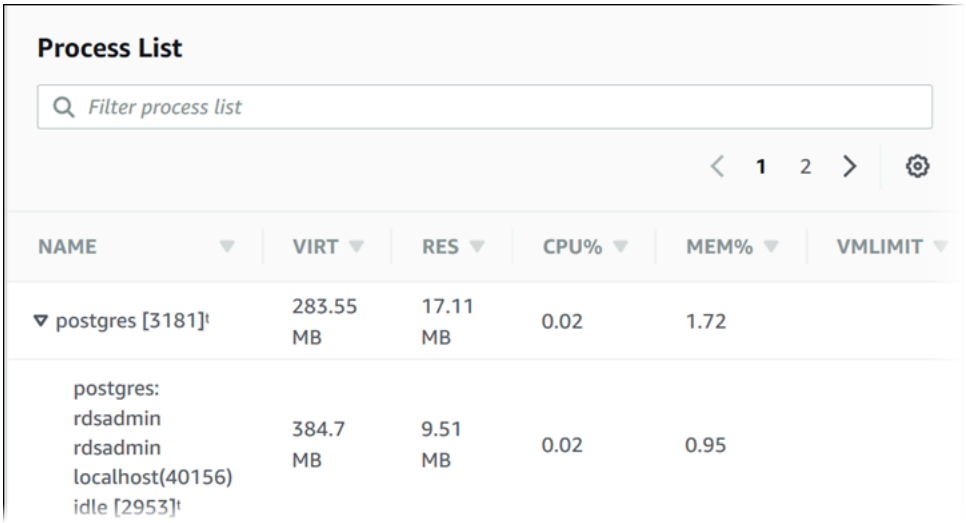
拡張モニタリングによってレポートされた OS のメトリクスを RDS コンソールで表示するには、[モニタリング] の [拡張モニタリング] を選択します。

次の例は、拡張モニタリングのページを示しています。拡張モニタリングメトリクスの説明については、「拡張モニタリングの OS メトリクス」を参照してください。



DB インスタンスで実行中のプロセスの詳細を確認する場合は、[モニタリング] の [OS プロセスリスト] を選択します。

[処理一覧] ビューは次のように表示されます。



[処理一覧] ビューに表示される拡張モニタリングのメトリクスは以下のように整理されます。

- [RDS 子プロセス] – DB インスタンスをサポートする RDS プロセス (Amazon Aurora DB クラスターの場合は `aurora`、) の概要を表示します。プロセスのスレッドは親プロセスの下にネストされて表示されます。プロセスのスレッドには CPU 使用率のみが表示されます。他のメトリクスはプロセスのすべてのスレッドで同じであるためです。コンソールには最大 100 個のプロセスとスレッドが表示されます。結果は、プロセスとスレッドを消費している上位の CPU とメモリの組み合わせです。プロセスとスレッドが 50 個よりも多い場合、コンソールではカテゴリ別に上位 50

個の消費元が表示されます。この表示は、パフォーマンスに最大の影響を与えているプロセスを特定するために役立ちます。

- [RDS プロセス] – RDS DB インスタンスをサポートするために必要な RDS 管理エージェント、診断モニタリングプロセス、その他の AWS プロセスによって使用されているリソースの概要を表示します。
- [OS processes] – 一般的にパフォーマンスに最小の影響を与えているカーネルとシステムプロセスの概要を表示します。

各プロセスに対して表示される項目は次のとおりです。

- VIRT – プロセスの仮想サイズを表示します。
- RES – プロセスが使用する実際の物理メモリを表示します。
- [CPU%] – プロセスで使用されている合計 CPU 帯域幅のパーセンテージを表示します。
- [MEM%] – プロセスで使用されている合計メモリのパーセンテージを表示します。

RDS コンソールに表示するモニタリングデータは、Amazon CloudWatch Logs から取得されます。また、DB インスタンスのメトリクスも CloudWatch Logs からログストリームとして取得できます。詳細については、「CloudWatch Logs を使用した OS メトリクスの表示」を参照してください。

以下の実行中は拡張モニタリングメトリクスは返されません:

- DB インスタンスのフェイルオーバー。
- DB インスタンスのインスタンスクラスの変更 (コンピューティングのスケール)。

拡張モニタリングのメトリクスは DB インスタンスの再起動中も返されます。これはデータベースエンジンのみが再起動するためです。オペレーティングシステムのメトリクスは、引き続き報告されます。

CloudWatch Logs を使用した OS メトリクスの表示

DB クラスターの拡張モニタリングを有効にした後、CloudWatch Logs を使用してそのメトリクスを表示できます。各ログストリームは、モニタリング中の 1 つの DB インスタンスまたは DB クラスターを表します。ログストリーム識別子は DB インスタンスまたは DB クラスターのリソース識別子 (`DbiResourceId`) です。

拡張モニタリングのログデータを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. 必要に応じて、DB クラスターが存在する AWS リージョン を選択します。詳細については、Amazon Web Services 全般のリファレンスの「リージョンとエンドポイント」を参照してください。
3. ナビゲーションペインで [ログ] を選択します。
4. ロググループのリストから [RDSOSMetrics] を選択します。
5. ログストリームのリストから、表示するログストリームを選択します。

Amazon Aurora のメトリクスリファレンス

このリファレンスでは、Amazon CloudWatch、Performance Insights、および Enhanced Monitoring に関する Amazon Aurora メトリクスが説明されています。

トピック

Amazon Aurora の Amazon CloudWatch メトリクス

AWS/RDS 名前空間には、Amazon Aurora で実行されるデータベースエンティティに適用される以下のメトリクスが含まれます。一部のメトリクスは、Aurora MySQL、Aurora PostgreSQL、またはその両方に適用されます。さらに、一部のメトリクスは、DB クラスター、プライマリ DB インスタンス、レプリカ DB インスタンス、またはすべての DB インスタンスに固有です。

Aurora Global Database メトリクスについては、「Aurora MySQL での書き込み転送の Amazon CloudWatch メトリクス」および「Aurora PostgreSQL での書き込み転送の Amazon CloudWatch メトリクス」を参照してください。Aurora パラレルクエリメトリクスについては、「パラレルクエリのモニタリング」を参照してください。

トピック

Amazon Aurora のクラスターレベルのメトリクス

次の表に、Aurora クラスター固有のメトリクスを示します。

Amazon Aurora でのクラスターレベルのメトリクス

メトリクス	説明	Applies to	単位
`AuroraGlobalDBDataTransferBytes`	Aurora Global Database で、マスター AWS リージョンからセカンダリ AWS リージョンに転送された REDO ログデータの量。 **Note** このメトリクスはセカンダリ AWS リージョン でのみ使用できます。	Aurora MySQL および Aurora PostgreSQL	バイト
`AuroraGlobalDBProgressLag`	Aurora Global Database で、ユーザートランザクションとシステムトランザクション両方の、プライマリクラスターの背後にあるセカンダリクラスターの距離を測定します。 **Note** このメトリクスはセカンダリ AWS リージョン でのみ使用できます。	Aurora MySQL および Aurora PostgreSQL	Milliseconds
`AuroraGlobalDBReplicatedWriteIO`	Aurora Global Database で、プライマリ AWS リージョンからセカンダリ AWS リージョンのクラスターボリュームに複製された書き	Aurora MySQL および Aurora PostgreSQL	カウント

メトリクス	説明	Applies to	単位
	込み I/O 操作の数。グローバルデータベース内のセカンダリ AWS リージョンの課金計算は、クラスター内で実行された書き込みを説明するために VolumeWriteIOPs が使用されます。グローバルデータベースのプライマリ AWS リージョンの課金計算では、そのクラスター内の書き込みアクティビティを示すために VolumeWriteIOPs を使用し、グローバルデータベース内のクロスリージョンレプリケーションを示すために AuroraGlobalDBReplicatedWriteIO を使用します。 **Note** このメトリクスはセカンダリ AWS リージョン でのみ使用できます。		

メトリクス	説明	Applies to	単位
`AuroraGlobalDBReplicationLag`	Aurora Global Database の場合、プライマリ AWS リージョンからアップデートをレプリケートする際の遅延時間。 **Note** このメトリクスはセカンダリ AWS リージョン でのみ使用できます。	Aurora MySQL および Aurora PostgreSQL	Milliseconds
`AuroraGlobalDBRPOLag`	Aurora Global Database での、目標復旧時点 (RPO) ラグタイム。このメトリクスにより、ユーザートランザクションのプライマリクラスターの背後にあるセカンダリクラスターの距離が測定されます。 **Note** このメトリクスはセカンダリ AWS リージョン でのみ使用できます。	Aurora MySQL および Aurora PostgreSQL	Milliseconds

メトリクス	説明	Applies to	単位
AuroraVolumeBytesLeftTotal	クラスターボリュームの残りの空き容量。クラスターボリュームが大きくなると、この値は減少します。ゼロに到達すると、クラスターはスペースがないというエラーを報告します。 Aurora MySQL クラスターが 128 tebibytes (TiB) のサイズ制限に近づいているかどうかを確認する場合、この値は VolumeBytesUsed よりもシンプルで信頼性があります。AuroraVolumeBytesLeftTotal は、ストレージ請求に影響しない内部のハウスキーピングやその他の割り当てに使用されているストレージも反映しています。	Aurora MySQL	バイト
BacktrackChangeRecordsCreationRate	DB クラスターで 5 分間に作成されたバックトラック変更レコードの数。	Aurora MySQL	5 分あたりのカウント
BacktrackChangeRecordsStored	DB クラスターで使用されたバックトラック変更レコードの数。	Aurora MySQL	カウント

メトリクス	説明	Applies to	単位
BackupRetentionPeriodStorageUsed	Aurora DB クラスターのバックアップ保存期間内で特定時点への復元機能をサポートするために使用されるバックアップストレージの合計容量。この金額は、TotalBackupStorageBilled メトリクスによって報告される合計に含まれます。Aurora クラスターごとに個別に計算されます。手順については、「Amazon Aurora バックアップストレージの使用状況を確認する」を参照してください。	Aurora MySQL および Aurora PostgreSQL	バイト
ServerlessDatabaseCapacity	Aurora Serverless DB クラスターの現在の容量。	Aurora MySQL および Aurora PostgreSQL	カウント

メトリクス	説明	Applies to	単位
SnapshotStorageUsed	バックアップ保存期間外で、Aurora DB クラスターのすべての Aurora スナップショットで消費されているバックアップストレージの合計容量。この金額は、TotalBackupStorageBilled メトリクスによって報告される合計に含まれます。Aurora クラスターごとに個別に計算されます。手順については、「Amazon Aurora バックアップストレージの使用状況を確認する」を参照してください。	Aurora MySQL および Aurora PostgreSQL	バイト

メトリクス	説明	Applies to	単位
TotalBackupStorageBilled	特定の Aurora DB クラスターに関して請求対象のバックアップストレージの合計容量 (バイト単位)。このメトリクスには、BackupRetentionPeriodStorageUsed メトリクスおよび SnapshotStorageUsed メトリクスによって測定されるバックアップストレージが含まれます。このメトリクスは、Aurora クラスターごとに個別に計算されます。手順については、「Amazon Aurora バックアップストレージの使用状況を確認する」を参照してください。	Aurora MySQL および Aurora PostgreSQL	バイト

メトリクス	説明	Applies to	単位
VolumeBytesUsed	Aurora DB クラスターで使用したストレージ容量。 この値は、Aurora DB クラスターのコストに影響します (料金の詳細については、Amazon RDS の料金設定ページを参照してください)。 この値は、ストレージ請求に影響しないいくつかの内部ストレージ割り当てを反映しません。Aurora MySQL の場合、AuroraVolumeBytesLeftTotal を 128 TiB のストレージ制限と比較する代わりに、VolumeBytesUsed がゼロに近づいているかどうかテストすることで、容量不足に関する問題をより正確に予想できます。 クラスターがクローンである場合、このメトリクスの値は、クローンに対して追加または変更したデータの量によって異なります。また、このメトリクスは元のクラスターの削除、新しいクローンの追加または削除に応じて増減します。詳細については、「ソースクラ	Aurora MySQL および Aurora PostgreSQL	バイト

メトリクス	説明	Applies to	単位
	スターボリュームの削除」を参照してください。		

メトリクス	説明	Applies to	単位
VolumeReadIOPs	5 分以内の、クラスターボリュームからの課金読み取り I/O オペレーションの回数。 課金読み取りオペレーションはクラスターボリュームレベルで計算され、Aurora DB クラスター内のすべてのインスタンスから集計された後、5 分おきに報告されます。この値は、5 分間にわたる読み取りオペレーションメトリクスの値を受け取ることによって計算されます。課金読み取りオペレーションメトリクスの値を受け取って 300 秒で割ることで、1 秒あたりの課金読み取りオペレーションの回数を決定できます。例えば、課金読み取りオペレーションが 13,686 を返す場合、1 秒あたりの課金読み取りオペレーションは 45 (13,686 / 300 = 45.62) です。 バッファキャッシュにないデータベースのページをリクエストするクエリの課金読み取りオペレーションが発生します。これはストレージからロードする必要があります。課金読み取り	Aurora MySQL および Aurora PostgreSQL	5 分あたりのカウント

メトリクス	説明	Applies to	単位
	オペレーションはストレージからクエリの結果が読み取られるのと同様に急増することがありますが、その後バッファキャッシュにロードされます。 **Tip** Aurora MySQL クラスターがパラレルクエリを使用している場合、VolumeReadIOPS 値が増加することがあります。パラレルクエリでは、バッファプールは使用されません。したがって、クエリは高速ですが、この最適化された処理により、読み取り操作とそれに関連する料金が増加する可能性があります。		

メトリクス	説明	Applies to	単位
VolumeWriteIOPs	クラスターボリュームに対する書き込みディスク I/O オペレーションの回数 (5 分間隔で報告されます)。課金書き込みオペレーションの計算方法の詳細については、「VolumeReadIOPs 」を参照してください。	Aurora MySQL および Aurora PostgreSQL	5 分あたりのカウント

Amazon Aurora のインスタンスレベルのメトリクス

以下のインスタンス固有の CloudWatch メトリクスは、特に断りのない限り、すべての Aurora MySQL および Aurora PostgreSQL インスタンスに適用されます。

Amazon Aurora でのインスタンスレベルのメトリクス

メトリクス	説明	Applies to	単位
AbortedClients	適切に閉じられなかったクライアント接続の数。	Aurora MySQL	カウント
ActiveTransactions	Aurora データベースインスタンスで実行されている現在のトランザクションの 1 秒あたりの平均数。 Aurora では、このメトリクスはデフォルトで有効になっていません。この値の計測をスタートするには、特定の DB インスタンス用の DB パラメータグループに innodb_monitor_ena	Aurora MySQL	1 秒あたりのカウント数

メトリクス	説明	Applies to	単位
	ble='all' を設定します。		
ACUUtilization	ServerlessDatabaseCapacity メトリクスの値を DB クラスターの最大 ACU 値で割った値。 このメトリクスは Aurora Serverless v2 にのみ適用されます。	Aurora MySQL および Aurora PostgreSQL	割合 (%)

メトリクス	説明	Applies to	単位
AuroraBinlogReplicaLag	バイナリログレプリケーションソースから Aurora MySQL 互換エディション で実行されるバイナリログレプリカ DB クラスターまでの遅延時間。遅延があることは、レプリカがレコードを適用できる速さよりソースがレコードを速く生成していることを意味します。 このメトリクスは、エンジンのバージョンに応じて異なる値を報告します。 Aurora MySQL バージョン 2 MySQL SHOW SLAVE STATUS の Seconds_Behind_Master フィールド Aurora MySQL バージョン 3 SHOW REPLICA STATUS このメトリクスを使用して、バイナリログレプリカとして機能するクラスターのエラーとレプリカの遅延を監視できます。メトリクス値は、次のことを示します。	Aurora MySQL のプライマリ	[秒]

メトリクス	説明	Applies to	単位
	高い値 レプリカがレプリケーションソースより遅れています。 0 または 0 に近い値 レプリカプロセスはアクティブで最新です。 -1 Aurora はレプリカのセットアップ中やレプリカがエラー状態にあるときに発生する遅延を確認できません。 バイナリログのレプリケーションはクラスターのライターインスタンスでのみ行われるため、WRITER ロールに関連付けられているこのメトリクスのバージョンを使用することをお勧めします。 レプリケーションの管理の詳細については、「AWS リージョン 間での Amazon Aurora MySQL DB クラスターのレプリケーション」を参照してください。トラブルシューティングの詳細については、「Amazon Aurora MySQL レプリケーションの問題」を参照してください。		

メトリクス	説明	Applies to	単位
AuroraEstimatedSharedMemoryBytes	最後に設定したポーリング間隔でアクティブに使用した共有バッファまたはバッファプールメモリの推定量。		バイト
AuroraOptimizedReadsCacheHitRatio	Optimized Reads キャッシュから提供されたリクエストの割合 (パーセント)。 値は、次の式を使用して計算されます。 orcache_blks_hit/(orcache_blks_hit + storage_blks_read) AuroraOptimizedReadsCacheHitRatio が 100% の場合、Optimized Reads キャッシュからページが読み取られなかったことを意味し、値は 0 になります。	Aurora PostgreSQL のプライマリ	割合 (%)
AuroraReplicaLag	Aurora レプリカについて、プライマリインスタンスからアップデートをレプリケートする際の遅延時間。	Aurora MySQL と Aurora PostgreSQL のレプリカ	Milliseconds
AuroraReplicaLagMaximum	DB クラスター内のプライマリインスタンスと、いずれかの Aurora インスタンス間の最大遅延量。	Aurora MySQL と Aurora PostgreSQL のプライマリ	Milliseconds

メトリクス	説明	Applies to	単位
AuroraReplicaLagMinimum	DB クラスター内のプライマリインスタンスと、いずれかの Aurora インスタンス間の最少遅延量。	Aurora MySQL と Aurora PostgreSQL のプライマリ	Milliseconds
AuroraSlowConnectionHandleCount	ハンドシェイクを開始するために 2 秒 以上待機した接続の数。 このメトリクスは、Aurora MySQL バージョン 3 のみに適用されます。	Aurora MySQL	カウント
AuroraSlowHandshakeCount	ハンドシェイクを終了するまで 50 ミリ以上かかった接続の数。 このメトリクスは、Aurora MySQL バージョン 3 のみに適用されます。	Aurora MySQL	カウント
BacktrackWindowActual	ターゲットバックトラックウィンドウと実際のバックトラックウィンドウの差異。	Aurora MySQL のプライマリ	分
BacktrackWindowAlert	指定された期間の、実際のバックトラックウィンドウがターゲットバックトラックウィンドウより小さかった回数。	Aurora MySQL のプライマリ	カウント
BlockedTransactions	1 秒あたりのブロックされたデータベース内のトランザクションの平均数。	Aurora MySQL	1 秒あたりのカウント数

メトリクス	説明	Applies to	単位
BufferCacheHitRatio	バッファキャッシュから提供されたリクエストの割合 (パーセント)。	Aurora MySQL および Aurora PostgreSQL	割合 (%)
CommitLatency	エンジンとストレージがコミット操作を完了するのにかかる平均時間。	Aurora MySQL および Aurora PostgreSQL	Milliseconds
CommitThroughput	1 秒あたりのコミット操作の平均回数。	Aurora MySQL および Aurora PostgreSQL	1 秒あたりのカウント数
ConnectionAttempts	成功したかどうかを問わず、インスタンスへの接続を試行した回数。	Aurora MySQL	カウント

メトリクス	説明	Applies to	単位
`CPUCreditBalance`	インスタンスが累積される CPU クレジットの数。5 分間隔で報告されます。このメトリクスは、どのくらいの期間にわたり、DB インスタンスが指定されたレートのベースラインパフォーマンスレベルを超えてバーストできるかを判断するために使用できます。 このメトリクスは、次のインスタンスクラスにのみ適用されます。 • Aurora MySQL: `db.t2.small` 、`db.t2.medium` 、`db.t3`、`db.t4g` • Aurora PostgreSQL: `db.t3` および `db.t4g` **Note** T DB インスタンスクラスを、開発サーバーおよびテストサーバー、または他の本稼働以外のサーバーにのみ使用することをお勧めします。T インスタンスクラスの詳細については、「DB インスタンスクラスタイプ」	Aurora MySQL および Aurora PostgreSQL	カウント

メトリクス	説明	Applies to	単位
	を参照してください。 起動クレジットは Amazon RDS でも Amazon EC2 と同じように機能します。詳細については、「Linux インスタンス向け Amazon Elastic Compute Cloud ユーザーガイド」の「起動クレジット」を参照してください。		

メトリクス	説明	Applies to	単位
`CPUCreditUsage`	指定された期間中に消費される CPU クレジットの数。5 分間隔で報告されます。このメトリクスは、DB インスタンスに割り当てられた仮想 CPU による命令処理に使用される、物理 CPU の時間を測定します。 このメトリクスは、次のインスタンスクラスにのみ適用されます。 • Aurora MySQL: `db.t2.small` 、`db.t2.medium` 、`db.t3`、`db.t4g` • Aurora PostgreSQL: `db.t3` および `db.t4g` **Note** T DB インスタンスクラスを、開発サーバーおよびテストサーバー、または他の本稼働以外のサーバーにのみ使用することをお勧めします。T インスタンスクラスの詳細については、「DB インスタンスクラスタイプ」	Aurora MySQL および Aurora PostgreSQL	カウント

メトリクス	説明	Applies to	単位
	を参照してください。		
CPUSurplusCreditBalance	CPUCreditBalance 値がゼロの場合に unlimited インスタンスによって消費された余剰クレジットの数。 CPUSurplusCreditBalance 値は獲得した CPU クレジットによって支払われます。余剰クレジットの数が、24 時間にインスタンスが獲得できるクレジットの最大数を超えている場合、最大数を超えて消費された余剰クレジットに対しては料金が発生します。 CPU クレジットメトリクスは、5 分間隔でのみ利用可能です。	Aurora MySQL および Aurora PostgreSQL	クレジット (vCPU 分)

メトリクス	説明	Applies to	単位
CPUSurplusCreditsCharged	獲得 CPU クレジットにより支払われないために追加料金が発生した、消費された余剰クレジットの数。 消費された余剰クレジットは、以下のいずれかの状況に当てはまると料金が発生します。 • 消費された余剰クレジットが、インスタンスが 24 時間に獲得できる最大クレジット数を超えている。最大数を越えて消費された余剰クレジットは、時間の最後に課金されます。 • インスタンスが停止または終了した。 • インスタンスは unlimited から standard に切り替わります。 CPU クレジットメトリクスは、5 分間隔でのみ利用可能です。	Aurora MySQL および Aurora PostgreSQL	クレジット (vCPU 分)
CPUUtilization	Aurora DB インスタンスによって使用される CPU のパーセント。	Aurora MySQL および Aurora PostgreSQL	割合 (%)

メトリクス	説明	Applies to	単位
DatabaseConnections	データベースインスタンスへのクライアントネットワーク接続の数。 データベースセッションの数は、メトリクスが示す値よりも多くなることがあります。これは、メトリクス値には以下が含まれないためです。 • ネットワークへの接続がなくなっているものの、データベースによるクリーンアップ行われていないセッション • データベースエンジンが、固有の目的のために作成したセッション • データベースエンジンの並列実行機能によって作成されたセッション • データベースエンジンのジョブスケジューラによって作成されたセッション • Amazon Aurora の接続	Aurora MySQL および Aurora PostgreSQL	カウント
DDLLatency	リクエスト (例、作成、変更、削除リクエストなど) の平均時間。	Aurora MySQL	Milliseconds
DDLThroughput	1 秒あたりの DDL リクエストの平均数。	Aurora MySQL	1 秒あたりのカウント数

メトリクス	説明	Applies to	単位
Deadlocks	1 秒あたりのデータベース内のデッドロックの平均回数。	Aurora MySQL および Aurora PostgreSQL	1 秒あたりのカウント数
DeleteLatency	削除操作の平均時間。	Aurora MySQL	Milliseconds
DeleteThroughput	1 秒あたりの DELETE クエリの平均回数。	Aurora MySQL	1 秒あたりのカウント数
DiskQueueDepth	未処理のディスクアクセス (読み取り/書き込みリクエスト) の数。	Aurora MySQL および Aurora PostgreSQL	カウント
DMLLatency	挿入、更新、削除の平均時間。	Aurora MySQL	Milliseconds
DMLThroughput	1 秒あたりの挿入、更新、削除の平均回数。	Aurora MySQL	1 秒あたりのカウント数
EngineUptime	インスタンス実行時間。	Aurora MySQL および Aurora PostgreSQL	[秒]
FreeableMemory	使用可能な RAM の容量。	Aurora MySQL および Aurora PostgreSQL	バイト
FreeEphemeralStorage	使用できる Ephemeral NVMe ストレージの容量。	Aurora PostgreSQL	バイト

メトリクス	説明	Applies to	単位
FreeLocalStorage	使用できるローカルストレージの量。 他の DB エンジンとは異なり、Aurora DB インスタンスの場合、このメトリクスでは各 DB インスタンスで使用できるストレージの量がレポートされます。この値は、DB インスタンスクラスによって異なります (料金の詳細については、Amazon RDS の料金設定ページを参照してください)。インスタンスに対してより大きな DB インスタンスクラスを選択することで、インスタンス用の空きストレージ容量を増やすことができます。 (これは Aurora Serverless v2 には適用されません。)	Aurora MySQL および Aurora PostgreSQL	バイト
InsertLatency	挿入操作の平均時間。	Aurora MySQL	Milliseconds
InsertThroughput	1 秒あたりの挿入操作の平均回数。	Aurora MySQL	1 秒あたりのカウント数
LoginFailures	1 秒あたりの失敗したログインの平均試行回数。	Aurora MySQL	1 秒あたりのカウント数

メトリクス	説明	Applies to	単位
MaximumUsedTransactionIDs	最も古い未バキュームトランザクション ID の古さ。この値が 2,146,483,648 (2^31 - 1,000,000) に達した場合、トランザクション ID の循環を防ぐためにデータベースは読み取り専用モードになります。詳細については、PostgreSQL ドキュメントの Preventing transaction ID wraparound failures を参照してください。	Aurora PostgreSQL	カウント
NetworkReceiveThroughput	Aurora DB クラスター内の各インスタンスが各クライアントから受信したネットワークスループットの量。Aurora DB クラスターとクラスターボリューム内のインスタンス間のネットワークトラフィックは、このスループットに含まれません。	Aurora MySQL および Aurora PostgreSQL	1 秒あたりのバイト数 (コンソールには 1 秒あたりのメガバイト数が表示されます)
NetworkThroughput	Aurora DB クラスター内の各インスタンスが各クライアントで送受信したネットワークスループットの量。Aurora DB クラスターとクラスターボリューム内のインスタンス間のネットワークトラフィックは、このスループットに含まれません。	Aurora MySQL および Aurora PostgreSQL	1 秒あたりのバイト数

メトリクス	説明	Applies to	単位
NetworkTransmitThroughput	Aurora DB クラスター内の各インスタンスが各クライアントに対して送信したネットワークスループットの量。DB クラスターとクラスターボリューム内のインスタンス間のネットワークトラフィックは、このスループットに含まれません。	Aurora MySQL および Aurora PostgreSQL	1 秒あたりのバイト数 (コンソールには 1 秒あたりのメガバイト数が表示されます)
NumBinaryLogFiles	生成されたバイナリログファイルの数。	Aurora MySQL	カウント
OldestReplicationSlotLag	受信した先行書き込み (WAL) データに関して最も遅延の長いレプリカの遅延サイズ。	Aurora PostgreSQL	バイト
PurgeBoundary	InnoDB パージ可能領域の最後のトランザクション番号。このメトリクスが長く進まない場合は、InnoDB パージが長時間実行中のトランザクションによってブロックされていることを示す良い目安となります。調査するには、Aurora MySQL DB クラスターのアクティブなトランザクション数を確認します。	Aurora MySQL バージョン 2、バージョン 2.11 以降	カウント
PurgeFinishedPoint	InnoDB パージを実行する領域の最後のトランザクション番号。このメトリクスは、InnoDB パージの進行速度を調べるのに役立ちます。	Aurora MySQL バージョン 2、バージョン 2.11 以降	カウント

メトリクス	説明	Applies to	単位
Queries	1 秒あたりに実行されたクエリの平均回数。	Aurora MySQL	1 秒あたりのカウント数
RDSToAuroraPostgreSQLReplicaLag	更新をプライマリ RDS PostgreSQL インスタンスからクラスター内の他のノードにレプリケートする際の遅延。	Aurora PostgreSQL のレプリカ	[秒]
ReadIOPS	1 秒あたりのディスク I/O オペレーションの平均回数。ただし 1 分間隔で読み込みおよび書き込みを個別に報告します。	Aurora MySQL および Aurora PostgreSQL	1 秒あたりのカウント数
ReadIOPSEphemeralStorage	Ephemeral NVMe ストレージへのディスク読み取り I/O オペレーションの平均数。	Aurora PostgreSQL	1 秒あたりのカウント数
ReadLatency	1 回のディスク I/O 操作にかかる平均時間。	Aurora MySQL および Aurora PostgreSQL	[秒]
ReadLatencyEphemeralStorage	Ephemeral NVMe ストレージのディスク読み取り I/O オペレーションごとにかかる平均時間。	Aurora PostgreSQL	Milliseconds
ReadThroughput	1 秒あたりのディスクからの平均読み取りバイト数。	Aurora MySQL および Aurora PostgreSQL	1 秒あたりのバイト数

メトリクス	説明	Applies to	単位
ReadThroughputEphemeralStorage	Ephemeral NVMe ストレージ用にディスクから読み取られた 1 秒あたりの平均バイト数。	Aurora PostgreSQL	1 秒あたりのバイト数
ReplicationSlotDiskUsage	レプリケーションスロットファイルで消費するディスク領域の量。	Aurora PostgreSQL	バイト
ResultSetCacheHitRatio	Resultset キャッシュから提供されたリクエストの割合 (パーセント)。	Aurora MySQL	割合 (%)
RollbackSegmentHistoryListLength	コミットされたトランザクションが削除とマークされたレコードを記録する UNDO ログ。これらのレコードは、InnoDB のパージオペレーションによって処理されるようにスケジュールされています。	Aurora MySQL	カウント
RowLockTime	InnoDB テーブルの行ロックの取得にかかった合計時間。	Aurora MySQL	Milliseconds
SelectLatency	選択した操作の平均時間。	Aurora MySQL	Milliseconds
SelectThroughput	1 秒あたりの選択クエリの平均回数。	Aurora MySQL	1 秒あたりのカウント数
ServerlessDatabaseCapacity	Aurora Serverless DB クラスターの現在の容量。	Aurora MySQL および Aurora PostgreSQL	カウント

メトリクス	説明	Applies to	単位
StorageNetworkReceiveThroughput	DB クラスター内の各インスタンスが、Aurora のストレージサブシステムから受信した、ネットワークスループットの量。	Aurora MySQL および Aurora PostgreSQL	1 秒あたりのバイト数
StorageNetworkThroughput	Aurora DB クラスター内の各インスタンスが、Aurora のストレージサブシステムとの間で送受信した、ネットワークスループットの量。	Aurora MySQL および Aurora PostgreSQL	1 秒あたりのバイト数
StorageNetworkTransmitThroughput	Aurora DB クラスター内の各インスタンスが、Aurora のストレージサブシステムに送信した、ネットワークスループットの量。	Aurora MySQL および Aurora PostgreSQL	1 秒あたりのバイト数
SumBinaryLogSize	バイナリログファイルの合計サイズ。	Aurora MySQL	バイト
SwapUsage	使用したスワップ領域の量。このメトリクスは、以下の DB インスタンスクラスでは使用できません。 • db.r3.*、db.r4.*、db.r7g.* (Aurora MySQL) • db.r7g.* (Aurora PostgreSQL)	Aurora MySQL および Aurora PostgreSQL	バイト

メトリクス	説明	Applies to	単位
TempStorageIOPS	DB インスタンスにアタッチされたローカルストレージの読み取りと書き込み両方で実行された IOPS の数。このメトリクスはカウントを表し、1 秒に 1 回測定されます。 このメトリクスは Aurora Serverless v2 にのみ適用されます。	Aurora MySQL および Aurora PostgreSQL	1 秒あたりのカウント数
TempStorageThroughput	DB インスタンスに関連するローカルストレージとの間で転送されるデータの量です。このメトリクスはバイトを表し、1 秒に 1 回測定されます。 このメトリクスは Aurora Serverless v2 にのみ適用されます。	Aurora MySQL および Aurora PostgreSQL	1 秒あたりのバイト数

メトリクス	説明	Applies to	単位
TransactionLogsDiskUsage	Aurora PostgreSQL DB インスタンスでトランザクションログが消費するディスク領域の量。 このメトリクスは、Aurora PostgreSQL が、論理的なレプリケーションまたは AWS Database Migration Service を使用している場合にのみ生成されます。デフォルトでは、Aurora PostgreSQL はトランザクションログではなくログレコードを使用します。トランザクションログが使用されていない場合、このメトリクスの値は -1 です。	Aurora PostgreSQL のプライマリ	バイト
TruncateFinishedPoint	切り捨てを元に戻す操作を実行する最後のトランザクション識別子。	Aurora MySQL バージョン 2、バージョン 2.11 以降	カウント
UpdateLatency	更新オペレーションにかかった平均時間。	Aurora MySQL	Milliseconds
UpdateThroughput	1 秒あたりの平均更新数。	Aurora MySQL	1 秒あたりのカウント数

メトリクス	説明	Applies to	単位
WriteIOPS	1 秒あたりに生成された Aurora ストレージ書き込みレコードの数。これは、データベースによって生成されるログレコードの概数です。これらは 8K ページの書き込みや、送信されるネットワークパケットと一致しません。	Aurora MySQL および Aurora PostgreSQL	1 秒あたりのカウント数
WriteIOPSEphemeralStorage	Ephemeral NVMe ストレージへのディスク書き込み I/O オペレーションの平均数。	Aurora PostgreSQL	1 秒あたりのカウント数
WriteLatency	1 回のディスク I/O 操作にかかる平均時間。	Aurora MySQL および Aurora PostgreSQL	[秒]
WriteLatencyEphemeralStorage	Ephemeral NVMe ストレージのディスク書き込み I/O オペレーションごとにかかる平均時間。	Aurora PostgreSQL	Milliseconds
WriteThroughput	永続的ストレージに 1 秒ごとに書き込まれた平均バイト数。	Aurora MySQL および Aurora PostgreSQL	1 秒あたりのバイト数
WriteThroughputEphemeralStorage	Ephemeral NVMe ストレージ用にディスクに書き込まれる 1 秒あたりの平均バイト数。	Aurora PostgreSQL	1 秒あたりのバイト数

Amazon Aurora の Amazon CloudWatch 使用状況メトリクス

Amazon CloudWatch の AWS/Usage 名前空間には、Amazon RDS サービスクォータのアカウントレベルの使用状況メトリクスが含まれています。CloudWatch では、すべての AWS リージョン の使用状況メトリクスを自動的に収集します。

詳細については、Amazon CloudWatch ユーザーガイドの「Amazon CloudWatch 使用状況メトリクスの使用」を参照してください。クォータの詳細については、Service Quotas ユーザーガイドの Amazon Aurora のクォータと制約 および クォータの引き上げのリクエストを参照してください。

メトリクス	説明	単位*
DBClusterParameterGroups	AWS アカウント の DB クラスターパラメータグループの数。カウントでは、デフォルトのパラメータグループは除外されます。	カウント
DBClusters	AWS アカウント の Amazon Aurora DB クラスターの数。	カウント
DBInstances	AWS アカウント の DB インスタンスの数。	カウント
DBParameterGroups	AWS アカウント の DB パラメータグループの数。カウントでは、デフォルトの DB パラメータグループは除外されます。	カウント
DBSubnetGroups	AWS アカウント の DB サブネットグループの数。カウントでは、デフォルトのサブネットグループは除外されます。	カウント
ManualClusterSnapshots	AWS アカウント にある、手動で作成された DB クラスタースナップショットの数。このカウントでは、無効なスナップショットは除外されます。	カウント
OptionGroups	AWS アカウント 内のオプショングループの数。カウントでは、デフォルトのオプショングループは除外されます。	カウント
ReservedDBInstances	AWS アカウント の予約済み DB インスタンスの数。カウントでは、使用停止または拒否されたインスタンスは除外されます。	カウント

> **Note**
>
> Amazon RDS は、使用状況メトリクスのユニットを CloudWatch に発行しません。ユニットはドキュメントにのみ表示されます。

Aurora の Amazon CloudWatch ディメンション

次の表に示す任意のディメンションを使用して、 メトリックスデータをフィルタリングができます。

ディメンション	以下で要求されたデータをフィルタリングします。
`DBInstanceIdentifier`	特定の DB インスタンス。
`DBClusterIdentifier`	特定の Aurora DB クラスター。
`DBClusterIdentifier, Role`	インスタンスロール (WRITER/READER) ごとにメトリクスを集約する特定の Aurora DB クラスター。例えば、クラスターに属するすべての READER インスタンスのメトリクスを集計できます。
`DbClusterIdentifier, EngineName`	特定の Aurora DB クラスターとエンジン名の組み合わせ。例えば、クラスター `ams1`およびエンジン `aurora` の `VolumeReadIOPs` メトリクスを表示できます。
`DatabaseClass`	データベースクラスのすべてのインスタンス。例えば、`db.r5.large` というデータベースクラスに属するすべてのインスタンスのメトリクスを集計できます。
`EngineName`	特定されたエンジン名のみ。例えば、メトリクスを組み合わせて、`aurora-postgresql` というエンジン名を有する全インスタンスを抽出することができます。
`SourceRegion`	指定されたリージョンのみ。例えば、`us-east-1` リージョンのすべての DB インスタンスのメトリクスを集計できます。

Amazon RDS コンソールでの Aurora メトリクスの使用可否

Amazon Aurora で提供されるすべてのメトリクスを Amazon RDS コンソールで使用できるわけではありません。ただし、これらのメトリクスを、AWS CLI や CloudWatch API などのツールを使用して確認することはできます。また、Amazon RDS コンソールでのメトリクスの中には、特定のインスタンスクラスでのみ表示される場合や、異なる名前や測定単位で表示される場合があります。

トピック

[Last Hour] (過去 1 時間) ビューで利用可能な Aurora メトリクス

分類された Aurora メトリクスの一部が、Amazon RDS コンソールのデフォルトの [Last Hour] (過去 1 時間) ビューに表示されます。次の表は、Aurora インスタンスの Amazon RDS コンソールに表示されるカテゴリと、関連するメトリクスの一覧です。

カテゴリ	メトリクス
SQL	`ActiveTransactions`
	`BlockedTransactions`
	`BufferCacheHitRatio`
	`CommitLatency`
	`CommitThroughput`
	`DatabaseConnections`
	`DDLLatency`
	`DDLThroughput`
	`Deadlocks`
	`DMLLatency`

カテゴリ	メトリクス
	DMLThroughput LoginFailures ResultSetCacheHitRatio SelectLatency SelectThroughput
システム	AuroraReplicaLag AuroraReplicaLagMaximum AuroraReplicaLagMinimum CPUCreditBalance CPUCreditUsage CPUUtilization FreeableMemory FreeLocalStorage （これは Aurora Serverless v2 には適用されません。） NetworkReceiveThroughput
デプロイメント	AuroraReplicaLag BufferCacheHitRatio ResultSetCacheHitRatio SelectThroughput

特定のケースで利用可能な Aurora メトリクス

また、Aurora メトリクスの中には、特定のインスタンスクラス、または DB インスタンスでのみ表示されるか、異なる名前や測定単位が含まれる場合があります。

- `CPUCreditBalance` と `CPUCreditUsage` の各メトリクスは、Aurora MySQL `db.t2` インスタンスクラスと Aurora PostgreSQL `db.t3` インスタンスクラスの場合にのみ表示されます。
- 次のメトリクスでは、以下のように、表示される名前が異なります。

メトリクス	表示名
`AuroraReplicaLagMaximum`	最大レプリカラグ
`AuroraReplicaLagMinimum`	最小レプリカラグ
`DDLThroughput`	DDL
`NetworkReceiveThroughput`	ネットワークスループット
`VolumeBytesUsed`	[請求済み] 使用されたボリュームのバイト数
`VolumeReadIOPs`	[Billed] (請求済み) ボリューム読み取り IOPS
`VolumeWriteIOPs`	[Billed] (請求済み) ボリューム読み取り IOPS

- 次のメトリクスは Aurora DB クラスター全体に適用されますが、Amazon RDS コンソールで Aurora DB クラスターの DB インスタンスを表示する場合にのみ表示されます。
  - `VolumeBytesUsed`
  - `VolumeReadIOPs`
  - `VolumeWriteIOPs`
- 以下のメトリクスは、Amazon RDS コンソールで、バイト単位ではなくメガバイト単位で表示されます。
  - `FreeableMemory`
  - `FreeLocalStorage`
  - `NetworkReceiveThroughput`
  - `NetworkTransmitThroughput`

- Aurora Optimized Reads を使用する Aurora PostgreSQL DB クラスターには、次のメトリクスが適用されます。
  - AuroraOptimizedReadsCacheHitRatio
  - FreeEphemeralStorage
  - ReadIOPSEphemeralStorage
  - ReadLatencyEphemeralStorage
  - ReadThroughputEphemeralStorage
  - WriteIOPSEphemeralStorage
  - WriteLatencyEphemeralStorage
  - WriteThroughputEphemeralStorage

コンソールで使用できない Aurora メトリクス

以下の Aurora メトリクスは、Amazon RDS コンソールで使用できません。

- AuroraBinlogReplicaLag
- DeleteLatency
- DeleteThroughput
- EngineUptime
- InsertLatency
- InsertThroughput
- NetworkThroughput
- Queries
- UpdateLatency
- UpdateThroughput

Performance Insights の Amazon CloudWatch メトリクス

Performance Insights はメトリクスを自動的に Amazon CloudWatch に発行します。Performance Insights から同じデータに対してクエリを実行できますが、CloudWatch にメトリクスを含めると、CloudWatch アラームを追加しやすくなります。また、既存の CloudWatch ダッシュボードにメトリクスを追加しやすくなります。

メトリクス	説明
DBLoad	DB エンジンのアクティブセッション数。通常、アクティブセッションの平均数に関するデータを使用します。Performance Insights で、このデータは `db.load.avg` としてクエリされます。
DBLoadCPU	待機イベントタイプが CPU であるアクティブセッションの数。Performance Insights で、このデータは、待機イベントタイプ `db.load.avg` でフィルタ処理された CPU としてクエリされます。
DBLoadNonCPU	待機イベントタイプが CPU でないアクティブセッションの数。

> **Note**
>
> これらのメトリクスは、DB インスタンスに負荷がある場合にのみ CloudWatch に公開されます。

これらのメトリクスは、CloudWatch コンソール、AWS CLI、または CloudWatch API を使用して調査できます。特別な Metric Math 関数を使用して、他の Performance Insights カウンターメトリクスを調べることもできます。詳細については、「CloudWatch での他の Performance Insights カウンターメトリクスのクエリ」を参照してください。

例えば、DBLoad メトリクスの統計情報は、get-metric-statistics コマンドを実行して取得できます。

```
aws cloudwatch get-metric-statistics \
    --region us-west-2 \
    --namespace AWS/RDS \
    --metric-name DBLoad  \
    --period 60 \
    --statistics Average \
    --start-time 1532035185 \
    --end-time 1532036185 \
```

```
    --dimensions Name=DBInstanceIdentifier,Value=db-loadtest-0
```

次のコマンドでは、以下のような出力が生成されます。

```
{
  "Datapoints": [
  {
  "Timestamp": "2021-07-19T21:30:00Z",
  "Unit": "None",
  "Average": 2.1
  },
  {
  "Timestamp": "2021-07-19T21:34:00Z",
  "Unit": "None",
  "Average": 1.7
  },
  {
  "Timestamp": "2021-07-19T21:35:00Z",
  "Unit": "None",
  "Average": 2.8
  },
  {
  "Timestamp": "2021-07-19T21:31:00Z",
  "Unit": "None",
  "Average": 1.5
  },
  {
  "Timestamp": "2021-07-19T21:32:00Z",
  "Unit": "None",
  "Average": 1.8
  },
  {
  "Timestamp": "2021-07-19T21:29:00Z",
  "Unit": "None",
  "Average": 3.0
  },
  {
  "Timestamp": "2021-07-19T21:33:00Z",
  "Unit": "None",
  "Average": 2.4
  }
  ],
  "Label": "DBLoad"
```

```
}
```

CloudWatch の詳細については、Amazon CloudWatch ユーザーガイドの「Amazon CloudWatch とは」を参照してください。

CloudWatch での他の Performance Insights カウンターメトリクスのクエリ

CloudWatch から RDS Performance Insights メトリクスのクエリ、アラーム、グラフを実行できます。CloudWatch の DB_PERF_INSIGHTS Metric Math 関数を使用して、DB クラスターに関する情報にアクセスできます。この関数を使用すると、CloudWatch に直接レポートされない Performance Insights メトリクスを使用して、新しい時系列を作成できます。

新しい Metric Math 関数を使用するには、CloudWatch コンソールの [メトリクスの選択] 画面の [数式を追加] ドロップダウンメニューをクリックします。これを使用して、Performance Insights メトリクス、または CloudWatch と Performance Insights メトリクス (1 分未満のメトリクスの高解像度アラームなど) の組み合わせに関するアラームとグラフを作成できます。`get-metric-data` リクエストに Metric Math 式を含めることで、プログラムでこの関数を使用することもできます。詳細については、「Metric Math 構文と関数」および「AWS データベースから Performance Insights カウンターメトリクスのアラームを作成する」を参照してください。

Performance Insights カウンターメトリクス

カウンターメトリクスは、Performance Insights ダッシュボードのオペレーティングシステムとデータベースのパフォーマンスメトリクスのことです。カウンターメトリクスを DB ロードと関連付けることで、パフォーマンスの問題を特定して分析できます。統計関数をメトリクスに追加して、メトリクス値を取得できます。例えば、`os.memory.active` メトリクスでサポートされている関数は、`.avg`、`.min`、`.max`、`.sum`、および `.sample_count` です。

カウンターメトリクスは 1 分に 1 回収集されます。OS メトリクスの収集は、拡張モニタリングがオンかオフかによって異なります。拡張モニタリングがオフになっている場合、OS メトリックは 1 分に 1 回収集されます。拡張モニタリングがオンになっている場合、選択した期間の OS メトリックが収集されます。拡張モニタリングのオンまたはオフの詳細については、拡張モニタリングのオンとオフを切り替える を参照してください。

トピック

- Performance Insights オペレーティングシステムのカウンター
- Aurora MySQL の Performance Insights のカウンター
- Aurora PostgreSQL の Performance Insights カウンター

Performance Insights オペレーティングシステムのカウンター

次のオペレーティングシステムカウンターは、os のプレフィックスが付き、Aurora PostgreSQL および Aurora MySQL では、Performance Insights で使用できます。

DB インスタンスで使用可能なカウンターメトリクスのリストについて、`ListAvailableResourceMetrics` API を使用できます。詳細については、「Amazon RDS Performance Insights API リファレンスガイド」の「ListAvailableResourceMetics」を参照してください。

Counter	タイプ	メトリクス	説明
[アクティブ]	「メモリ」	os.memory.active	割り当てられたメモリの量 (キロバイト単位)。
バッファ	「メモリ」	os.memory.buffers	ストレージデバイスへの書き込み前に I/O バッファリングリクエストに使用されたメモリの量 (キロバイト単位)。
キャッシュ済み	「メモリ」	os.memory.cached	ファイルシステムベースの I/O のキャッシュに使用されたメモリの量 (キロバイト単位)。
DB キャッシュ	「メモリ」	os.memory.db.cache	tmpfs (shmem) を含めて、データベースプロセスがページキャッシュに使用したメモリの量 (バイト単位)。
DB レジデントセットサイズ	「メモリ」	os.memory.db.residentSetSize	tmpfs (shmem) を含めずに、データベースプロセスが匿名

Counter	タイプ	メトリクス	説明
			キャッシュとスワップキャッシュに使用したメモリの量 (バイト単位)。
DB スワップ	「メモリ」	os.memory.db.swap	データベースプロセスがスワップに使用したメモリの量 (バイト単位)。
ダーティ	「メモリ」	os.memory.dirty	変更されたがストレージ内のその関連データブロックに書き込まれなかった RAM 内のメモリページの量 (キロバイト単位)。
空き	「メモリ」	os.memory.free	未割り当てのメモリの量 (キロバイト単位)。
huge ページ (空き)	「メモリ」	os.memory.hugePagesFree	空き huge ページの数。huge ページは Linux カーネルの機能です。
huge ページ (予約)	「メモリ」	os.memory.hugePagesRsvd	コミットされた huge ページの数。
huge ページサイズ	「メモリ」	os.memory.hugePagesSize	各 huge ページユニットのサイズ (キロバイト単位)。
huge ページ (余剰)	「メモリ」	os.memory.hugePagesSurp	使用可能な huge ページの余剰数/合計数。

Counter	タイプ	メトリクス	説明
huge ページ (合計)	「メモリ」	os.memory.hugePagesTotal	huge ページの合計数。
無効	「メモリ」	os.memory.inactive	最も使用されていないメモリページの量 (キロバイト単位)。
マップ済み	「メモリ」	os.memory.mapped	プロセスアドレス空間内でメモリマップされているファイルシステムの内容の合計量 (キロバイト単位)。
メモリ不足キルカウント	「メモリ」	os.memory.outOfMemoryKillCount	前回の収集間隔で発生した OOM キルの数。
ページテーブル	「メモリ」	os.memory.pageTables	ページテーブルが使用中のメモリの量 (キロバイト単位)。
スラブ	「メモリ」	os.memory.slab	再利用可能なカーネルデータ構造体の量 (キロバイト単位)。
合計	「メモリ」	os.memory.total	メモリの合計量 (キロバイト単位)。
書き戻し	「メモリ」	os.memory.writeback	バックアップストレージにまだ書き込み中の RAM 内のダーティページの量 (キロバイト単位)。

Counter	タイプ	メトリクス	説明
ゲスト	CPU 使用率	os.cpuUtilization.guest	ゲストプログラムが使用中の CPU の使用率。
アイドル状態	CPU 使用率	os.cpuUtilization.idle	アイドル状態の CPU の使用率。
irq	CPU 使用率	os.cpuUtilization.irq	ソフトウェア割り込みが使用中の CPU の使用率。
Nice	CPU 使用率	os.cpuUtilization.nice	最も低い優先順位で実行されているプログラムが使用中の CPU の使用率。
Steal	CPU 使用率	os.cpuUtilization.steal	他の仮想マシンが使用中の CPU の使用率。
システム	CPU 使用率	os.cpuUtilization.system	カーネルが使用中の CPU の使用率。
合計	CPU 使用率	os.cpuUtilization.total	使用中の CPU の合計使用率。この値は nice 値を含みます。
ユーザー	CPU 使用率	os.cpuUtilization.user	ユーザープログラムが使用中の CPU の使用率。
待機	CPU 使用率	os.cpuUtilization.wait	I/O アクセスを待機中の CPU の未使用率。
Aurora ストレージ Aurora ストレージ受信バイト数	ディスク IO	os.diskIO.auroraStorage.auroraStorageBytesRx	Aurora ストレージの 1 秒あたりの受信バイト数。

Counter	タイプ	メトリクス	説明
Aurora ストレージ Aurora ストレージ送信バイト数	ディスク IO	os.diskIO.auroraStorage.auroraStorageBytesTx	Aurora ストレージの 1 秒あたりのアップロードバイト数。
Aurora ストレージディスクキューの深さ	ディスク IO	os.diskIO.auroraStorage.diskQueueDepth	Aurora ストレージディスクキューの長さ。
Aurora ストレージ読み取り IO PS	ディスク IO	os.diskIO.auroraStorage.readIOsPS	読み取りオペレーションの 1 秒あたりの数。
Aurora ストレージ読み取りレイテンシー	ディスク IO	os.diskIO.auroraStorage.readLatency	Aurora ストレージへの読み取り I/O リクエストの平均レイテンシーをミリ秒単位で表します。
Aurora ストレージ読み取りスループット	ディスク IO	os.diskIO.auroraStorage.readThroughput	DB クラスターへのリクエストによって使用されるネットワークスループットの量 (バイト/秒単位)。
Aurora ストレージ書き込み IO PS	ディスク IO	os.diskIO.auroraStorage.writeIOsPS	書き込みオペレーションの 1 秒あたりの数。
Aurora ストレージ書き込みレイテンシー	ディスク IO	os.diskIO.auroraStorage.writeLatency	Aurora ストレージへの書き込み I/O リクエストの平均レイテンシーをミリ秒単位で表します。

Counter	タイプ	メトリクス	説明
Aurora ストレージ書き込みスループット	ディスク IO	os.diskIO.auroraStorage.writeThroughput	DB クラスターからのレスポンスによって使用されるネットワークスループットの量 (バイト/秒単位)。
Rdstemp 平均キュー長さ	ディスク IO	os.diskIO.rdstemp.avgQueueLen	I/O デバイスのキューで待機中のリクエストの数。
Rdstemp 平均リクエストサイズ	ディスク IO	os.diskIO.rdstemp.avgReqSz	I/O デバイスのキューで待機中のリクエストの数。
Rdstemp 待機中	ディスク IO	os.diskIO.rdstemp.await	リクエストへの応答に必要なミリ秒数 (キュー時間とサービス時間を含む)。
Rdstemp 読み取り IOPS	ディスク IO	os.diskIO.rdstemp.readIOsPS	読み取りオペレーションの 1 秒あたりの数。
Rdstemp 読み取り KB	ディスク IO	os.diskIO.rdstemp.readKb	読み取りの合計キロバイト数。
Rdstemp 読み取り KBPS	ディスク IO	os.diskIO.rdstemp.readKbPS	読み取りの 1 秒あたりのキロバイト数。
Rdstemp Rrqm PS	ディスク IO	os.diskIO.rdstemp.rrqmPS	キューに入れられてマージされた読み取りリクエストの 1 秒あたりの数。
Rdstemp TPS	ディスク IO	os.diskIO.rdstemp.tps	I/O トランザクションの 1 秒あたりの数。

Counter	タイプ	メトリクス	説明
Rdstemp 使用率	ディスク IO	os.diskIO.rdstemp.util	リクエスト発行中の CPU 時間の消費率。
Rdstemp 書き込み IOPS	ディスク IO	os.diskIO.rdstemp.writeIOsPS	書き込みオペレーションの 1 秒あたりの数。
Rdstemp 書き込み KB	ディスク IO	os.diskIO.rdstemp.writeKb	書き込みの合計キロバイト数。
Rdstemp 書き込み KBPS	ディスク IO	os.diskIO.rdstemp.writeKbPS	書き込みの 1 秒あたりのキロバイト数。
Rdstemp Wrqm PS	ディスク IO	os.diskIO.rdstemp.wrqmPS	キューに入れられてマージされた書き込みリクエストの 1 秒あたりの数。
ブロック	タスク	os.tasks.blocked	ブロックされているタスクの数。
実行中	タスク	os.tasks.running	実行中のタスクの数。
Sleeping	タスク	os.tasks.sleeping	スリープ中のタスクの数。
停止	タスク	os.tasks.stopped	停止中のタスクの数。
合計	タスク	os.tasks.total	タスクの合計数。
ゾンビ	タスク	os.tasks.zombie	アクティブな親タスクの非アクティブな子タスクの数。

Counter	タイプ	メトリクス	説明
1	負荷平均分	os.loadAverageMinute.one	過去 1 分間に CPU 時間をリクエストしたプロセスの数。
15	負荷平均分	os.loadAverageMinute.fifteen	過去 15 分間に CPU 時間をリクエストしたプロセスの数。
Five	負荷平均分	os.loadAverageMinute.five	過去 5 分間に CPU 時間をリクエストしたプロセスの数。
キャッシュ済み	スワップ	os.swap.cached	キャッシュメモリとして使用されたスワップメモリの量 (キロバイト単位)。
空き	スワップ	os.swap.free	空きスワップメモリの量 (キロバイト単位)。
In	スワップ	os.swap.in	ディスクからスワップされたメモリの量 (キロバイト単位)。
Out	スワップ	os.swap.out	ディスクにスワップされたメモリの量 (キロバイト単位)。
合計	スワップ	os.swap.total	使用可能なスワップメモリの合計量 (キロバイト単位)。
最大ファイル数	ファイルシステム	os.fileSys.maxFiles	ファイルシステム用に作成できるファイルの最大数。

Counter	タイプ	メトリクス	説明
使用済みファイル	ファイルシステム	os.fileSys.usedFiles	ファイルシステム内のファイルの数。
使用済みファイルパーセント	ファイルシステム	os.fileSys.usedFilePercent	使用中のファイルの割合。
使用率	ファイルシステム	os.fileSys.usedPercent	ファイルシステムが使用中のディスク領域の割合。
使用済み	ファイルシステム	os.fileSys.used	ファイルシステム内のファイルが使用中のディスク領域の量 (キロバイト単位)。
合計	ファイルシステム	os.fileSys.total	ファイルシステムに使用できるディスク領域の合計量 (キロバイト単位)。
受信	ネットワーク	os.network.rx	1 秒あたりの受信バイト数。
送信	ネットワーク	os.network.tx	1 秒あたりのアップロードバイト数。
ACU 使用率	全般	os.general.acuUtilization	設定された最大容量のうち、現在の容量の割合。
最大構成 ACU	全般	os.general.maxConfiguredAcu	ユーザーが設定した最大容量 (ACU 数) 。
最小構成 ACU	全般	os.general.minConfiguredAcu	ユーザーが設定した最小容量 (ACU 数) 。

Counter	タイプ	メトリクス	説明
Num VCPU	全般	os.general.numVCPUs	DB インスタンスの仮想 CPU の数。
サーバーレスデータベース容量	全般	os.general.serverlessDatabaseCapacity	ACU 内の DB インスタンスの現在の容量。

Aurora MySQL の Performance Insights のカウンター

以下のデータベースカウンターは、Aurora MySQL の Performance Insights で利用できます。

トピック

- Aurora MySQL のネイティブカウンター
- Aurora MySQL の非ネイティブカウンター

Aurora MySQL のネイティブカウンター

ネイティブメトリクスは、Amazon Aurora ではなく、データベースエンジンによって定義されます。これらのネイティブメトリクスの定義については、MySQL ドキュメントの「サーバーステータス変数」を参照してください。

Counter	タイプ	単位	メトリクス
Com_analyze	SQL	1 秒あたりのクエリ数	db.SQL.Com_analyze
Com_optimize	SQL	1 秒あたりのクエリ数	db.SQL.Com_optimize
Com_select	SQL	1 秒あたりのクエリ数	db.SQL.Com_select

Counter	タイプ	単位	メトリクス
Innodb_rows_deleted	SQL	1 秒あたりの行数	db.SQL.Innodb_rows_deleted
Innodb_rows_inserted	SQL	1 秒あたりの行数	db.SQL.Innodb_rows_inserted
Innodb_rows_read	SQL	1 秒あたりの行数	db.SQL.Innodb_rows_read
Innodb_rows_updated	SQL	1 秒あたりの行数	db.SQL.Innodb_rows_updated
クエリ	SQL	1 秒あたりのクエリ数	db.SQL.Queries
Questions	SQL	1 秒あたりのクエリ数	db.SQL.Questions
Select_full_join	SQL	1 秒あたりのクエリ数	db.SQL.Select_full_join
Select_full_range_join	SQL	1 秒あたりのクエリ数	db.SQL.Select_full_range_join
Select_range	SQL	1 秒あたりのクエリ数	db.SQL.Select_range
Select_range_check	SQL	1 秒あたりのクエリ数	db.SQL.Select_range_check

Counter	タイプ	単位	メトリクス
Select_scan	SQL	1 秒あたりのクエリ数	db.SQL.Select_scan
Slow_queries	SQL	1 秒あたりのクエリ数	db.SQL.Slow_queries
Sort_merge_passes	SQL	1 秒あたりのクエリ数	db.SQL.Sort_merge_passes
Sort_range	SQL	1 秒あたりのクエリ数	db.SQL.Sort_range
Sort_rows	SQL	1 秒あたりのクエリ数	db.SQL.Sort_rows
Sort_scan	SQL	1 秒あたりのクエリ数	db.SQL.Sort_scan
Total_query_time	SQL	ミリ秒	db.SQL.Total_query_time
Table_locks_immediate	ロック	1 秒あたりのリクエスト	db.Locks.Table_locks_immediate
Table_locks_waited	ロック	1 秒あたりのリクエスト	db.Locks.Table_locks_waited
Innodb_row_lock_time	ロック	ミリ秒 (平均)	db.Locks.Innodb_row_lock_time

Counter	タイプ	単位	メトリクス
Aborted_clients	[ユーザー]	接続	db.Users.Aborted_clients
Aborted_connects	[ユーザー]	接続	db.Users.Aborted_connects
接続	[ユーザー]	接続	db.Users.Connections
External_threads_connected	[ユーザー]	接続	db.Users.External_threads_connected
max_connections	[ユーザー]	接続	db.User.max_connections
Threads_connected	[ユーザー]	接続	db.Users.Threads_connected
Threads_created	[ユーザー]	接続	db.Users.Threads_created
Threads_running	[ユーザー]	接続	db.Users.Threads_running
Created_tmp_disk_tables	Temp	1 秒あたりのテーブル数	db.Temp.Created_tmp_disk_tables
Created_tmp_tables	Temp	1 秒あたりのテーブル数	db.Temp.Created_tmp_tables
Innodb_buffer_pool_pages_data	Cache	ページ	db.Cache.Innodb_buffer_pool_pages_data
Innodb_buffer_pool_pages_total	Cache	ページ	db.Cache.Innodb_buffer_pool_pages_total

Counter	タイプ	単位	メトリクス
Innodb_buffer_pool_read_requests	Cache	1 秒あたりのページ数	db.Cache.Innodb_buffer_pool_read_requests
Innodb_buffer_pool_reads	Cache	1 秒あたりのページ数	db.Cache.Innodb_buffer_pool_reads
Opened_tables	Cache	テーブル	db.Cache.Opened_tables
Opened_table_definitions	Cache	テーブル	db.Cache.Opened_table_definitions
Qcache_hits	Cache	クエリ	db.Cache.Qcache_hits

Aurora MySQL の非ネイティブカウンター

非ネイティブカウンターメトリクスは、Amazon RDS で定義されているカウンターです。非ネイティブメトリクスは、特定のクエリで取得するメトリクスである場合があります。非ネイティブメトリクスは派生メトリクスである場合もあります。この場合は、複数のネイティブカウンターが比率、ヒット率、またはレイテンシーの計算で使用されます。

Counter	タイプ	メトリクス	説明	定義
innodb_buffer_pool_hits	Cache	db.Cache.innoDB_buffer_pool_hits	InnoDB がバッファプールから満たすことができる読み取りの数。	`innodb_buffer_pool_read_requests - innodb_buffer_pool_reads`
innodb_buffer_pool_hit_rate	Cache	db.Cache.innoDB_buffer_pool_hit_rate	InnoDB がバッファプールから満たすことができる読み取りの割合 (%)。	`100 * innodb_buffer_pool_read_requests / (innodb_buffer_poo`

Counter	タイプ	メトリクス	説明	定義
				l_read_requests + innodb_buffer_pool_reads)
innodb_buffer_pool_usage	Cache	db.Cache.innoDB_buffer_pool_usage	データ (ページ) を含む InnoDB バッファプールの割合 (%)。 **Note** 圧縮テーブルを使用すると、この値は変動します。詳細については、MySQL ドキュメントの「サーバーステータス変数」の「Innodb_buffer_pool_pages_data 」と「Innodb_buffer_pool_pages_total 」を参照してください。	Innodb_buffer_pool_pages_data / Innodb_buffer_pool_pages_total * 100.0

Counter	タイプ	メトリクス	説明	定義
query_cache_hit_rate	Cache	db.Cache.query_cache_hit_rate	MySQL 結果セットキャッシュ (クエリキャッシュ) のヒット率。	`Qcache_hits / (QCache_hits + Com_select) * 100`
innodb_rows_changed	SQL	db.SQL.innodb_rows_changed	InnoDB の行オペレーションの合計数。	`db.SQL.Innodb_rows_inserted + db.SQL.Innodb_rows_deleted + db.SQL.Innodb_rows_updated`
active_transactions	トランザクション	db.Transactions.active_transactions	アクティブトランザクションの合計数。	`SELECT COUNT(1) AS active_transactions FROM INFORMATION_SCHEMA.INNODB_TRX`

Counter	タイプ	メトリクス	説明	定義
trx_rseg_history_len	トランザクション	db.Transactions.trx_rseg_history_len	マルチバージョン同時実行制御を実装するために InnoDB トランザクションシステムによって管理される、コミットされたトランザクションの UNDO ログページのリスト。元に戻すログレコードの詳細については、MySQL ドキュメントの「https://dev.mysql.com/doc/refman/8.0/en/innodb-multi-versioning.html」を参照してください。	`SELECT COUNT AS trx_rseg_history_len FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='trx_rseg_history_len'`
innodb_deadlocks	ロック	db.Locks.innodb_deadlocks	デッドロックの合計数。	`SELECT COUNT AS innodb_deadlocks FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='lock_deadlocks'`

Counter	タイプ	メトリクス	説明	定義
innodb_lock_timeouts	ロック	db.Locks.innodb_lock_timeouts	タイムアウトしたデッドロックの合計数。	`SELECT COUNT AS innodb_lock_timeouts FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='lock_timeouts'`
innodb_row_lock_waits	ロック	db.Locks.innodb_row_lock_waits	行ロックを待機した合計数。	`SELECT COUNT AS innodb_row_lock_waits FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='lock_row_lock_waits'`

Aurora PostgreSQL の Performance Insights カウンター

以下のデータベースカウンターは、Aurora PostgreSQL の Performance Insights で利用できます。

トピック

- Aurora PostgreSQL のネイティブカウンター
- Aurora PostgreSQL の非ネイティブカウンター

Aurora PostgreSQL のネイティブカウンター

ネイティブメトリクスは、Amazon Aurora ではなく、データベースエンジンによって定義されます。これらのネイティブメトリクスの定義については、PostgreSQL の「統計情報の表示」を参照してください。

Counter	タイプ	単位	メトリクス
tup_deleted	SQL	1 秒あたりのタプル数	db.SQL.tup_deleted
tup_fetched	SQL	1 秒あたりのタプル数	db.SQL.tup_fetched
tup_inserted	SQL	1 秒あたりのタプル数	db.SQL.tup_inserted
tup_returned	SQL	1 秒あたりのタプル数	db.SQL.tup_returned
tup_updated	SQL	1 秒あたりのタプル数	db.SQL.tup_updated
blks_hit	Cache	1 秒あたりのブロック数	db.Cache.blks_hit
buffers_alloc	Cache	1 秒あたりのブロック数	db.Cache.buffers_alloc
buffers_checkpoint	Checkpoint	1 秒あたりのブロック数	db.Checkpoint.buffers_checkpoint
checkpoints_req	Checkpoint	1 分あたりのチェックポイント数	db.Checkpoint.checkpoints_req
checkpoint_sync_time	Checkpoint	チェックポイントあたりのミリ秒数	db.Checkpoint.checkpoint_sync_time
checkpoints_timed	Checkpoint	1 分あたりのチェックポイント数	db.Checkpoint.checkpoints_timed
checkpoint_write_time	Checkpoint	チェックポイントあたりのミリ秒数	db.Checkpoint.checkpoint_write_time
maxwritten_clean	Checkpoint	1 分あたりの Bgwriter の完全停止数	db.Checkpoint.maxwritten_clean
deadlocks	Concurrency	1 分あたりのデッドロック数	db.Concurrency.deadlocks
blk_read_time	I/O	ミリ秒	db.IO.blk_read_time

Counter	タイプ	単位	メトリクス
blks_read	I/O	1 秒あたりのブロック数	db.IO.blks_read
buffers_backend	I/O	1 秒あたりのブロック数	db.IO.buffers_backend
buffers_backend_fsync	I/O	1 秒あたりのブロック数	db.IO.buffers_backend_fsync
buffers_clean	I/O	1 秒あたりのブロック数	db.IO.buffers_clean
temp_bytes	Temp	1 秒あたりのバイト数	db.Temp.temp_bytes
temp_files	Temp	1 分あたりのファイル数	db.Temp.temp_files
xact_commit	トランザクション	1 秒あたりのコミット数	db.Transactions.xact_commit
xact_rollback	トランザクション	1 秒あたりのロールバック数	db.Transactions.xact_rollback
numbackends	ユーザー	接続	db.User.numbackends
archived_count	WAL	1 分あたりのファイル数	db.WAL.archived_count

Aurora PostgreSQL の非ネイティブカウンター

非ネイティブカウンターメトリクスは、Amazon Aurora で定義されているカウンターです。非ネイティブメトリクスは、特定のクエリで取得するメトリクスである場合があります。非ネイティブメトリクスは派生メトリクスである場合もあります。この場合は、複数のネイティブカウンターが比率、ヒット率、またはレイテンシーの計算で使用されます。

Counter	タイプ	メトリクス	説明	定義
checkpoint_sync_latency	Checkpoint	db.Checkpoint.checkpoint_sync_latency	チェックポイント処理でファイルをディスクに同期する部分に費やした合計時間。	`checkpoint_sync_time / (checkpoints_timed + checkpoints_req)`
checkpoint_write_latency	Checkpoint	db.Checkpoint.checkpoint_write_latency	チェックポイント処理でファイルをディスクに書き込む部分に費やした合計時間。	`checkpoint_write_time / (checkpoints_timed + checkpoints_req)`
local_blks_read	I/O	db.IO.local_blks_read	読み取られたローカルブロックの総数。	–
local_blk_read_time	I/O	db.IO.local_blk_read_time	`track_io_timing` を有効にすると、ローカルデータファイルブロックの読み取りにかかった合計時間をミリ秒単位で追跡します。それ以外の場合、値は 0 です。詳細については、track_io_timing を参照してください。	–
orcache_blks_hit	I/O	db.IO.orcache_blks_hit	Optimized Reads キャッシュからの共有ブロックヒットの総数。	–
orcache_blk_read_time	I/O	db.IO.orcache_blk_read_time	`track_io_timing` を有効にすると、Optimized Reads キャッシュからデータファイルブロックを読み取るのにかかった合計時間をミリ秒単位で追跡します。それ以外の	–

Counter	タイプ	メトリクス	説明	定義
			場合は 0 です。詳細については、track_io_timing を参照してください。	
read_latency	I/O	db.IO.read_latency	このインスタンスのバックエンドでデータファイルブロックの読み取りに費やした時間。	`blk_read_time / blks_read`
storage_blks_read	I/O	db.IO.storage_blks_read	Aurora ストレージから読み取られた共有ブロックの総数。	-
storage_blk_read_time	I/O	db.IO.storage_blk_read_time	`track_io_timing` を有効にすると、Aurora ストレージからのデータファイルブロックの読み取りにかかった合計時間をミリ秒単位で追跡します。それ以外の場合、値はゼロです。詳細については、track_io_timing を参照してください。	-
idle_in_transaction_aborted_count	都道府県	db.state.idle_in_transaction_aborted_count	`idle in transaction (aborted)` 状態のセッションの数。	-
idle_in_transaction_count	都道府県	db.state.idle_in_transaction_count	`idle in transaction` 状態のセッションの数。	-

Counter	タイプ	メトリクス	説明	定義
idle_in_transaction_max_time	都道府県	db.state.idle_in_transaction_max_time	`idle in transaction` 状態で実行されている最も長いトランザクションの時間を秒単位で表します。	–
logical_reads	SQL	db.SQL.logical_reads	ヒットしたブロックと読み取ったブロックの合計数。	`blks_hit + blks_read`
queries_started	SQL	db.SQL.queries	開始されたクエリの数。	–
queries_finished	SQL	db.SQL.queries	完了したクエリの数。	–
total_query_time	SQL	db.SQL.total_query_time	ステートメントの実行にかかった合計時間をミリ秒単位で表します。	–
active_transactions	トランザクション	db.Transactions.active_transactions	アクティブなトランザクションの数。	–
blocked_transactions	トランザクション	db.Transactions.blocked_transactions	ブロックされたトランザクションの数。	–
commit_latency	トランザクション	db.Transactions.commit_latency	コミット操作の平均時間。	`db.Transactions.duration_commits / db.Transactions.xact_commit`

Counter	タイプ	メトリクス	説明	定義
duration_commits	トランザクション	db.Transactions.duration_commits	過去 1 分間の合計トランザクション時間をミリ秒単位で表します。	–
max_used_xact_ids	トランザクション	db.Transactions.max_used_xact_ids	バキューム処理されていないトランザクションの数。	–
max_connections	[ユーザー]	db.User.max_connections	`max_connections` パラメータで設定されたデータベースに許可される接続の最大数。	–
total_auth_attempts	[ユーザー]	db.User.total_auth_attempts	このインスタンスへの接続試行の数。	–
archive_failed_count	WAL	db.WAL.archive_failed_count	WAL ファイルのアーカイブに失敗した 1 分あたりのファイル数。	–

Performance Insights の SQL 統計

SQL 統計は、Performance Insights によって収集される SQL クエリに関するパフォーマンス関連のメトリックです。Performance Insights は、クエリが実行中の 1 秒ごとおよび SQL 呼び出しごとに統計を収集します。SQL 統計は、選択した時間範囲の平均です。

SQL ダイジェストは、特定のパターンを持つすべてのクエリの複合体ですが、必ずしも同じリテラル値を持つ必要はありません。ダイジェストは、リテラル値を疑問符に置き換えます。例えば、`SELECT * FROM emp WHERE lname= ?` です。このダイジェストは、次の子クエリで構成されます。

```
SELECT * FROM emp WHERE lname = 'Sanchez'
```

```
            SELECT * FROM emp WHERE lname = 'Olagappan'
            SELECT * FROM emp WHERE lname = 'Wu'
```

すべてのエンジンは、ダイジェストクエリの SQL 統計をサポートしています。

この機能のリージョン、DB エンジン、およびインスタンスクラスのサポート情報については、「Amazon Aurora DB エンジン、リージョン、およびインスタンスクラスでサポートされている Performance Insights 機能」を参照してください。

トピック

- Aurora MySQL のSQL統計
- Aurora PostgreSQL での SQL 統計

Aurora MySQL のSQL統計

Aurora MySQLは、ダイジェストレベルでのみSQL 統計を収集します。ステートメントレベルでは、統計は表示されません。

トピック

- Aurora MySQL の Digest 統計
- Aurora MySQL の秒単位の統計データ
- Aurora MySQL の呼び出しごとの統計データ

Aurora MySQL の Digest 統計

Performance Insightsは、`events_statements_summary_by_digest` テーブルから SQL ダイジェスト統計を収集します。`events_statements_summary_by_digest`テーブルは、データベースによって管理されます。

ダイジェストテーブルには削除ポリシーはありません。テーブルがいっぱいになると、AWS Management Console に次のメッセージが表示されます。

```
Performance Insights is unable to collect SQL Digest statistics on new queries because
 the table events_statements_summary_by_digest is full.
Please truncate events_statements_summary_by_digest table to clear the issue. Check the
 User Guide for more details.
```

このような状況では、、Aurora MySQLはSQL クエリを追跡しません。この問題に対処するため、Performance Insights は、次の条件の両方が満たされた場合に、ダイジェストテーブルを自動的に切り捨てます。

- テーブルがいっぱいの場合、
- Performance Insights は、Performance Schema を自動的に管理します。

  自動管理の場合、`performance_schema` パラメータを `0` に設定する必要があります。[Source (ソース)] を `user` に設定しないでください。Performance Insights がパフォーマンススキーマを自動的に管理していない場合は、Aurora MySQL における Performance Insights の Performance Schema の有効化 を参照してください。

AWS CLI で、describe-db-pameters コマンドを実行し、パラメータ値のソースをチェックします。

Aurora MySQL の秒単位の統計データ

次の SQL 統計は、Aurora MySQL DB クラスターで使用できます。

メトリクス	Unit
db.sql_tokenized.stats.count_star_per_sec	1 秒あたりの呼び出し数
db.sql_tokenized.stats.sum_timer_wait_per_sec	1 秒あたりの平均アクティブ実行 (AAE)
db.sql_tokenized.stats.sum_select_full_join_per_sec	1 秒ごとに完全結合を選択
db.sql_tokenized.stats.sum_select_range_check_per_sec	1 秒ごとに範囲チェックを選択
db.sql_tokenized.stats.sum_select_scan_per_sec	1 秒ごとにスキャンを選択
db.sql_tokenized.stats.sum_sort_merge_passes_per_sec	1 秒ごとにマージパスを並べ替え
db.sql_tokenized.stats.sum_sort_scan_per_sec	1 秒あたりの並べ替えスキャン数

メトリクス	Unit
db.sql_tokenized.stats.sum_sort_rang e_per_sec	1 秒ごとの並べ替え範囲
db.sql_tokenized.stats.sum_sort_rows_per_sec	1 秒あたりの行の並べ替え
db.sql_tokenized.stats.sum_rows_affected_per_ sec	1 秒あたりの影響を受ける行数
db.sql_tokenized.stats.sum_rows_exam ined_per_sec	1 秒あたりの検査される行数
db.sql_tokenized.stats.sum_rows_sent_per_sec	1 秒あたりに送信される行数
db.sql_tokenized.stats.sum_created_tmp_disk_t ables_per_sec	1 秒ごとに作成されるテンポラリディスクテーブル
db.sql_tokenized.stats.sum_created_tmp_tables _per_sec	1 秒ごとに作成されるテンポラリテーブル
db.sql_tokenized.stats.sum_lock_time_per_sec	1 秒あたりのロック時間 (ミリ秒)

Aurora MySQL の呼び出しごとの統計データ

以下のメトリクスは、SQL ステートメントの呼び出しごとの統計を提供します。

メトリクス	単位
db.sql_tokenized.stats.sum_timer_wait_per_call	呼び出しごとの平均レイテンシー (ミリ秒)
db.sql_tokenized.stats.sum_select_full_join_p er_call	コールごとに完全結合を選択
db.sql_tokenized.stats.sum_select_ra nge_check_per_call	コールごとに範囲チェックを選択
db.sql_tokenized.stats.sum_select_sc an_per_call	コールごとにスキャンを選択

メトリクス	単位
db.sql_tokenized.stats.sum_sort_merge_passes_per_call	コールごとにマージパスを並べ替え
db.sql_tokenized.stats.sum_sort_scan_per_call	コールごとに並べ替えスキャン
db.sql_tokenized.stats.sum_sort_range_per_call	コールごとの並べ替え範囲
db.sql_tokenized.stats.sum_sort_rows_per_call	呼び出しごとの行の並べ替え
db.sql_tokenized.stats.sum_rows_affected_per_call	コールごとに影響を受ける行数
db.sql_tokenized.stats.sum_rows_examined_per_call	コールごとに検査される行数
db.sql_tokenized.stats.sum_rows_sent_per_call	コールごとに送信される行数
db.sql_tokenized.stats.sum_created_tmp_disk_tables_per_call	コールごとに作成されたテンポラリディスクテーブル数
db.sql_tokenized.stats.sum_created_tmp_tables_per_call	コールごとに作成されたテンポラリテーブル数
db.sql_tokenized.stats.sum_lock_time_per_call	呼び出しごとのロック時間 (ミリ秒)

Aurora PostgreSQL での SQL 統計

Performance Insights は、SQL 呼び出しごとおよびクエリが実行中の 1 秒ごとに SQL 統計を収集します。すべての Aurora エンジンは、ダイジェストレベルでのみ統計を収集します。

Aurora PostgreSQL のダイジェストレベルの統計の詳細については、以下を参照してください。

トピック

- Aurora PostgreSQL でのダイジェスト統計
- Aurora PostgreSQL での秒単位のダイジェスト統計
- Aurora PostgreSQL のコールごとのダイジェスト統計

Aurora PostgreSQL でのダイジェスト統計

SQL ダイジェストの統計を表示するには、pg_stat_statements ライブラリをロードする必要があります。PostgreSQL 10 と互換性のある Aurora PostgreSQL DB クラスターでは、このライブラリはデフォルトでロードされます。PostgreSQL 9.6 と互換性のある Aurora PostgreSQL DB クラスターでは、このライブラリを手動で有効にします。手動で有効にするには、DB インスタンスに関連付けられた DB パラメータグループの shared_preload_libraries に pg_stat_statements を追加します。次に DB インスタンスを再起動します。詳細については、「「パラメータグループを使用する」」を参照してください。

> **Note**
>
> Performance Insights は、切り捨てられない pg_stat_activity 内のクエリの統計のみを収集できます。デフォルトでは、PostgreSQL データベースは 1,024 バイトより長い問い合わせを切り捨てます。問い合わせサイズを増やすには、DB インスタンスに関連付けられた DB パラメータグループの track_activity_query_size パラメータを変更します。このパラメータを変更した場合は、DB インスタンスの再起動が必要です。

Aurora PostgreSQL での秒単位のダイジェスト統計

Aurora PostgreSQL DB インスタンスでは、次の SQL ダイジェストの統計を使用できます。

メトリクス	単位
db.sql_tokenized.stats.calls_per_sec	1 秒あたりの呼び出し数
db.sql_tokenized.stats.rows_per_sec	1 秒あたりの行数
db.sql_tokenized.stats.total_time_per_sec	1 秒あたりの平均アクティブ実行 (AAE)
db.sql_tokenized.stats.shared_blks_hit_per_sec	1 秒あたりのブロックヒット数
db.sql_tokenized.stats.shared_blks_read_per_sec	1 秒あたりのブロック読み取り数
db.sql_tokenized.stats.shared_blks_dirtied_per_sec	1 秒あたりのダーティになったブロック数

メトリクス	単位
db.sql_tokenized.stats.shared_blks_written_per_sec	1 秒あたりのブロック書き込み数
db.sql_tokenized.stats.local_blks_hit_per_sec	1 秒あたりのローカルブロックヒット数
db.sql_tokenized.stats.local_blks_read_per_sec	1 秒あたりのローカルブロック読み取り数
db.sql_tokenized.stats.local_blks_dirtied_per_sec	1 秒あたりのローカルブロックのダーティの数
db.sql_tokenized.stats.local_blks_written_per_sec	1 秒あたりのローカルブロック書き込み数
db.sql_tokenized.stats.temp_blks_written_per_sec	1 秒あたりの一時的な書き込み数
db.sql_tokenized.stats.temp_blks_read_per_sec	1 秒あたりの一時的な読み取り数
db.sql_tokenized.stats.blk_read_time_per_sec	1 秒あたりの平均的な同時読み取り数
db.sql_tokenized.stats.blk_write_time_per_sec	1 秒あたりの平均的な同時書き込み数

Aurora PostgreSQL のコールごとのダイジェスト統計

以下のメトリクスは、SQL ステートメントの呼び出しごとの統計を提供します。

メトリクス	単位
db.sql_tokenized.stats.rows_per_call	呼び出しごとの行数
db.sql_tokenized.stats.avg_latency_per_call	呼び出しごとの平均レイテンシー (ミリ秒)
db.sql_tokenized.stats.shared_blks_hit_per_call	呼び出しごとのブロックヒット数
db.sql_tokenized.stats.shared_blks_read_per_call	呼び出しごとのブロック読み取り数

メトリクス	単位
db.sql_tokenized.stats.shared_blks_written_per_call	呼び出しごとのブロック書き込み数
db.sql_tokenized.stats.shared_blks_dirtied_per_call	呼び出しあたりのダーティになったブロック数
db.sql_tokenized.stats.local_blks_hit_per_call	呼び出しあたりのローカルブロックヒット数
db.sql_tokenized.stats.local_blks_read_per_call	呼び出しあたりのローカルブロック読み取り数
db.sql_tokenized.stats.local_blks_dirtied_per_call	呼び出しあたりのローカルブロックのダーティの数
db.sql_tokenized.stats.local_blks_written_per_call	呼び出しあたりのローカルブロック書き込み数
db.sql_tokenized.stats.temp_blks_written_per_call	呼び出しあたりのテンポラリブロック書き込み数
db.sql_tokenized.stats.temp_blks_read_per_call	呼び出しあたりのテンポラリブロック読み取り数
db.sql_tokenized.stats.blk_read_time_per_call	呼び出しごとの読み取り時間 (ミリ秒)
db.sql_tokenized.stats.blk_write_time_per_call	呼び出しごとの書き込み時間 (ミリ秒)

これらのメトリクスの詳細については、PostgreSQL ドキュメントの「pg_stat_statements」を参照してください。

拡張モニタリングの OS メトリクス

Amazon Aurora には、DB クラスターが実行されているオペレーティングシステム (OS) のリアルタイムのメトリクスが用意されています。Aurora は、拡張モニタリングのメトリクスを Amazon CloudWatch Logs アカウントに配信します。以下の表では、Amazon CloudWatch Logs で使用できる OS メトリクスを示しています。

トピック

- Aurora の OS メトリクス

Aurora の OS メトリクス

グループ	メトリクス	コンソール名	説明
General	engine	該当しません	DB インスタンスのデータベースエンジン。
	instanceID	該当しません	DB インスタンス識別子。
	instanceResourceID	該当しません	AWS リージョンに固有の DB インスタンスの不変の識別子。ログストリーム識別子としても使用されます。
	numVCPUs	該当しません	DB インスタンスの仮想 CPU の数。
	timestamp	該当しません	メトリクスが作成された時間。
	uptime	該当しません	DB インスタンスがアクティブになってからの経過時間。
	version	該当しません	OS メトリクスのストリーム JSON 形式のバージョン。
cpuUtilization	guest	CPU ゲスト	ゲストプログラムが使用中の CPU の使用率。
	idle	CPU アイドル	アイドル状態の CPU の使用率。
	irq	CPU IRQ	ソフトウェア割り込みが使用中の CPU の使用率。
	nice	CPU Nice	最も低い優先順位で実行されているプログラムが使用中の CPU の使用率。

グループ	メトリクス	コンソール名	説明
	steal	CPU Steal	他の仮想マシンが使用中の CPU の使用率。
	system	CPU システム	カーネルが使用中の CPU の使用率。
	total	CPU 合計	使用中の CPU の合計使用率。この値は nice 値を含みます。
	user	CPU ユーザー	ユーザープログラムが使用中の CPU の使用率。
	wait	CPU 待機	I/O アクセスを待機中の CPU の未使用率。
diskIO	avgQueueLen	平均キューサイズ	I/O デバイスのキューで待機中のリクエストの数。
	avgReqSz	平均リクエストサイズ	平均リクエストサイズ (キロバイト単位)。
	await	ディスク I/O 待機	リクエストへの応答に必要なミリ秒数 (キュー時間とサービス時間を含む)。
	device	該当しません	使用中のディスクデバイスの識別子。
	readIOsPS	読み取り I/O/秒	読み取りオペレーションの 1 秒あたりの数。
	readKb	読み取り合計	読み取りの合計キロバイト数。
	readKbPS	読み取り KB/秒	読み取りの 1 秒あたりのキロバイト数。

グループ	メトリクス	コンソール名	説明
	readLatency	読み取りレイテンシー	読み取り I/O リクエスト送信からその完了までの経過時間 (ミリ秒単位) 。 このメトリクスは Amazon Aurora にのみ使用できます。
	readThroughput	読み取りスループット	DB クラスターへのリクエストによって使用されるネットワークスループットの量 (バイト/秒単位)。 このメトリクスは Amazon Aurora にのみ使用できます。
	rrqmPS	Rrqms	キューに入れられてマージされた読み取りリクエストの 1 秒あたりの数。
	tps	TPS	I/O トランザクションの 1 秒あたりの数。
	util	ディスク I/O 使用率	リクエスト発行中の CPU 時間の消費率。
	writeIOsPS	書き込み IO/秒	書き込みオペレーションの 1 秒あたりの数。
	writeKb	書き込み合計	書き込みの合計キロバイト数。
	writeKbPS	書き込み KB/秒	書き込みの 1 秒あたりのキロバイト数。
	writeLatency	書き込みレイテンシー	書き込み I/O リクエスト送信から完了までの平均経過時間 (ミリ秒単位) 。 このメトリクスは Amazon Aurora にのみ使用できます。

グループ	メトリクス	コンソール名	説明
	writeThroughput	書き込みスループット	DB クラスターからのレスポンスによって使用されるネットワークスループットの量 (バイト/秒単位)。 このメトリクスは Amazon Aurora にのみ使用できます。
	wrqmPS	Wrqms	キューに入れられてマージされた書き込みリクエストの 1 秒あたりの数。
fileSys	maxFiles	最大 i ノード	ファイルシステム用に作成できるファイルの最大数。
	mountPoint	該当しません	ファイルシステムへのパス。
	name	該当しません	ファイルシステムの名前。
	total	合計ファイルシステム	ファイルシステムに使用できるディスク領域の合計量 (キロバイト単位)。
	used	使用済みファイルシステム	ファイルシステム内のファイルが使用中のディスク領域の量 (キロバイト単位)。
	usedFilePercent	使用済み i ノード	使用中のファイルの割合。
	usedFiles	使用済み (%)	ファイルシステム内のファイルの数。
	usedPercent	使用済みファイルシステム	ファイルシステムが使用中のディスク領域の割合。

グループ	メトリクス	コンソール名	説明
loadAverageMinute	fifteen	平均負荷 (15 分)	過去 15 分間に CPU 時間をリクエストしたプロセスの数。
	five	平均負荷 (5 分)	過去 5 分間に CPU 時間をリクエストしたプロセスの数。
	one	平均負荷 (1 分)	過去 1 分間に CPU 時間をリクエストしたプロセスの数。
memory	active	アクティブなメモリ	割り当てられたメモリの量 (キロバイト単位)。
	buffers	バッファ済みメモリ	ストレージデバイスへの書き込み前に I/O バッファリングリクエストに使用されたメモリの量 (キロバイト単位)。
	cached	キャッシュ済みメモリ	ファイルシステムベースの I/O のキャッシュに使用されたメモリの量。
	dirty	ダーティメモリ	変更されたがストレージ内のその関連データブロックに書き込まれなかった RAM 内のメモリページの量 (キロバイト単位)。
	free	空きメモリ	未割り当てのメモリの量 (キロバイト単位)。
	hugePagesFree	huge ページ (空き)	空き huge ページの数。huge ページは Linux カーネルの機能です。
	hugePagesRsvd	huge ページ (予約)	コミットされた huge ページの数。
	hugePagesSize	huge ページサイズ	各 huge ページユニットのサイズ (キロバイト単位)。
	hugePagesSurp	huge ページ (余剰)	使用可能な huge ページの余剰数/合計数。

グループ	メトリクス	コンソール名	説明
	hugePagesTotal	huge ページ (合計)	huge ページの合計数。
	inactive	非アクティブメモリ	最も使用されていないメモリページの量 (キロバイト単位)。
	mapped	マップ済みメモリ	プロセスアドレス空間内でメモリマップされているファイルシステムの内容の合計量 (キロバイト単位)。
	pageTables	ページテーブル	ページテーブルが使用中のメモリの量 (キロバイト単位)。
	slab	スラブメモリ	再利用可能なカーネルデータ構造体の量 (キロバイト単位)。
	total	合計メモリ	メモリの合計量 (キロバイト単位)。
	writeback	書き戻しメモリ	バックアップストレージにまだ書き込み中の RAM 内のダーティページの量 (キロバイト単位)。
network	interface	該当しません	DB インスタンスに使用中のネットワークインターフェイスの識別子。
	rx	RX	1 秒あたりの受信バイト数。
	tx	TX	1 秒あたりのアップロードバイト数。
processList	cpuUsedPc	CPU %	プロセスが使用中の CPU の使用率。
	id	該当しません	プロセスの識別子。
	memoryUsedPc	MEM%	プロセスが使用中のメモリの使用率。

グループ	メトリクス	コンソール名	説明
	name	該当しません	プロセスの名前。
	parentID	該当しません	プロセスの親プロセスの識別子。
	rss	RES	プロセスに割り当てられた RAM の量 (キロバイト単位)。
	tgid	該当しません	スレッドグループ識別子 (スレッドが属するプロセス ID を表す番号)。この識別子は同じプロセスのグループスレッドに使用されます。
	vss	VIRT	プロセスに割り当てられた仮想メモリの量 (キロバイト単位)。
swap	swap	スワップ	使用可能なスワップメモリの量 (キロバイト単位)。
	swap in	スワップイン	ディスクからスワップされたメモリの量 (キロバイト単位)。
	swap out	スワップアウト	ディスクにスワップされたメモリの量 (キロバイト単位)。
	free	空きスワップ	空きスワップメモリの量 (キロバイト単位)。
	committed	コミット済みスワップ	キャッシュメモリとして使用されたスワップメモリの量 (キロバイト単位)。
tasks	blocked	ブロック済みタスク	ブロックされているタスクの数。
	running	実行中のタスク	実行中のタスクの数。

グループ	メトリクス	コンソール名	説明
	sleeping	スリープ中のタスク	スリープ中のタスクの数。
	stopped	停止済みタスク	停止中のタスクの数。
	total	タスク合計	タスクの合計数。
	zombie	Zombie タスク	アクティブな親タスクの非アクティブな子タスクの数。

Amazon Aurora DB クラスターでの、イベント、ログ、およびストリーミングのモニタリング

Amazon Aurora データベースおよびその他の AWS ソリューションをモニタリングする場合、目標は次の条件を維持することです。

- 信頼性
- 可用性
- パフォーマンス
- セキュリティ

「Amazon Aurora クラスターでのメトリクスのモニタリング」では、メトリクスを使用してクラスターをモニタリングする方法について説明します。完全なソリューションでは、データベースイベント、ログファイル、およびアクティビティストリーミングもモニタリングする必要があります。AWS には、次のモニタリングツールが用意されています。

- Amazon EventBridge は、アプリケーションをさまざまなソースのデータに簡単に接続できるようにするサーバーレスイベントバスサービスです。EventBridge は、お客様独自のアプリケーション、Software-as-a-Service (SaaS) アプリケーション、および AWS サービスから受け取るリアルタイムデータをストリームとして配信します。EventBridge では、データは AWS Lambda などのターゲットにルーティングされます。これにより、サービスで発生したイベントをモニタリングし、イベント駆動型アーキテクチャを構築できます。詳細については、「Amazon EventBridge ユーザーガイド」を参照してください。
- Amazon CloudWatch Logs を使用すると、 Amazon Aurora インスタンス、AWS CloudTrail、およびその他のソースからのログファイルをモニタリングして保存し、それらにアクセスすることができます。Amazon CloudWatch Logs は、ログファイル内の情報をモニタリングし、特定のしきい値が満たされたときに通知します。高い耐久性を備えたストレージにログデータをアーカイブすることもできます。詳細については、『Amazon CloudWatch Logs ユーザーガイド』を参照してください。
- AWS CloudTrail は、AWS アカウントによって行われた、またはそのアカウントに代わって実行された API コールと関連イベントをキャプチャします。次に、CloudTrailは指定した Amazon S3 バケットにログファイルを渡します。AWS を呼び出したユーザーとアカウント、呼び出し元の IP アドレス、および呼び出しの発生日時を特定できます。詳細については、AWS CloudTrail ユーザーガイドを参照してください。

- データベースアクティビティストリームは、Amazon Aurora の機能であり、DB クラスター内のアクティビティのストリームをほぼリアルタイムで提供します。Amazon Aurora は、アクティビティを Amazon Kinesis データストリーミングにプッシュします。Kinesis ストリーミングが自動的に作成されます。Kinesis から、Amazon Data Firehose や AWS Lambda などの AWS サービスを設定して、ストリーミングを消費し、データを保存できます。

トピック

Amazon RDS コンソールでのログ、イベント、およびストリーミングの表示

Amazon RDS が AWS のサービス と統合し、RDS コンソールでログ、イベント、およびデータベースアクティビティストリーミングに関する情報を表示できるようになりました。

Aurora DB クラスターの [Logs & events] (ログとイベント) タブで、次の情報が表示されます。

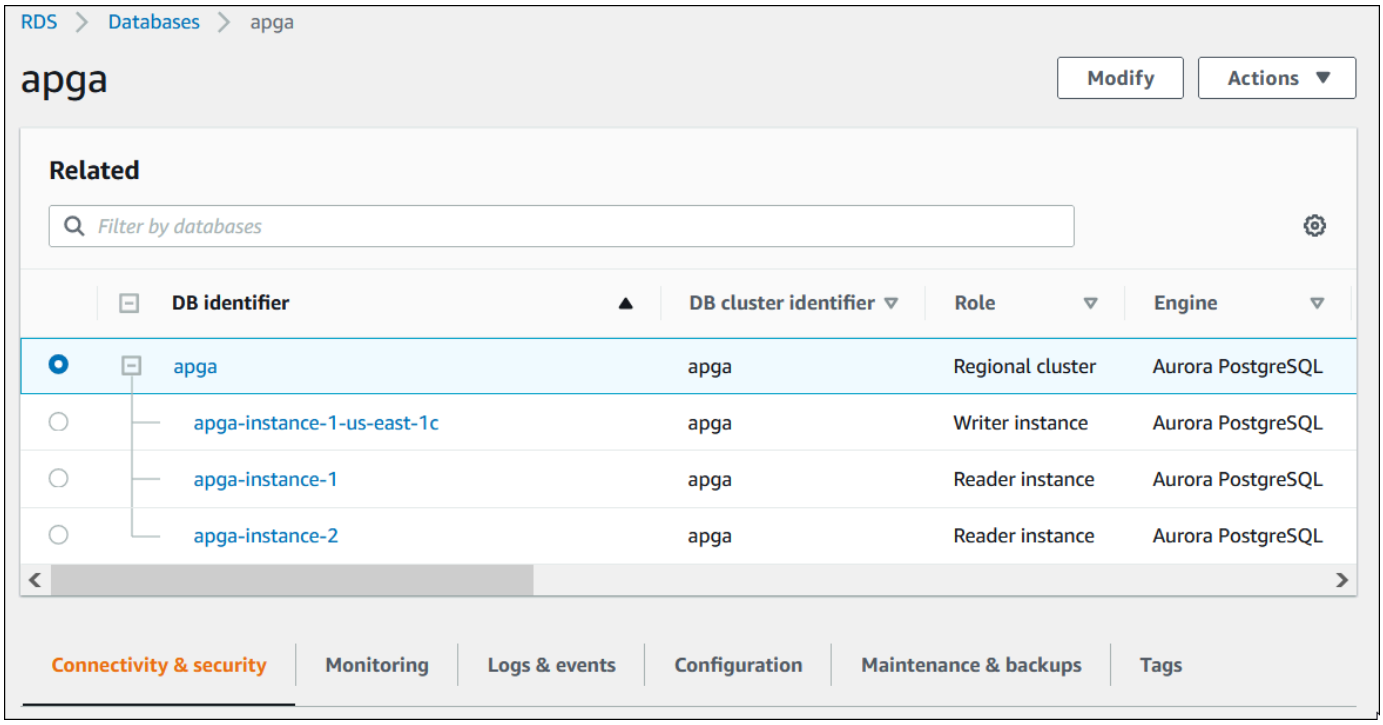
- [Auto scaling policies and activities] (Auto Scaling ポリシーおよびアクティビティ) – Aurora Auto Scaling 機能に関連するポリシーおよびアクティビティを表示します。この情報は、クラスターレベルで [Logs & events] (ログとイベント) タブでのみ表示されます。
- [Amazon CloudWatch alarms] (Amazon CloudWatch アラーム) – Aurora クラスター内の DB インスタンス用に設定したメトリクスアラームがすべて表示されます。アラームを設定していない場合は、RDS コンソールでアラームを作成できます。
- [Recent events] (最近のイベント) – Aurora DB インスタンスまたはクラスターのイベント (環境の変更) の概要が表示されます。詳細については、「Amazon RDS イベントの表示」を参照してください。
- [Logs] (ログ) – Aurora クラスター内の DB インスタンスによって生成されたデータベースログファイルが表示されます。詳細については、「Amazon Aurora ログファイルのモニタリング」を参照してください。

[Configuration] (設定) タブには、データベースアクティビティストリーミングに関する情報が表示されます。

RDS コンソールで、Aurora DB クラスターのログ、イベント、およびストリームを表示するには

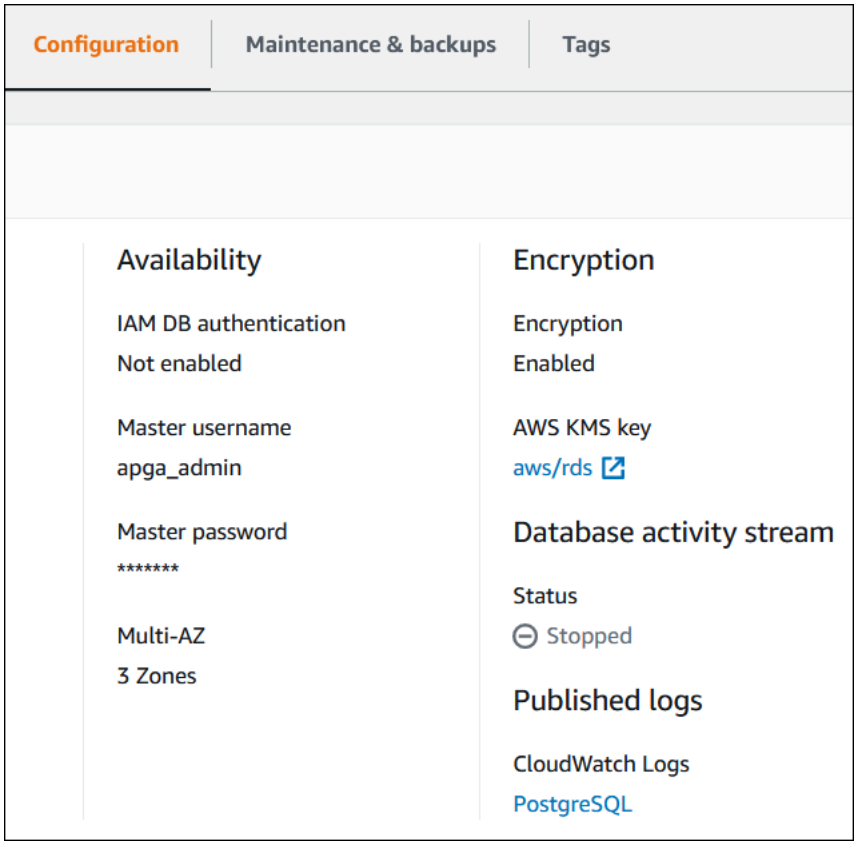
1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、データベース を選択します。
3. モニタリングする Aurora DB クラスターの名前を選択します。

   [Database] (データベース) ページが表示されます。次の例は、apga という名前の Amazon Aurora PostgreSQL DB クラスターを示しています。



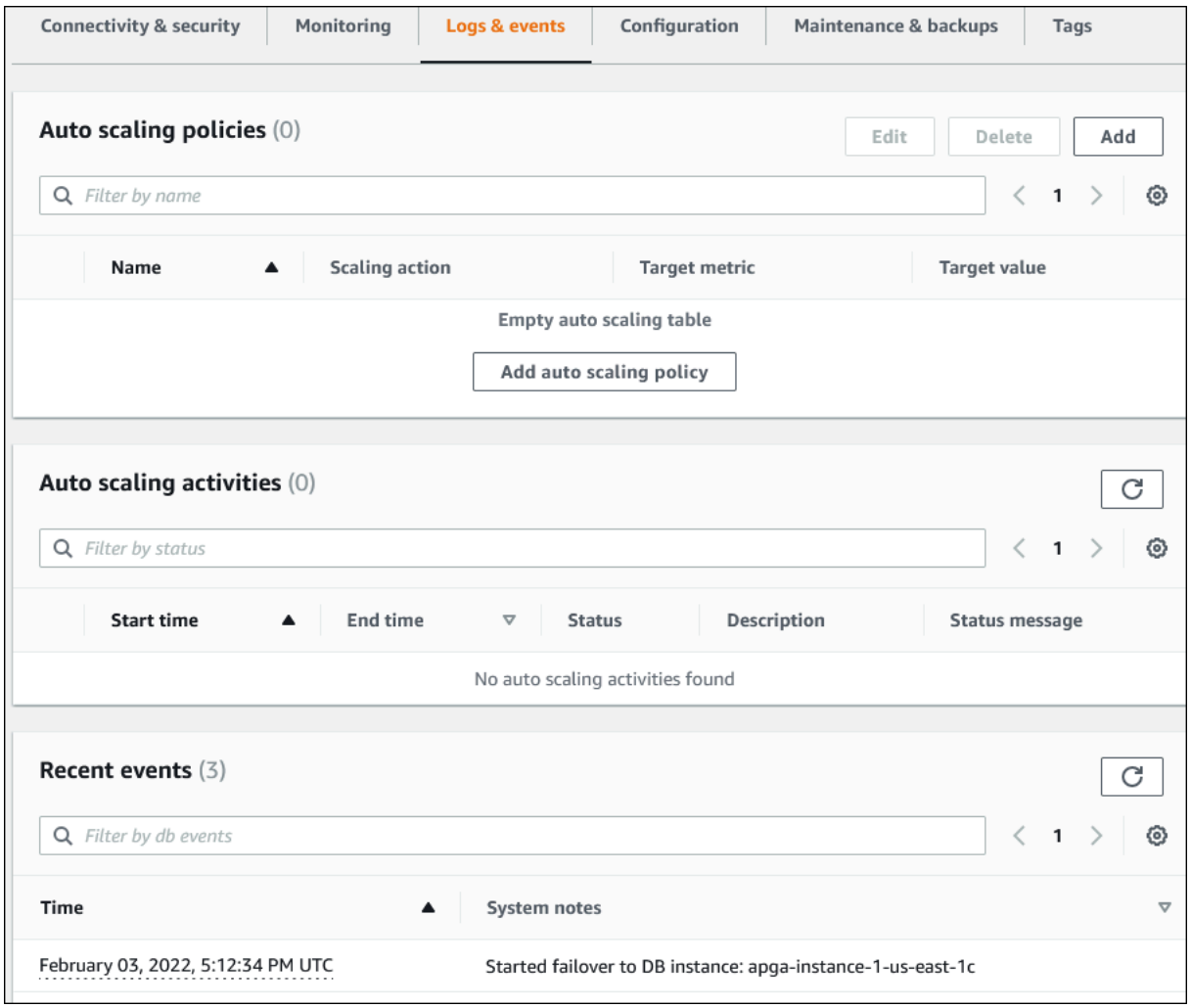
4. 下にスクロールし、[Configuration] (設定) を選択します。

   次の例は、クラスターのデータベースアクティビティストリーミングのステータスを示しています。



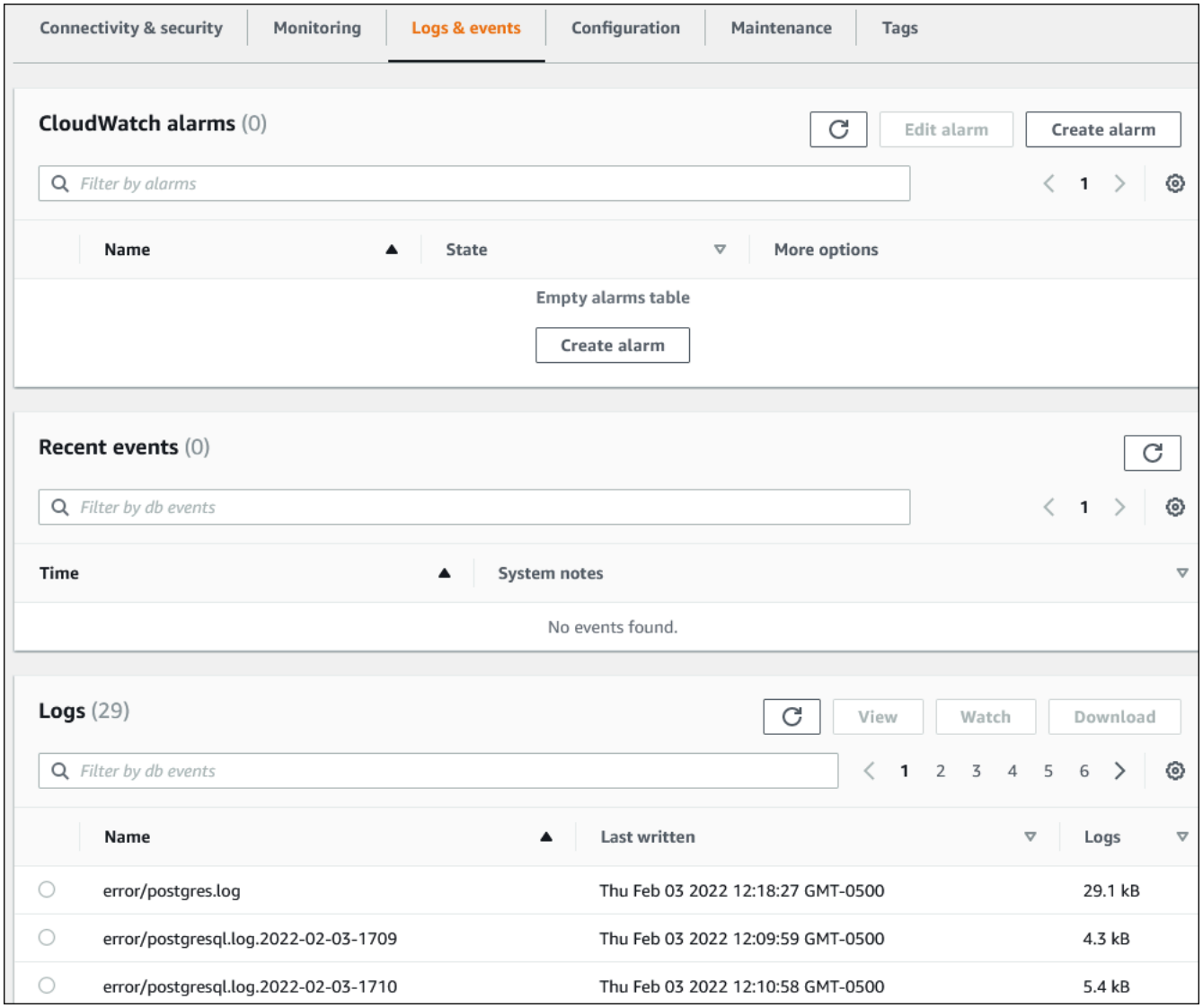
5. [ログとイベント] を選択します。

   [Logs & events] (ログとイベント) セクションが表示されます。



6. Aurora クラスター内の DB インスタンスを選択した上で、そのインスタンスの [Logs & events] (ログとイベント) を選択します。

   次の例は、[DB instance] (DB インスタンス) ページと [DB cluster] (DB クラスター) ページで内容が異なることを示しています。[DB instance] (DB インスタンス) ページには、ログとアラームが表示されます。


Connectivity & security
Monitoring
Logs & events
Configuration
Maintenance
Tags
CloudWatch alarms (0)
Edit alarm
Create alarm
Filter by alarms
1
Name
State
More options
Empty alarms table
Create alarm
Recent events (0)
Filter by db events
1
Time
System notes
No events found.
Logs (29)
View
Watch
Download
Filter by db events
1 2 3 4 5 6
Name
Last written
Logs
error/postgres.log
Thu Feb 03 2022 12:18:27 GMT-0500
29.1 kB
error/postgresql.log.2022-02-03-1709
Thu Feb 03 2022 12:09:59 GMT-0500
4.3 kB
error/postgresql.log.2022-02-03-1710
Thu Feb 03 2022 12:10:58 GMT-0500
5.4 kB

Amazon Aurora イベントのモニタリング

イベントは環境の変更を示します。これは、AWS 環境、SaaS パートナーサービスやアプリケーション、カスタムアプリケーションまたはサービスのいずれかです。Aurora イベントの説明については、「Amazon RDS のイベントカテゴリとイベントメッセージ」を参照してください。

トピック

Aurora のイベントの概要

RDS イベントは、Aurora 環境における変更を示します。例えば、DB クラスターにパッチが適用され、と Amazon Aurora はイベントを生成します。Amazon Aurora は、EventBridge に対してほぼリアルタイムでイベントを配信します。

> **Note**
>
> Amazon RDS は、ベストエフォートベースでイベントを発行します。通知イベントの順序または存在に依存するプログラムは、順序が正しくないか、または欠落している可能性があるため、作成しないことをお勧めします。

Amazon RDS は、次のリソースに関連するイベントを記録します。

- DB クラスター

  クラスターイベントのリストについては、「DB クラスターイベント」を参照してください。
- DB インスタンス

  DB インスタンスイベントのリストについては、「DB インスタンスイベント」を参照してください。
- DB パラメータグループ

DB パラメータグループイベントのリストについては、「DB パラメータグループイベント」を参照してください。

- DB セキュリティグループ

  DB セキュリティグループイベントのリストについては、「DB セキュリティグループイベント」を参照してください。

- DB クラススナップショット

  DB クラススナップショットイベントのリストについては、「DB クラススナップショットイベント」を参照してください。

- RDS Proxy イベント

  RDS Proxy イベントのリストについては、「RDS Proxy イベント」を参照してください。

- ブルー/グリーンデプロイイベント

  ブルー/グリーンデプロイイベントのリストについては、「ブルー/グリーンデプロイイベント」を参照してください。

この情報には以下が含まれます。

- リクエストの日付と時刻
- イベントのソース名とソースタイプ
- イベントに関連付けられたメッセージ。
- イベント通知には、メッセージが送信されたときのタグが含まれ、イベントが発生した時点のタグが反映されない場合があります

Amazon RDS イベントの表示

Amazon Aurora リソースの次のイベント情報を取得できます:

- リソース名
- リソースタイプ
- イベントの時刻
- イベントのメッセージの概要

AWS Management Console からイベントにアクセスして、過去 24 時間のイベントを確認できます。また、describe-events AWS CLI コマンドまたは DescribeEvents RDS API オペレーションを使用してイベントを取得することもできます。AWS CLI または RDS API を使用してイベントを表示する場合は、最大で過去 14 日間のイベントを取得できます。

> **Note**
>
> イベントを長期間保存する必要がある場合は、Amazon RDS イベントを EventBridge に送信できます。詳細については、「Amazon Aurora イベントでトリガーするルールの作成」を参照してください。

Amazon Aurora イベントの説明については、「Amazon RDS のイベントカテゴリとイベントメッセージ」を参照してください。

AWS CloudTrail を使用してリクエストパラメータを含むイベントの詳細情報にアクセスするには、「CloudTrail のイベント」を参照してください。

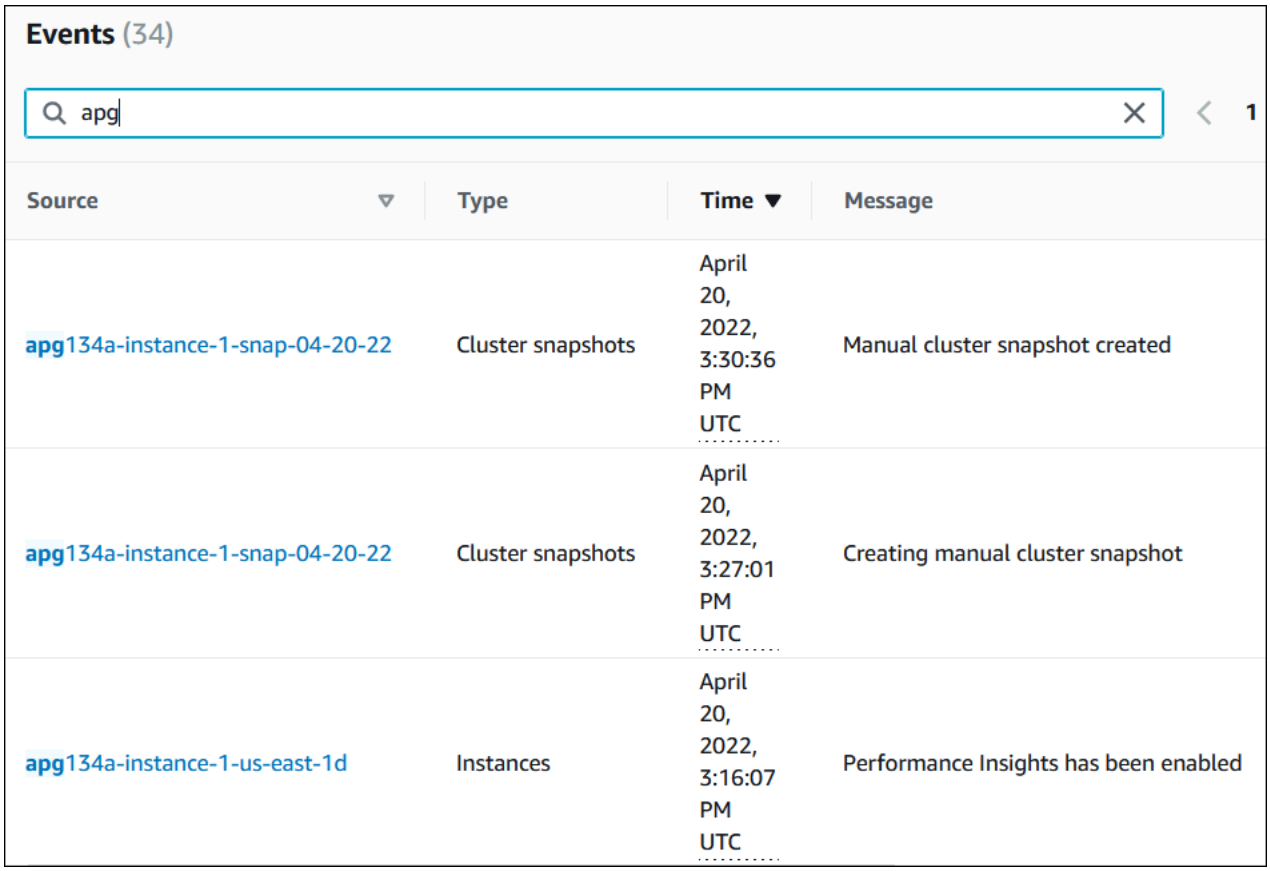
コンソール

過去 24 時間のすべての Amazon RDS イベントを表示するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインの [Events] (イベント) を選択します。

   使用できるイベントがリストに表示されます。
3. (オプション) 検索語を入力して、結果をフィルタリングします。

次の例は、文字 **apg** によってフィルタリングされたイベントのリストを示しています。



AWS CLI

過去 1 時間に生成されたすべてのイベントを表示するには、describe-events をパラメータを指定せずに呼び出します。

```
aws rds describe-events
```

次の出力例は、DB クラスターインスタンスが復旧を開始したことを示しています。

```
{
    "Events": [
        {
            "EventCategories": [
                "recovery"
            ],
            "SourceType": "db-instance",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:mycluster-instance-1",
```

```
            "Date": "2022-04-20T15:02:38.416Z",
            "Message": "Recovery of the DB instance has started. Recovery time will
 vary with the amount of data to be recovered.",
            "SourceIdentifier": "mycluster-instance-1"
        }, ...
```

過去 10080 分間 (7 日間) のすべての Amazon RDS イベントを表示するには、describe-events AWS CLI コマンドを呼び出し、`--duration` パラメータを `10080` に設定します。

```
aws rds describe-events --duration 10080
```

次の例は、DB インスタンス *testtest-instance* について、指定された時間範囲内のイベントを示しています。

```
aws rds describe-events \
    --source-identifier test-instance \
    --source-type db-instance \
    --start-time 2022-03-13T22:00Z \
    --end-time 2022-03-13T23:59Z
```

次の出力例は、バックアップのステータスを示しています。

```
{
    "Events": [
        {
            "SourceType": "db-instance",
            "SourceIdentifier": "test-instance",
            "EventCategories": [
                "backup"
            ],
            "Message": "Backing up DB instance",
            "Date": "2022-03-13T23:09:23.983Z",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance"
        },
        {
            "SourceType": "db-instance",
            "SourceIdentifier": "test-instance",
            "EventCategories": [
                "backup"
            ],
            "Message": "Finished DB Instance backup",
            "Date": "2022-03-13T23:15:13.049Z",
```

```
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance"
        }
    ]
}
```

API

過去 14 日間のすべての Amazon RDS インスタンスのイベントを表示するには、RDS API の DescribeEvents オペレーションを呼び出して、`Duration` パラメータを `20160` に設定します。

Amazon RDS イベント通知の操作

Amazon RDS では、Amazon RDS のイベントが発生したときに、Amazon Simple Notification Service (Amazon SNS) を使用して通知を送信します。これらの通知については、AWS リージョンの Amazon SNS でサポートされているすべての通知の形式が使用可能です (E メール、テキストメッセージ、HTTP エンドポイントの呼び出しなど)。

トピック

Amazon RDS イベント通知の概要

Amazon RDS は、サブスクライブ可能なカテゴリにイベントをグループ分けします。これにより、そのカテゴリのイベントが発生すると、通知を受け取ることができます。

トピック

イベントサブスクリプションの対象となる RDS リソース

Amazon Aurora の場合、イベントは DB クラスターレベルと DB インスタンスレベルの両方で発生します。次のリソースのイベントカテゴリにサブスクライブできます。

- DB インスタンス
- DB クラスター
- DB クラスタースナップショット
- DB パラメータグループ
- DB セキュリティグループ
- RDS Proxy
- カスタムエンジンバージョン

例えば、特定の DB インスタンスのバックアップカテゴリにサブスクライブした場合、DB インスタンスに影響するバックアップ関連のイベントが発生するたびに通知が送信されます。DB インスタンスの設定変更カテゴリにサブスクライブした場合は、DB インスタンスが変更されると、通知を受け取ります。また、イベント通知サブスクリプションが変更されても、通知を受け取ります。

複数の異なるサブスクリプションを作成することもできます。例えば、すべての DB インスタンスの全イベント通知を受信するサブスクリプションや、DB インスタンスのサブセットに関する重要なイベントのみを含むサブスクリプションを作成することもできます。2 番目のサブスクリプションでは、フィルターで 1 つ以上の DB インスタンスを指定します。

Amazon RDS イベント通知にサブスクライブする基本的な手順は次のとおりです。

Amazon RDS イベント通知にサブスクライブする手順は次のとおりです。

1. Amazon RDS コンソール、AWS CLI、または API を使用して、Amazon RDS イベント通知サブスクリプションを作成します。

   Amazon RDS では、Amazon SNS トピックの ARN を使用して各サブスクリプションを識別します。Amazon RDS コンソールでは、サブスクリプションの作成時に ARN が作成されます。Amazon SNS コンソール、AWS CLI、または Amazon SNS API を使用して ARN を作成します。
2. サブスクリプションの作成で指定したアドレス宛てに、Amazon RDS から承認の E メールまたは SMS メッセージが送信されます。
3. サブスクリプションを確認するには、受信した通知に記載されているリンクを選択します。

4. Amazon RDS コンソールでは、サブスクリプションのステータスで [My Event Subscriptions] (イベントのサブスクリプション) が更新されます。
5. Amazon RDS は、サブスクリプションを作成したときに指定したアドレスへの通知の送信を開始します。

Amazon SNS を使用する際の Identity and Access Management に関する詳細については、Amazon Simple Notification Service デベロッパーガイドの「Identity and access management in Amazon SNS」を参照してください。

DB インスタンスからのイベント通知の処理には、AWS Lambda を使用することができます。詳細については、AWS Lambda デベロッパーガイドの「Amazon RDS で AWS Lambda を使用する」を参照してください。

RDS イベント通知のデリバリー

Amazon RDS は、サブスクリプションを作成するときに指定したアドレスに通知を送信します。通知には、メッセージに関する構造化メタデータを提供する、メッセージ属性を含めることができます。メッセージ属性の詳細については、「Amazon RDS のイベントカテゴリとイベントメッセージ」を参照してください。

イベント通知が配信されるまでに最大 5 分かかります。

> **Important**
>
> Amazon RDS はイベントストリーミングのイベントの順番を保証しません。イベントの順番は変わる場合があります。

Amazon SNS が受信登録している HTTP または HTTPS エンドポイントに通知を送信した場合、エンドポイントに送信される POST メッセージには、JSON ドキュメントを含むメッセージ本文が含まれます。詳細については、Amazon Simple Notification Service デベロッパーガイドの「Amazon SNS メッセージと JSON 形式」を参照してください。

テキストメッセージで通知を送信するように SNS を設定できます。詳細については、Amazon Simple Notification Service デベロッパーガイドの「Mobile text messaging (SMS)」を参照してください。

サブスクリプションを削除せずに通知を無効にするには、Amazon RDS コンソールで [Enabled] (有効) として [No] (いいえ) を選択します。または、AWS CLI や Amazon RDS API を使用して `Enabled` パラメータを`false` に設定します。

Amazon RDS イベント通知の請求

Amazon RDS イベント通知の請求は、Amazon SNS を通じて行われます。使用したイベント通知に対して、Amazon SNS 料金が適用されます。Amazon SNS の請求の詳細については、「Amazon Simple Notification Service 料金表」を参照してください。

Amazon EventBridgei を使用した Aurora イベントの例

次の例は、JSON 形式のさまざまなタイプの Aurora イベントを示しています。JSON 形式でイベントをキャプチャして表示する方法を示すチュートリアルについては、「チュートリアル: Amazon EventBridge を使用して DB インスタンスの状態変化をログに記録する」を参照してください。

トピック

- DB クラスターイベントの例
- DB パラメータグループイベントの例
- DB クラススナップショットイベントの例

DB クラスターイベントの例

以下は、JSON 形式の DB クラスターイベントの例です。イベントは、`my-db-cluster` という名前のクラスターにパッチが適用されたことを示します。イベント ID は `RDS-EVENT-0173` です。

```
{
  "version": "0",
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Cluster Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:cluster:my-db-cluster"
  ],
  "detail": {
    "EventCategories": [
```

```
      "notification"
    ],
    "SourceType": "CLUSTER",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-db-cluster",
    "Date": "2018-10-06T12:26:13.882Z",
    "Message": "Database cluster has been patched",
    "SourceIdentifier": "my-db-cluster",
    "EventID": "RDS-EVENT-0173"
  }
}
```

DB パラメータグループイベントの例

以下は、JSON 形式の DB パラメータグループイベントの例です。イベントは、パラメータグループ time_zone でパラメータ my-db-param-group が更新されたことを示します。イベント ID は RDS-EVENT-0037 です。

```
{
  "version": "0",
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Parameter Group Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:pg:my-db-param-group"
  ],
  "detail": {
    "EventCategories": [
      "configuration change"
    ],
    "SourceType": "DB_PARAM",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:pg:my-db-param-group",
    "Date": "2018-10-06T12:26:13.882Z",
    "Message": "Updated parameter time_zone to UTC with apply method immediate",
    "SourceIdentifier": "my-db-param-group",
    "EventID": "RDS-EVENT-0037"
  }
}
```

DB クラススナップショットイベントの例

以下は、JSON 形式の DB クラススナップショットイベントの例です。イベントは、my-db-cluster-snapshot という名前のスナップショットの作成を示します。イベント ID は RDS-EVENT-0074 です。

```
{
  "version": "0",
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Cluster Snapshot Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:cluster-snapshot:rds:my-db-cluster-snapshot"
  ],
  "detail": {
    "EventCategories": [
      "backup"
    ],
    "SourceType": "CLUSTER_SNAPSHOT",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster-snapshot:rds:my-db-
cluster-snapshot",
    "Date": "2018-10-06T12:26:13.882Z",
    "SourceIdentifier": "my-db-cluster-snapshot",
    "Message": "Creating manual cluster snapshot",
    "EventID": "RDS-EVENT-0074"
  }
}
```

Amazon SNS トピックに通知を発行するアクセス許可を付与する

Amazon Simple Notification Service (Amazon SNS) トピックに通知を発行するための Amazon RDS アクセス許可を付与するには、AWS Identity and Access Management (IAM) ポリシーを送信先トピックに添付します。アクセス許可の詳細については、Amazon Simple Notification Service デベロッパーガイドの「Amazon Simple Notification Service のケース例」を参照してください。

デフォルトで、Amazon SNS トピックには、同じアカウント内のすべての Amazon RDS リソースが通知を発行するのを許可するポリシーがあります。カスタムポリシーをアタッチして、クロスアカウント通知を許可したり、特定のリソースへのアクセスを制限したりできます。

発行先の Amazon SNS トピックにアタッチする IAM ポリシーの例を次に示します。トピックは、指定したプレフィックスと一致する名前を持つ DB インスタンスに制限されます。このポリシーを使用するには、次の値を指定します。

- `Resource` – Amazon SNS トピックの Amazon リソースネーム (ARN)
- `SourceARN` – RDS リソース ARN
- `SourceAccount` – 自分の AWS アカウント ID。

リソースのタイプとその ARN のリストを確認するには、サービス認証リファレンス の「Amazon RDS で定義されるリソース」を参照してください。

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "events.rds.amazonaws.com"
      },
      "Action": [
        "sns:Publish"
      ],
      "Resource": "arn:aws:sns:us-east-1:123456789012:topic_name",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:rds:us-east-1:123456789012:db:prefix-*"
        },
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
```

```
          }
        }
      }
    ]
}
```

Amazon RDS イベント通知にサブスクライブする

サブスクリプションを作成する最も簡単な方法は、RDS コンソールを使用する方法です。CLI または API を使用してイベント通知サブスクリプションを作成する場合、Amazon Simple Notification Service トピックを作成し、Amazon SNS コンソールまたは Amazon SNS API を使用してそのトピックにサブスクライブする必要があります。トピックの Amazon Resource Name (ARN) は、CLI コマンドや API オペレーションを送信するときに使用されるため維持する必要があります。SNS トピックの作成とサブスクライブについては、Amazon Simple Notification Service デベロッパーガイドの「Amazon SNS のスタート方法」を参照してください。

通知を受け取る対象となるソースのタイプ、およびイベントをトリガーする Amazon RDS ソースを指定できます。

ソースタイプ

ソースのタイプ。例えば、[Source type] (ソースタイプ) が [Instances] (インスタンス) の可能性があります。ソースタイプを選択する必要があります。

インクルードする####

イベントを生成している Amazon RDS リソース。例えば、[Select specific instances] (特定のインスタンスを選択) を選択して、[myDBInstance1] を選択します。

次の表は、インクルードする####を指定した場合と、指定しなかった場合の結果について説明しています。

インクルードするリソース	説明	例
指定	RDS は、指定したリソースのみに関するすべてのイベントを通知します。	選択した [Source type] (ソースタイプ) が [Instances] (インスタンス) で、リソースが [myDBInstance1] の場合は、RDS は MyDBInstance1 のみに関するすべてのイベントを通知します。
指定されていません	RDS は、すべての Amazon RDS リソースについて、指定したソースタイプのイベントを通知します。	[Source type] (ソースタイプ) が [Instances] (インスタンス) の場合は、RDS はそのアカウントの

インクルードするリソース	説明	例
		すべてのインスタンス関連イベントを通知します。

デフォルトでは、Amazon SNS トピックのサブスクライバーは、トピックに対して発行されたすべてのメッセージを受信します。メッセージのサブセットのみを受信するには、サブスクライバーはトピックのサブスクリプションにフィルターポリシーを割り当てる必要があります。詳細については、「Amazon SNS 開発者ガイド」の「Amazon SNS メッセージのフィルター処理」を参照してください。

コンソール

RDS イベント通知にサブスクライブするには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[イベントサブスクリプション] を選択します。
3. [イベントサブスクリプション] ページで、[イベントサブスクリプションの作成] を選択します。
4. サブスクリプションの詳細を次のように入力します。

   a. [Name] (名前) に、イベント通知サブスクリプションの名前を入力します。

   b. [Send notifications to] (通知の送信先) については、次のいずれかを実行します。

      - [New email topic] (新しい E メールトピック) を選択します。E メールトピックの名前と受信者のリストを入力します。イベントのサブスクリプションは、プライマリアカウントの連絡先と同じメールアドレスに設定することをお勧めします。推奨事項、サービスイベント、および個人の健康メッセージは、さまざまなチャネルを使用して送信されます。同じメールアドレスでサブスクリプションすると、すべてのメッセージが 1 か所にまとめられます。
      - [Amazon Resource Name (ARN)] (Amazon リソースネーム (ARN)) を選択します。次に、Amazon SNS トピックで既存の Amazon SNS ARN を選択します。

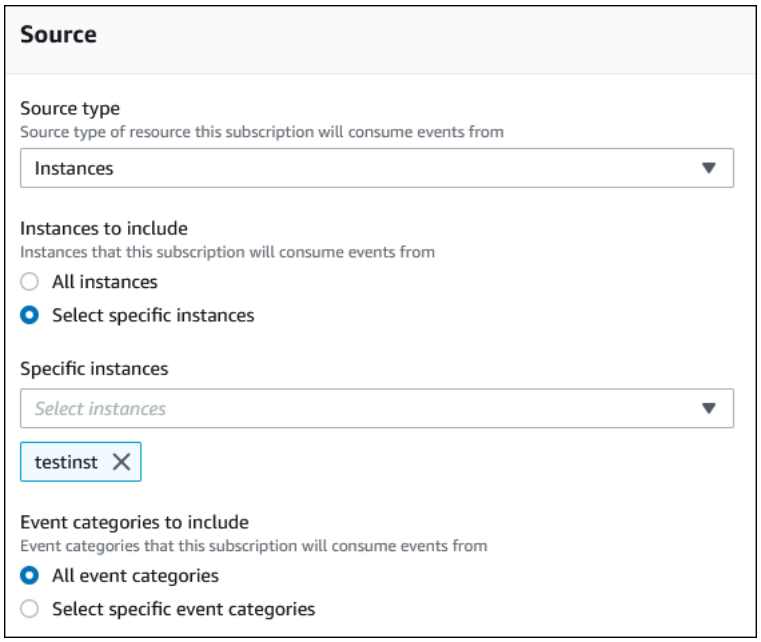
        サーバー側の暗号化 (SSE) で有効になっているトピックを使用する場合は、Amazon RDS に、AWS KMS key にアクセスするのに必要な許可を付与します。詳細について

は、Amazon Simple Notification Service デベロッパーガイドの「AWS サービスと暗号化されたトピックのイベントソース間の互換性を保つ」を参照してください。

c. [ソースタイプ] で、ソースタイプを選択します。例えば、[Cluster] (クラスター) または [Cluster snapshots] (クラスタースナップショット) を選択します。

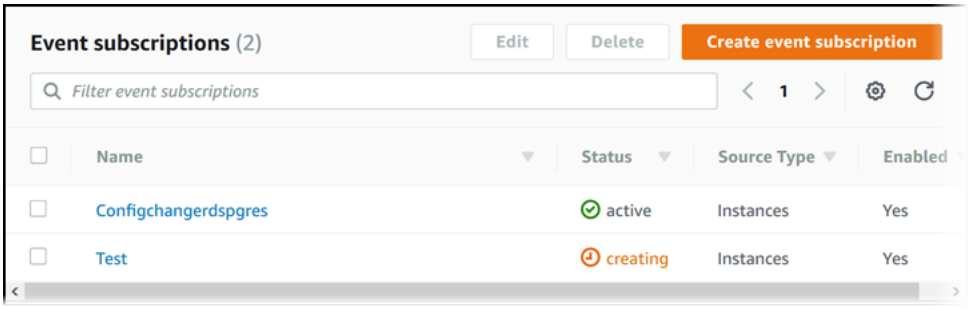
d. イベントの通知を受け取るイベントのカテゴリとリソースを選択します。

次の例では、`testinst` という名前の DB インスタンスについて、イベント通知を設定します。



e. [作成] を選択します。

Amazon RDS コンソールでは、サブスクリプションが作成されることが示されます。



AWS CLI

RDS イベント通知を受信するには、AWS CLI `create-event-subscription` コマンドを使用します。以下の必須パラメータを含めます。

- `--subscription-name`
- `--sns-topic-arn`

Example

Linux、macOS、Unix の場合:

```
aws rds create-event-subscription \
    --subscription-name myeventsubscription \
    --sns-topic-arn arn:aws:sns:us-east-1:123456789012:myawsuser-RDS \
    --enabled
```

Windows の場合:

```
aws rds create-event-subscription ^
    --subscription-name myeventsubscription ^
    --sns-topic-arn arn:aws:sns:us-east-1:123456789012:myawsuser-RDS ^
    --enabled
```

API

Amazon RDS イベント通知を受信するには、Amazon RDS API 関数 `CreateEventSubscription` を呼び出します。以下の必須パラメータを含めます。

- `SubscriptionName`
- `SnsTopicArn`

Amazon RDS イベント通知タグと属性

Amazon RDS が Amazon Simple Notification Service (SNS) または Amazon EventBridge にイベント通知を送信するとき、通知にはメッセージ属性とイベントタグが含まれます。RDS はメッセージと共にメッセージ属性を個別に送信しますが、イベントタグはメッセージの本文に含まれています。メッセージ属性と Amazon RDS タグを使用して、リソースにメタデータを追加します。これらのタグは、DB インスタンス、Aurora クラスターなどに関する独自の表記で変更できます。Amazon RDS リソースのタグ付けの詳細については、「Amazon RDS リソースのタグ付け」を参照してください。

デフォルトでは、Amazon SNS と Amazon EventBridge は、送信されたすべてのメッセージを受信します。SNS と EventBridge では、メッセージをフィルタリングして、メールやテキストメッセージ、または HTTP エンドポイントの呼び出しなど、希望する通信モードに通知を送信できます。

> **Note**
>
> メールまたはテキストメッセージで送信される通知には、イベントタグはありません。

次の表では、トピックサブスクライバーに送信された RDS イベントのメッセージ属性を示します。

Amazon RDS イベント属性	説明
EventID	RDS イベントメッセージの識別子。例えば、RDS-EVENT-0006。
リソース	イベントを発行するリソースの ARN 識別子 (例: `arn:aws:rds:ap-southeast-2:123456789012:db:database-1`)。

RDS タグは、サービスイベントの影響を受けたリソースに関するデータを提供します。通知が SNS または EventBridge に送信されると、RDS はメッセージ本文にタグの現在の状態を追加します。

SNS メッセージ属性のフィルター処理の詳細については、Amazon Simple Notification Service デベロッパーガイドの「Amazon SNS メッセージフィルター処理」を参照してください。

EventBrodge のイベントタグのフィルター処理の詳細については、Amazon EventBridge ユーザーガイドの「Amazon EventBridge イベントパターンでのコンテンツのフィルタリング」を参照してください。

SNS のペイロードベースタグのフィルター処理の詳細については、「https://aws.amazon.com/blogs/compute/introducing-payload-based-message-filtering-for-amazon-sns/」を参照してください

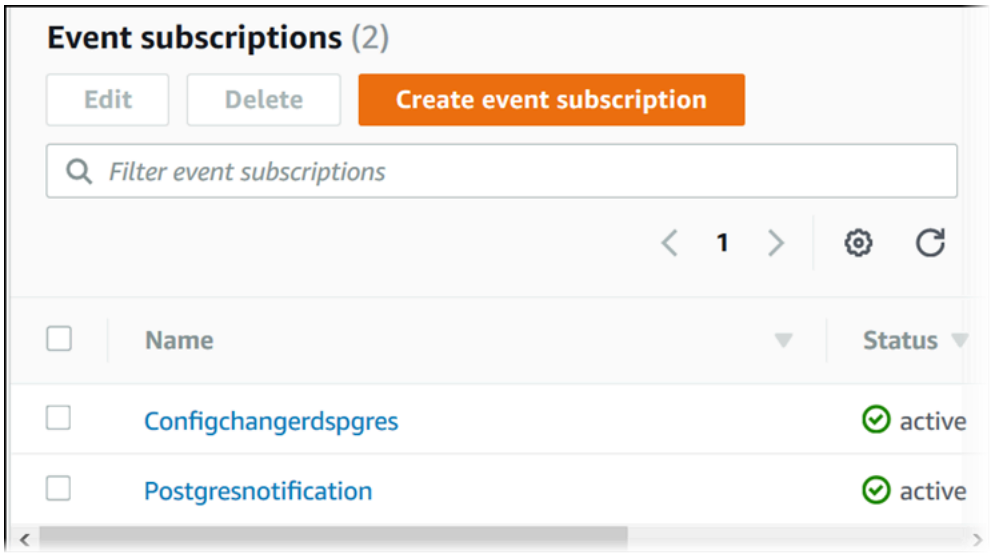
Amazon RDS DB イベント通知サブスクリプションのリスト化

現在の Amazon RDS イベント通知サブスクリプションのリストを表示できます。

コンソール

現在の Amazon RDS イベント通知サブスクリプションのリストを表示するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[イベントサブスクリプション] を選択します。[イベントサブスクリプション] ペインにイベント通知サブスクリプションが一覧表示されます。



AWS CLI

現在の Amazon RDS イベント通知サブスクリプションを一覧表示するには、AWS CLI の `describe-event-subscriptions` コマンドを使用します。

Example

次の例は、すべてのイベントサブスクリプションを表しています。

```
aws rds describe-event-subscriptions
```

次の例は、`myfirsteventsubscription` を表しています。

```
aws rds describe-event-subscriptions --subscription-name myfirsteventsubscription
```

API

Amazon RDS イベント通知に対する現在のサブスクリプションを一覧表示するには、Amazon RDS API の DescribeEventSubscriptions アクションを呼び出します。

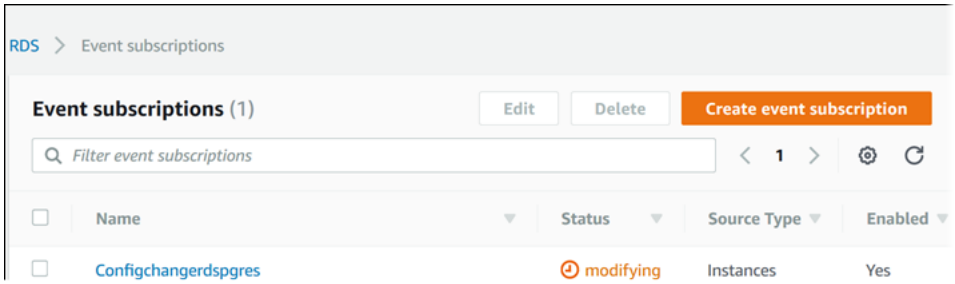
Amazon RDS イベント通知サブスクリプションを変更する

サブスクリプションを作成すると、サブスクリプション名、ソース識別子、カテゴリ、トピック ARN を変更できます。

コンソール

Amazon RDS イベント通知サブスクリプションを変更するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[イベントサブスクリプション] を選択します。
3. [イベントサブスクリプション] ペインで、変更するサブスクリプションを選択し、[編集] をクリックします。
4. [ターゲット] セクションまたは [ソース] セクションのいずれかでサブスクリプションを変更します。
5. [編集] を選択します。Amazon RDS コンソールでは、サブスクリプションが変更されることが示されます。



AWS CLI

Amazon RDS イベント通知サブスクリプションを変更するには、AWS CLI の `modify-event-subscription` コマンドを使用します。以下の必須パラメータを含めます。

- `--subscription-name`

Example

次のコードで `myeventsubscription` を有効にします。

Linux、macOS、Unix の場合:

```
aws rds modify-event-subscription \
    --subscription-name myeventsubscription \
    --enabled
```

Windows の場合:

```
aws rds modify-event-subscription ^
    --subscription-name myeventsubscription ^
    --enabled
```

API

Amazon RDS イベントを変更するには、Amazon RDS API オペレーション `ModifyEventSubscription` を呼び出します。以下の必須パラメータを含めます。

- `SubscriptionName`

Amazon RDS イベント通知サブスクリプションへのソース識別子の追加

既存のサブスクリプションにソース識別子 (イベントを生成する Amazon RDS ソース) を追加できます。

コンソール

Amazon RDS コンソールを使用し、サブスクリプションを変更するときに選択または選択解除することで、ソース識別子を簡単に追加または削除できます。詳細については、「Amazon RDS イベント通知サブスクリプションを変更する」を参照してください。

AWS CLI

Amazon RDS イベント通知サブスクリプションにソース識別子を追加するには、AWS CLI の `add-source-identifier-to-subscription` コマンドを使用します。以下の必須パラメータを含めます。

- `--subscription-name`
- `--source-identifier`

Example

次の例は、ソース識別子 `mysqldb` を `myrdseventsubscription` サブスクリプションに追加します。

Linux、macOS、Unix の場合:

```
aws rds add-source-identifier-to-subscription \
    --subscription-name myrdseventsubscription \
    --source-identifier mysqldb
```

Windows の場合:

```
aws rds add-source-identifier-to-subscription ^
    --subscription-name myrdseventsubscription ^
    --source-identifier mysqldb
```

API

Amazon RDS イベント通知サブスクリプションにソース識別子を追加するには、Amazon RDS API `AddSourceIdentifierToSubscription` を呼び出します。以下の必須パラメータを含めます。

- SubscriptionName
- SourceIdentifier

Amazon RDS イベント通知サブスクリプションからのソース識別子の削除

そのソースのイベントの通知を今後は受け取らない場合、サブスクリプションからソース識別子 (イベントを生成する Amazon RDS ソース) を削除できます。

コンソール

Amazon RDS コンソールを使用し、サブスクリプションを変更するときに選択または選択解除することで、ソース識別子を簡単に追加または削除できます。詳細については、「Amazon RDS イベント通知サブスクリプションを変更する」を参照してください。

AWS CLI

Amazon RDS イベント通知サブスクリプションからソース識別子を削除するには、AWS CLI の `remove-source-identifier-from-subscription` コマンドを使用します。以下の必須パラメータを含めます。

- `--subscription-name`
- `--source-identifier`

Example

次の例は、ソース識別子 `mysqldb` を `myrdseventsubscription` サブスクリプションから削除します。

Linux、macOS、Unix の場合:

```
aws rds remove-source-identifier-from-subscription \
    --subscription-name myrdseventsubscription \
    --source-identifier mysqldb
```

Windows の場合:

```
aws rds remove-source-identifier-from-subscription ^
    --subscription-name myrdseventsubscription ^
    --source-identifier mysqldb
```

API

Amazon RDS イベント通知サブスクリプションからソース識別子を削除するには、Amazon RDS API RemoveSourceIdentifierFromSubscription コマンドを使用します。以下の必須パラメータを含めます。

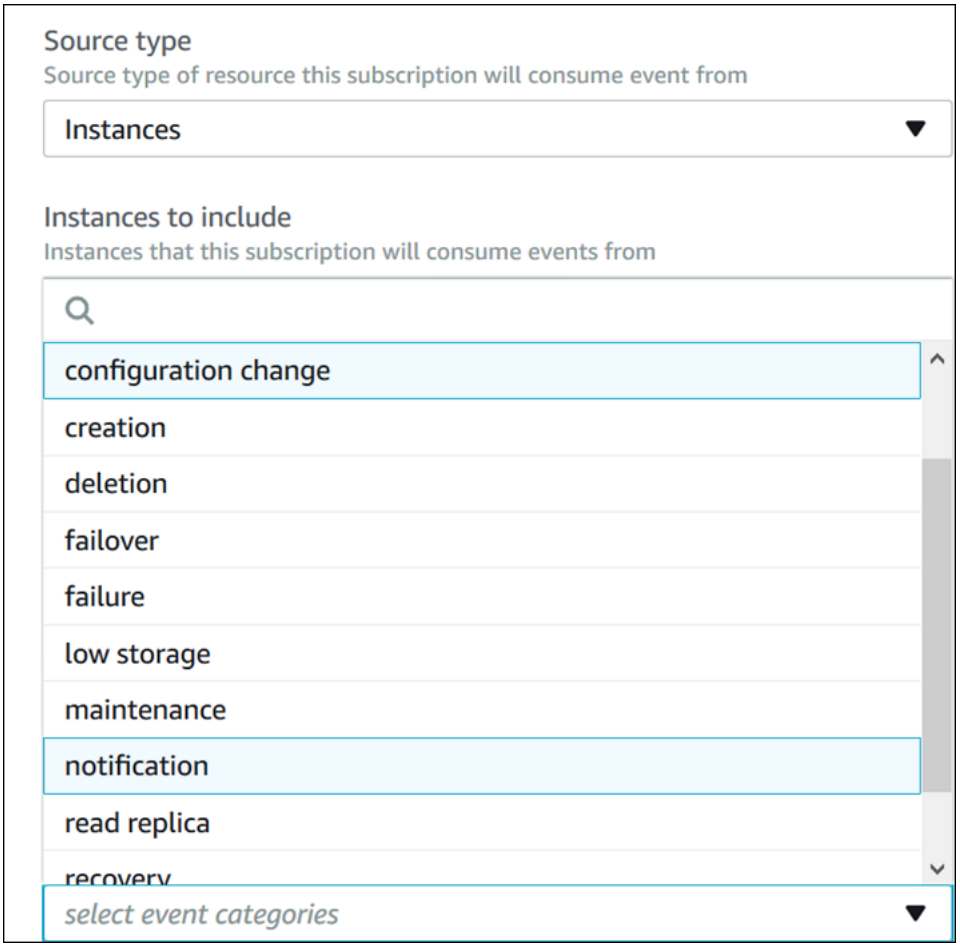
- SubscriptionName
- SourceIdentifier

Amazon RDS イベント通知カテゴリのリスト化

リソースタイプのすべてのイベントはカテゴリに分類されます。使用できるカテゴリのリストを表示するには、次の手順を実行します。

コンソール

イベント通知サブスクリプションを作成または変更するとき、Amazon RDS コンソールにイベントカテゴリが表示されます。詳細については、「Amazon RDS イベント通知サブスクリプションを変更する」を参照してください。



AWS CLI

Amazon RDS イベント通知カテゴリを一覧表示するには、AWS CLI の `describe-event-categories` コマンドを使用します。このコマンドには必須パラメータはありません。

Example

```
aws rds describe-event-categories
```

API

Amazon RDS イベント通知カテゴリを一覧表示するには、Amazon RDS API [DescribeEventCategories](#) コマンドを使用します。このコマンドには必須パラメータはありません。

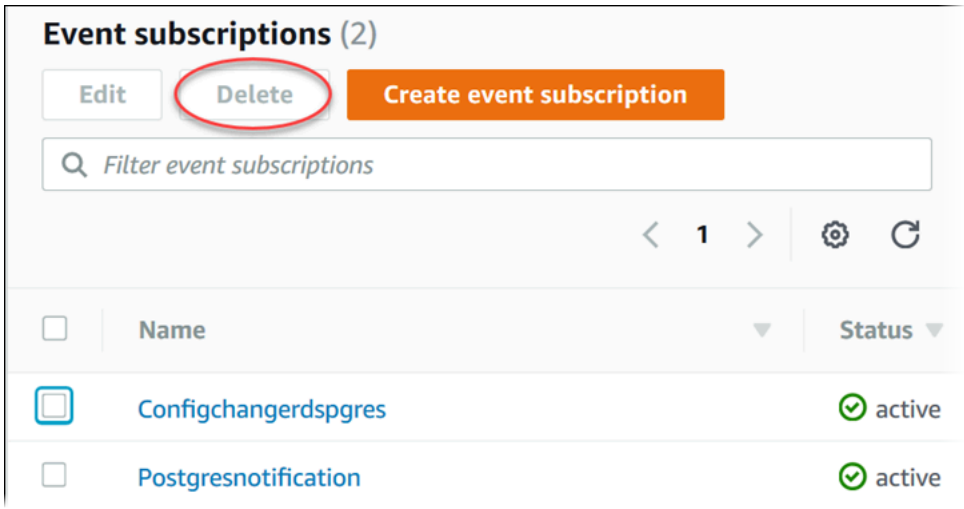
Amazon RDS イベント通知サブスクリプションの削除

不要になったサブスクリプションは削除できます。トピックへのすべてのサブスクライバは、サブスクリプションにより指定されたイベント通知を受け取らなくなります。

コンソール

Amazon RDS イベント通知サブスクリプションを削除するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[DB イベントサブスクリプション] を選択します。
3. [My DB Event Subscriptions] (自分の DB イベントサブスクリプション) ペインで、削除するサブスクリプションを選択します。
4. [削除] を選択します。
5. サブスクリプションを削除中であることが Amazon RDS コンソールに表示されます。



AWS CLI

Amazon RDS イベント通知サブスクリプションを削除するには、AWS CLI の `delete-event-subscription` コマンドを使用します。以下の必須パラメータを含めます。

- `--subscription-name`

Example

以下の例では、サブスクリプション myrdssubscription を削除します。

```
aws rds delete-event-subscription --subscription-name myrdssubscription
```

API

Amazon RDS イベント通知サブスクリプションを削除するには、RDS API DeleteEventSubscription コマンドを使用します。以下の必須パラメータを含めます。

- SubscriptionName

Amazon Aurora イベントでトリガーするルールの作成

Amazon EventBridge を使用すると、AWS のサービスを自動化して、アプリケーションの可用性の問題やリソースの変更などのシステムイベントに対応できます。

トピック

チュートリアル: Amazon EventBridge を使用して DB インスタンスの状態変化を口グに記録する

このチュートリアルでは、 インスタンスの状態変化を口グに記録する AWS Lambda 関数を作成します。次に、既存の RDS DB インスタンスの状態変化があったときに関数を実行するルールを作成します。このチュートリアルでは、一時的にシャットダウンできる小規模なテストインスタンスが実行されていることを前提としています。

> **Important**
>
> 実行中の本番 DB インスタンスに対して、このチュートリアルを実行しないでください。

トピック

ステップ 1: AWS Lambda 関数を作成する

状態変更イベントの口グを記録する Lambda 関数を作成します。ルールを作成するときに、この関数を指定します。

Lambda 関数を作成するには

1. AWS Lambda コンソールを https://console.aws.amazon.com/lambda/ で開きます。
2. Lambda を初めて使用する場合は、ウェルカムページを参照してください。[今すぐ始める] を選択します。それ以外の場合は、[関数の作成] を選択します。
3. [Author from scratch] を選択します。

4. [関数の作成] ページで、次の操作を実行します。

   a. Lambda 関数の名前と説明を入力します。例えば、関数名を **RDSInstanceStateChange** とします。

   b. [Runtime] (ランタイム) では、[Node.js 16x] を選択します。

   c. [Architecture] (アーキテクチャ) では、[x86_64] を選択します。

   d. [Execution role] (実行ロール) では、次のいずれかを実行します。

      - [基本的な Lambda アクセス権限で新しいロールを作成] を選択します。
      - [Existing role] (既存のロール) では、[Use an existing role] (既存のロールを使用する) を選択します。使用するロールを選択します。

   e. [Create function] (関数の作成) を選択します。

5. [RDSInstanceStateChange] ページで、次の操作を行います。

   a. [コードソース] で、[index.js] を選択します。

   b. で、[index.js] ウィンドウで、既存のコードを削除します。

   c. 次のコードを入力します。

```
console.log('Loading function');

exports.handler = async (event, context) => {
    console.log('Received event:', JSON.stringify(event));
};
```

   d. [デプロイ] を選択します。

ステップ 2: ルールを作成する

Amazon RDS インスタンスを起動するたびに Lambda 関数を実行するルールを作成します。

EventBridge ルールを作成するには

1. Amazon EventBridge コンソール (https://console.aws.amazon.com/events/) を開きます。
2. ナビゲーションペインで Rules] (ルール) を選択します。
3. ルールの作成 を選択します。
4. ルールの名前と説明を入力します。例えば、「**RDSInstanceStateChangeRule**」と入力します。

5. [Rule with an event pattern] (イベントパターンを持つルール) を選択してから、[Next] (次へ) を選択します。
6. [Event source] (イベントソース) で、[AWS events or EventBridge partner events] (イベントまたは EventBridge パートナーイベント) を選択します。
7. 下にスクロールして、[Event pattern] (イベントパターン) セクションを展開します。
8. イベントソース で AWS のサービス を選択します。
9. [AWS service] (AWS サービス) では [Relational Database Service (RDS)] (リレーショナルデータベースサービス (RDS)) を選択します。
10. [イベントタイプ] で、[RDS DB インスタンスイベント] を選択します。
11. デフォルトのイベントパターンのままにします。次いで、[次へ] を選択します。
12. ターゲットタイプ] では、AWSサービス] を選択します。
13. [Select a target] (ターゲットを選択) では、[Lambda function] (Lambda 関数) を選択します。
14. [Function] (関数) では、作成した Lambda 関数を選択します。次いで、[次へ] を選択します。
15. [Configure Tags] (タグの設定) で、[Next] (次へ) を選択します。
16. ルール内の手順を確認します。次に、[Create rule] (ルールの作成) を選択します。

ステップ 3: ルールをテストする

ルールをテストするには、RDS DB インスタンスをシャットダウンします。インスタンスのシャットダウンの数分後に、Lambda 関数が呼び出されたことが確認できます。

DB インスタンスを停止してルールをテストするには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. RDS DB インスタンスを停止します。
3. Amazon EventBridge コンソール (https://console.aws.amazon.com/events/) を開きます。
4. ナビゲーションペインで、[ルール] を選択し、作成したルールの名前を選択します。
5. [ルールの詳細] で、[モニタリング] を選択します。

   Amazon CloudWatch コンソールにリダイレクトされます。リダイレクトされない場合は、[CloudWatch でメトリクスを表示] をクリックします。
6. [すべてのメトリクス] で、作成したルールの名前を選択します。

   グラフには、ルールが呼び出されたことが示されます。

7. ナビゲーションペインで、[Log groups] (ロググループ) を選択します。
8. Lambda 関数 (/aws/lambda/***function-name***) のロググループの名前を選択します。
9. 起動したインスタンスの関数によって提供されるデータを表示するログのストリーミング名を選択します。次のような受信イベントが表示されます。

```
{
    "version": "0",
    "id": "12a345b6-78c9-01d2-34e5-123f4ghi5j6k",
    "detail-type": "RDS DB Instance Event",
    "source": "aws.rds",
    "account": "111111111111",
    "time": "2021-03-19T19:34:09Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:rds:us-east-1:111111111111:db:testdb"
    ],
    "detail": {
        "EventCategories": [
            "notification"
        ],
        "SourceType": "DB_INSTANCE",
        "SourceArn": "arn:aws:rds:us-east-1:111111111111:db:testdb",
        "Date": "2021-03-19T19:34:09.293Z",
        "Message": "DB instance stopped",
        "SourceIdentifier": "testdb",
        "EventID": "RDS-EVENT-0087"
    }
}
```

JSON 形式の RDS イベントの例については、「Aurora のイベントの概要」を参照してください。

10. (オプション) 終了したら、Amazon RDS コンソールを開き、停止したインスタンスをスタートできます。

Amazon RDS のイベントカテゴリとイベントメッセージ

Amazon RDS では、多数のイベントがカテゴリ分けされて生成され、これらには Amazon RDS コンソール、AWS CLI、または API を使用してのサブスクライブが可能です。

トピック

DB クラスターイベント

次の表は、DB クラスターがソースタイプである場合のイベントカテゴリとイベントのリストを示します。

> **Note**
>
> DB クラスターイベントタイプの Aurora Serverless にイベントカテゴリが存在しません。Aurora Serverless イベントの範囲は、RDS-EVENT-0141 から RDS-EVENT-0149 までです。

カテゴリ	RDS イベント ID	メッセージ	メモ
設定変更	RDS-EVENT-0016	マスター認証情報をリセットします。	
設定変更	RDS-EVENT-0179	Database Activity Streams は、データベースクラスターでスタートされます。	詳細については、「データベースアクティビティストリームを使用した Amazon Aurora のモニタリング」を参照してください。

カテゴリ	RDS イベント ID	メッセージ	メモ
設定変更	RDS-EVENT-0180	Database Activity Streams は、データベースクラスターで停止されます。	詳細については、「データベースアクティビティストリームを使用した Amazon Aurora のモニタリング」を参照してください。
作成	RDS-EVENT-0170	DB クラスターが作成されました。	
削除	RDS-EVENT-0171	DB クラスターが削除されました。	
フェイルオーバー	RDS-EVENT-0069	クラスターフェイルオーバーが失敗しました。クラスターインスタンスの状態を確認して、もう一度試してください。	
フェイルオーバー	RDS-EVENT-0070	以前のプライマリを再度プロモートします: *name*。	
フェイルオーバー	RDS-EVENT-0071	インスタンスへのフェイルオーバーが完了しました: *name*。	
フェイルオーバー	RDS-EVENT-0072	DB インスタンスへの同じ AZ フェイルオーバーを開始しました: *name*。	
フェイルオーバー	RDS-EVENT-0073	DB インスタンスへのクロスAZ フェイルオーバーを開始しました: *name*。	

カテゴリ	RDS イベント ID	メッセージ	メモ
失敗	RDS-EVENT-0083	Amazon RDS は、DB クラスター*name* の Amazon S3 バケットにアクセスするための認証情報を作成できませんでした。これは、S3 スナップショット取り込み IAM ロールがアカウントで正しく設定されていないか、指定した Amazon S3 バケットが見つからないことが原因です。詳細については、Amazon RDS ドキュメントのトラブルシューティングセクションを参照してください。	詳細については、Percona XtraBackup と Amazon S3 を使用した MySQL からの物理的な移行「」を参照してください。
失敗	RDS-EVENT-0143	DB クラスターが *units* から *units* へのスケーリングに失敗したのは、次の理由によるものです: *reason*。	Aurora Serverless DB クラスターのスケーリングが失敗しました。
失敗	RDS-EVENT-0354	リソースに互換性がないため、DB クラスターを作成できません。*#####*。	*#####*には、障害に関する詳細が含まれます。
失敗	RDS-EVENT-0355	リソース制限が不十分なため、DB クラスターを作成できません。*#####*。	*#####*には、障害に関する詳細が含まれます。

カテゴリ	RDS イベント ID	メッセージ	メモ
グローバルフェイルオーバー	RDS-EVENT-0181	リージョン *name* の DB クラスター *name* へのスイッチオーバーが開始されました。	このイベントは、スイッチオーバーオペレーション向けです (以前は「マネージドプランニングフェイルオーバー」と呼ばれていました)。 DB クラスターで他のオペレーションが実行されているため、プロセスが遅延する可能性があります。
グローバルフェイルオーバー	RDS-EVENT-0182	リージョン *name* の古いプライマリ DB クラスター *name* が正常にシャットダウンされました。	このイベントは、スイッチオーバーオペレーション向けです (以前は「マネージドプランニングフェイルオーバー」と呼ばれていました)。 グローバルデータベースの古いプライマリインスタンスは書き込みを受け付けていません。すべてのボリュームが同期されます。

カテゴリ	RDS イベント ID	メッセージ	メモ
グローバルフェイルオーバー	RDS-EVENT-0183	グローバルクラスタメンバー間のデータ同期を待っています。現在、プライマリ DB クラスターに遅れています：*reason*。	このイベントは、スイッチオーバーオペレーション向けです (以前は「マネージドプランニングフェイルオーバー」と呼ばれていました)。 グローバルデータベースフェイルオーバーの同期フェーズで、レプリケーションラグが発生しています。
グローバルフェイルオーバー	RDS-EVENT-0184	リージョン *name* の新しいプライマリ DB クラスター *name* が正常に昇格されました。	このイベントは、スイッチオーバーオペレーション向けです (以前は「マネージドプランニングフェイルオーバー」と呼ばれていました)。 グローバルデータベースのボリュームトポロジーが、新しいプライマリボリュームで再確立されます。

カテゴリ	RDS イベント ID	メッセージ	メモ
グローバルフェイルオーバー	RDS-EVENT-0185	リージョン *name* の DB クラスター *name* へのスイッチオーバーが終了しました。	このイベントは、スイッチオーバーオペレーション向けです (以前は「マネージドプランニングフェイルオーバー」と呼ばれていました)。 プライマリ DB クラスターのグローバルスイッチオーバーが終了しました。フェイルオーバーの完了後、レプリカがオンラインになるまでに時間がかかることがあります。
グローバルフェイルオーバー	RDS-EVENT-0186	リージョン *name* の DB クラスター *name* へのグローバルスイッチオーバーがキャンセルされました。	このイベントは、スイッチオーバーオペレーション向けです (以前は「マネージドプランニングフェイルオーバー」と呼ばれていました)。
グローバルフェイルオーバー	RDS-EVENT-0187	リージョン *name* の DB クラスター *name* へのスイッチオーバーに失敗しました。	このイベントは、スイッチオーバーオペレーション向けです (以前は「マネージドプランニングフェイルオーバー」と呼ばれていました)。
グローバルフェイルオーバー	RDS-EVENT-0238	リージョン *name* の DB クラスター *name* へのグローバルフェイルオーバーが完了しました。	

カテゴリ	RDS イベント ID	メッセージ	メモ
グローバルフェイルオーバー	RDS-EVENT-0239	リージョン *name* の DB クラスター *name* へのグローバルフェイルオーバーが失敗しました。	
グローバルフェイルオーバー	RDS-EVENT-0240	グローバルフェイルオーバー後、リージョン *name* の DB クラスターのメンバー *name* の再同期を開始しました。	
グローバルフェイルオーバー	RDS-EVENT-0241	グローバルフェイルオーバー後、リージョン *name* の DB クラスターのメンバー *name* の再同期を終了しました。	
メンテナンス	RDS-EVENT-0156	DB クラスターで DB エンジンのマイナーバージョンをアップグレードできます。	
メンテナンス	RDS-EVENT-0173	データベースクラスターエンジンのバージョンがアップグレードされました。	DB クラスターのパッチ適用が完了しました。
メンテナンス	RDS-EVENT-0176	データベースクラスターエンジンのメジャーバージョンがアップグレードされました。	
メンテナンス	RDS-EVENT-0286	データベースクラスターエンジンのバージョンのアップグレードが開始しました。	

カテゴリ	RDS イベント ID	メッセージ	メモ
メンテナンス	RDS-EVENT-0287	オペレーティングシステムのアップグレード要件が検出されました。	
メンテナンス	RDS-EVENT-0288	クラスターオペレーティングシステムのアップグレードが開始されます。	
メンテナンス	RDS-EVENT-0289	クラスターオペレーティングシステムのアップグレードが完了しました。	
メンテナンス	RDS-EVENT-0290	データベースクラスターにパッチが適用されました: ソースバージョン *version_number* => *new_version_number* 。	
通知	RDS-EVENT-0076	*name* から *name* の移行に失敗しました。理由: *reason*。	Aurora DB クラスターへの移行に失敗しました。
通知	RDS-EVENT-0077	*name*.*name* を InnoDB に変換できませんでした。理由: *reason*。	ソースデータベースから InnoDB にテーブルを変換しようとしましたが、Aurora DB クラスターへの移行中に失敗しました。

カテゴリ	RDS イベント ID	メッセージ	メモ
通知	RDS-EVENT-0085	インスタンス *name* のステータスが *name* であるため、DB クラスター *name* をアップグレードできません。問題を解決するか、インスタンスを削除して再試行します。	Aurora DB クラスターにパッチを適用しようとしたときにエラーが発生しました。インスタンスのステータスを確認し、問題を解決してから、もう一度お試しください。詳細については、「Amazon Aurora DB クラスターのメンテナンス」を参照してください。
通知	RDS-EVENT-0141	次の理由により DB クラスターを *units* から *units* にスケーリングできませんでした: *reason*。	Aurora Serverless DB クラスターのスケーリングがスタートされました。
通知	RDS-EVENT-0142	DB クラスターは *units* から *units* にスケーリングされました。	Aurora Serverless DB クラスターのスケーリングが完了しました。
通知	RDS-EVENT-0144	DB クラスターが一時停止しています。	Aurora Serverless DB クラスターで自動一時停止が開始されました。
通知	RDS-EVENT-0145	DB クラスターが一時停止しています。	Aurora Serverless DB クラスターが一時停止されました。
通知	RDS-EVENT-0146	DB クラスターの一時停止がキャンセルされました。	Aurora Serverless DB クラスターの一時停止がキャンセルされました。
通知	RDS-EVENT-0147	DB クラスターが再開されます。	Aurora Serverless DB クラスターの再開オペレーションが開始されました。

カテゴリ	RDS イベント ID	メッセージ	メモ
通知	RDS-EVENT-0148	DB クラスターが再開されました。	Aurora Serverless DB クラスターの再開オペレーションが完了しました。
通知	RDS-EVENT-0149	DB クラスターが *units* から *units* へとスケーリングされましたが、次の理由からスケーリングがシームレスに行われませんでした: *reason*。	Aurora Serverless DB クラスターの強制オプションにより、シームレスなスケーリングが完了しました。必要に応じて接続が中断された可能性があります。
通知	RDS-EVENT-0150	DB クラスターが停止しました。	
通知	RDS-EVENT-0151	DB クラスターが開始しました。	
通知	RDS-EVENT-0152	DB クラスターの停止が失敗しました。	
通知	RDS-EVENT-0153	停止中の最大許容時間を超えたため、DB クラスターがスタートされています。	
通知	RDS-EVENT-0172	クラスターの名前が *name* から *name* に変更されました。	
通知	RDS-EVENT-0234	エクスポートタスクが失敗しました。	DB クラスターのエクスポートタスクに失敗しました。
通知	RDS-EVENT-0235	エクスポートタスクがキャンセルされました。	DB クラスターのエクスポートタスクがキャンセルされました。

カテゴリ	RDS イベント ID	メッセージ	メモ
通知	RDS-EVENT-0236	エクスポートタスクが完了しました。	DB クラスターのエクスポートタスクが完了しました。

DB インスタンスイベント

次の表は、DB インスタンスがソースタイプである場合のイベントカテゴリとイベントを示しています。

カテゴリ	RDS イベント ID	メッセージ	メモ
可用性	RDS-EVENT-0004	DB インスタンスがシャットダウンしました。	
可用性	RDS-EVENT-0006	インスタンスが再起動しました。	
可用性	RDS-EVENT-0022	mysql の再起動中にエラーが発生しました: *message*。	Aurora MySQLまたは RDS for MariaDB の再起動中にエラーが発生しました。
バックトラック	RDS-EVENT-0131	実際のバックトラックウィンドウは、指定したターゲットバックトラックウィンドウよりも小さくなります。ターゲットのバックトラックウィンドウの時間数を減らすことを検討してください。	バックトラックの詳細については、「Aurora DB クラスターのバックトラック」を参照してください。
バックトラック	RDS-EVENT-0132	実際のバックトラックウィンドウは、ターゲットのバックトラックウィンドウと同じになります。	

カテゴリ	RDS イベント ID	メッセージ	メモ
設定変更	RDS-EVENT-0011	DBParameterGroup の*##*を使用するように更新されました。	
設定変更	RDS-EVENT-0012	変更をデータベースインスタンスクラスに適用しています。	
設定変更	RDS-EVENT-0014	インスタンスクラスへの変更適用が終了しました。	
設定変更	RDS-EVENT-0017	割り当てられたストレージへの変更適用が終了しました。	
設定変更	RDS-EVENT-0025	マルチ AZ DB インスタンスに変換するための変更適用が終了しました。	
設定変更	RDS-EVENT-0029	標準 (シングル AZ) DB インスタンスに変換するための変更適用が終了しました。	
設定変更	RDS-EVENT-0033	マスターユーザー名に一致するユーザーが *number* 人います。特定のホストに関連付けられていないユーザーのみがリセットされます。	
設定変更	RDS-EVENT-0067	パスワードをリセットできません。エラー情報: *message*。	

カテゴリ	RDS イベント ID	メッセージ	メモ
設定変更	RDS-EVENT-0078	モニタリング間隔が *number* に変更されました。	拡張モニタリングの設定が変更されました。
設定変更	RDS-EVENT-0092	DB パラメータグループの更新が完了しました。	
作成	RDS-EVENT-0005	DB インスタンスが作成されました。	
削除	RDS-EVENT-0003	DB インスタンスが削除されました。	
失敗	RDS-EVENT-0035	データベースインスタンスが *state* になりました。*message*。	DB インスタンスに無効なパラメータがあります。例えば、メモリ関連のパラメータがこのインスタンスクラスには高すぎる値に設定されていて、DB インスタンスをスタートできなかった場合、メモリのパラメータを変更して、DB インスタンスを再起動してください。
失敗	RDS-EVENT-0036	データベースインスタンスが *state* です。*message*。	DB インスタンスが互換性のないネットワーク上にあります。指定したサブネット ID の一部は無効であるか、存在しません。

カテゴリ	RDS イベント ID	メッセージ	メモ
失敗	RDS-EVENT-0079	Amazon RDS は拡張モニタリング用の認証情報を作成できなかったため、この機能は無効になっています。これは、アカウントに rds-monitoring-role が存在せず、正しく設定されていないことが原因と考えられます。詳細については、Amazon RDS ドキュメントのトラブルシューティングセクションを参照してください。	拡張モニタリングは、拡張モニタリング IAM ロールを使用してのみ有効にすることができます。IAM ロールの作成の詳細については、「Amazon RDS 拡張モニタリング用の IAM ロールを作成するには」を参照してください。
失敗	RDS-EVENT-0080	Amazon RDS がインスタンス:*name* に拡張モニタリングを設定できなかったため、この機能は無効になっています。これは、アカウントに rds-monitoring-role が存在せず、正しく設定されていないことが原因と考えられます。詳細については、Amazon RDS ドキュメントのトラブルシューティングセクションを参照してください。	拡張モニタリングは、設定変更中のエラーのため無効になりました。拡張モニタリング IAM ロールが正しく設定されていない可能性があります。拡張モニタリング IAM ロールの作成については、「Amazon RDS 拡張モニタリング用の IAM ロールを作成するには」を参照してください。

カテゴリ	RDS イベント ID	メッセージ	メモ
失敗	RDS-EVENT-0082	Amazon RDS は、DB インスタンス *name* の Amazon S3 バケットにアクセスするための認証情報を作成できませんでした。これは、S3 スナップショット取り込み IAM ロールがアカウントで正しく設定されていないか、指定した Amazon S3 バケットが見つからないことが原因です。詳細については、Amazon RDS ドキュメントのトラブルシューティングセクションを参照してください。	Aurora は、Amazon S3 バケットからバックアップデータをコピーできませんでした。Aurora から Amazon S3 バケットにアクセスするためのアクセス権限が正しく設定されていない可能性があります。詳細については、Percona XtraBackup と Amazon S3 を使用した MySQL からの物理的な移行「」を参照してください。
失敗	RDS-EVENT-0254	このカスタマーアカウントの基礎となるストレージクォータが制限を超えました。インスタンスでスケーリングを実行できるように、許容されるストレージクォータを増やしてください。	
失敗	RDS-EVENT-0353	リソース制限が不十分なため、DB インスタンスを作成できません。*#####*。	*#####*には、障害に関する詳細が含まれます。

カテゴリ	RDS イベント ID	メッセージ	メモ
ストレージの減少	RDS-EVENT-0007	割り当てられたストレージが使い果たされました。解決するには追加のストレージを割り当ててください。	DB インスタンスに割り当てられたストレージが消費されました。この問題を解決するには、DB インスタンスに追加のストレージを割り当てます。詳細については、「RDS に関数するよくある質問」を参照してください。[空きストレージ容量] メトリクスを使用して、DB インスタンスのストレージ容量をモニタリングできます。
ストレージの減少	RDS-EVENT-0089	DB インスタンス: *name* の空きストレージ容量が、プロビジョニングされたストレージ [プロビジョニングされたストレージ: *size*、空きストレージ: *size*] が *percentage* と残りわずかです。この問題に対処するために、プロビジョニングされたストレージを増やす必要がある場合があります。	DB インスタンスは割り当てられたストレージの 90% 以上を使用しています。[Free Storage Space] メトリクスを使用して、DB インスタンスのストレージ容量をモニタリングできます。
ストレージの減少	RDS-EVENT-0227	Aurora クラスターのストレージは *amount* テラバイトしか残っておらず、非常に危険です。クラスターのストレージ負荷を軽減するための対策を講じてください。	Aurora ストレージサブシステムの容量が不足しています。

カテゴリ	RDS イベント ID	メッセージ	メモ
メンテナンス	RDS-EVENT-0026	DB インスタンスへオフラインパッチを適用しています。	DB インスタンスのオフラインメンテナンスが実行中です。現在、DB インスタンスは利用できません。
メンテナンス	RDS-EVENT-0027	DB インスタンスへのオフラインパッチ適用が終了しました。	DB インスタンスのオフラインメンテナンスが完了しました。現在、DB インスタンスは利用できます。
メンテナンス	RDS-EVENT-0047	データベースインスタンスにパッチが適用されました。	
メンテナンス	RDS-EVENT-0155	DB インスタンスで DB エンジンのマイナーバージョンをアップグレードできます。	
通知	RDS-EVENT-0044	#####	これはオペレーターが発行する通知です。詳細については、イベントメッセージを参照してください。
通知	RDS-EVENT-0048	このインスタンスには最初にアップグレードする必要のあるリードレプリカがあるため、データベースエンジンのアップグレードを遅らせます。	DB インスタンスへのパッチ適用が遅れました。
通知	RDS-EVENT-0087	DB インスタンスが停止しました。	
通知	RDS-EVENT-0088	DB インスタンスが開始されました。	

カテゴリ	RDS イベント ID	メッセージ	メモ
リードレプリカ	RDS-EVENT-0045	レプリケーションが停止されました。	ストレージ不足のため、DB インスタンスのレプリケーションが停止されました。ストレージを拡張するか、REDO ログの最大サイズを小さくして、レプリケーションを続行してください。サイズが# MiB の REDO ログを収容するには、少なくとも# MiB の空きストレージが必要です。
リードレプリカ	RDS-EVENT-0046	リードレプリカのレプリケーションが再開されました。	このメッセージは、初期にリードレプリカを作成したとき、またはレプリケーションが適切に機能していることを確認するモニタリングメッセージとして表示されます。このメッセージが RDS-EVENT-0045 通知の後に表示される場合は、エラーの後またはレプリケーションが停止した後で、レプリケーションが再開されました。
リードレプリカ	RDS-EVENT-0057	レプリケーションストリーミングは終了しました。	
復旧	RDS-EVENT-0020	DB インスタンスの復旧がスタートされました。復旧時間は、復旧するデータの量に応じて変わります。	

カテゴリ	RDS イベント ID	メッセージ	メモ
復旧	RDS-EVENT-0021	DB インスタンスの復旧が完了しました。	
復旧	RDS-EVENT-0023	緊急スナップショットリクエスト: *message*。	手動バックアップがリクエストされましたが、現在、Amazon RDS は DB スナップショットの作成中です。Amazon RDS で DB スナップショットの作成が完了した後で、リクエストをもう一度送信してください。
復旧	RDS-EVENT-0052	マルチ AZ インスタンスの復旧が開始されました。	復旧時間は、復旧するデータの量に応じて変わります。
復旧	RDS-EVENT-0053	マルチ AZ インスタンスの復旧が完了しました。フェイルオーバーまたはアクティベーションが保留中です。	
復元	RDS-EVENT-0019	DB インスタンス *name* から *name* に復元しました。	DB インスタンスが特定の時点のバックアップから復元されました。

カテゴリ	RDS イベント ID	メッセージ	メモ
セキュリティパッチ	RDS-EVENT-0230	DB インスタンスにシステムアップデートが使用可能です。アップデートの適用方法については、RDS ユーザーガイドの、「DB インスタンスのメンテナンス」を参照してください。	新しいオペレーティングシステムのパッチが使用可能です。 DB インスタンスで、新しいマイナーバージョンのオペレーティングシステムを更新できます。更新の適用については、「オペレーティングシステムアップデートの操作」を参照してください。

DB パラメータグループイベント

次の表は、DB パラメータグループがソースタイプである場合のイベントカテゴリとイベントを示しています。

カテゴリ	RDS イベント ID	メッセージ	メモ
設定変更	RDS-EVENT-0037	パラメータ *name* を適用メソッド *method* で *value* に更新しました。	

DB セキュリティグループイベント

次の表は、DB セキュリティグループをソースタイプとするイベントカテゴリとイベントの一覧です。

> **Note**
>
> DB セキュリティグループは EC2-Classic 用リソースです。EC2-Classic は 2022 年 8 月 15 日に廃止されました。EC2-Classic から VPC に移行していない場合、できるだけ早く移行することをお勧めします。詳細については、「Amazon EC2 ユーザーガイド」の「EC2-

Classic から VPC へ移行」およびブログ記事「EC2-Classic ネットワーキングがリタイア — 準備方法」を参照してください。

カテゴリ	RDS イベント ID	メッセージ	メモ
設定変更	RDS-EVENT-0038	セキュリティグループに変更を適用しました。	
失敗	RDS-EVENT-0039	*user* としての権限を取り消しています。	*user*.が所有するセキュリティグループが存在しません。セキュリティグループの認証が無効なため取り消されました。

DB クラススナップショットイベント

次の表は、DB クラスターのスナップショットがソースタイプである場合のイベントカテゴリとイベントのリストを示しています。

カテゴリ	RDS イベント ID	メッセージ	メモ
バックアップ	RDS-EVENT-0074	手動クラススナップショットの作成	
バックアップ	RDS-EVENT-0075	手動クラススナップショットが作成されました。	
通知	RDS-EVENT-0162	クラスターのスナップショットのエクスポートタスクに失敗しました。	
通知	RDS-EVENT-0163	クラスターのスナップショットのエクスポートタスクがキャンセルされました。	

カテゴリ	RDS イベント ID	メッセージ	メモ
通知	RDS-EVENT-0164	クラスターのスナップショットのエクスポートタスクが完了しました。	
バックアップ	RDS-EVENT-0168	自動クラスタースナップショットを作成しています。	
バックアップ	RDS-EVENT-0169	自動クラスタースナップショットが作成されました。	

RDS Proxy イベント

ソースタイプが RDS Proxy である場合の、イベントのカテゴリとその一覧を次の表に示します。

カテゴリ	RDS イベント ID	メッセージ	メモ
設定変更	RDS-EVENT-0204	RDS が DB プロキシ *name* を変更しました。	
設定変更	RDS-EVENT-0207	RDS において、DB プロキシ *name* のエンドポイントが修正されました。	
設定変更	RDS-EVENT-0213	RDS が DB インスタンスの追加を検出し、そのインスタンスを DB プロキシ *name* のターゲットグループに自動的に追加しました。	
設定変更	RDS-EVENT-0213	RDS が DB インスタンス *name* の作成を検出し、そのインスタンスを DB プロキシ *name* のターゲット	

カテゴリ	RDS イベント ID	メッセージ	メモ
		グループ *name* に自動的に追加しました。	
設定変更	RDS-EVENT-0214	RDS が DB インスタンス *name* の削除を検出し、そのインスタンスを DB プロキシ *name* のターゲットグループ *name* から自動的に削除しました。	
設定変更	RDS-EVENT-0215	RDS が DB クラスター *name* の削除を検出し、そのインスタンスを DB プロキシ *name* のターゲットグループ *name* から自動的に削除しました。	
作成	RDS-EVENT-0203	RDS は DB プロキシ *name* を作成しました。	
作成	RDS-EVENT-0206	RDS は DB プロキシ *name* のエンドポイント*name*を作成しました。	
削除	RDS-EVENT-0205	RDS は DB プロキシ *name* を削除しました。	
削除	RDS-EVENT-0208	RDS は DB プロキシ*name*のエンドポイント *name* を削除しました。	

カテゴリ	RDS イベント ID	メッセージ	メモ
失敗	RDS-EVENT-0243	サブネット *name* に十分な IP アドレスがないため、RDS はプロキシの容量をプロビジョニングできませんでした: *name*。この問題を解決するには、RDS プロキシのドキュメントで推奨されているように、サブネットの未使用の IP アドレスが最小限であることを確認してください。	インスタンスクラスの推奨数を決定するには、「IP アドレス容量の計画」を参照してください。
失敗	RDS-EVENT-0275	RDS は DB プロキシ#への一部の接続をスロットリングしました。クライアントからプロキシへの同時接続リクエストの数が制限を超えました。	

ブルー/グリーンデプロイイベント

次の表は、ブルー/グリーンデプロイがソースタイプである場合のイベントカテゴリとイベントのリストを示します。

ブルー/グリーンデプロイの詳細については、「データベース更新のために Amazon RDS ブルー/グリーンデプロイを使用する」を参照してください。

カテゴリ	Amazon RDS イベント ID	メッセージ	メモ
作成	RDS-EVENT-0244	ブルー/グリーンデプロイタスクが完了しました。グリーン環境のデータベースにさらに変更を加えたり、	

カテゴリ	Amazon RDS イベント ID	メッセージ	メモ
		デプロイを切り替えたりできます。	
失敗	RDS-EVENT-0245	(ソース/ターゲット) DB (インスタンス/クラスター) が見つからなかったため、ブルー/グリーンデプロイの作成に失敗しました。	
削除	RDS-EVENT-0246	ブルー/グリーンデプロイが削除されました。	
通知	RDS-EVENT-0247	###から####へのスイッチオーバーが開始されました。	
通知	RDS-EVENT-0248	ブルー/グリーンデプロイの切り替えが完了しました。	
失敗	RDS-EVENT-0249	ブルー/グリーンデプロイの切り替えがキャンセルされました。	
通知	RDS-EVENT-0259	DB クラスターの###から####へのスイッチオーバーが開始されました。	
通知	RDS-EVENT-0260	DB クラスターの###から####へのスイッチオーバーが完了しました。###から#####、####から###に名前を変更しました。	

カテゴリ	Amazon RDS イベント ID	メッセージ	メモ
失敗	RDS-EVENT-0261	DB クラスターの###から###へのスイッチオーバーは、##によりキャンセルされました。	
通知	RDS-EVENT-0311	DB クラスターの###から###へのスイッチオーバーのシーケンス同期が開始されました。シーケンス使用時にスイッチオーバーを行うと、ダウンタイムが長くなる可能性があります。	
通知	RDS-EVENT-0312	DB クラスターの###から###へのスイッチオーバーのためのシーケンス同期が完了しました。	
失敗	RDS-EVENT-0314	DB クラスターの###から###へのスイッチオーバーのシーケンス同期は、シーケンスが同期に失敗したためキャンセルされました。	

Amazon Aurora ログファイルのモニタリング

すべての RDS データベースエンジンは、監査やトラブルシューティング時にアクセスするログを生成します。ログの種類は、データベースエンジンによって異なります。

AWS Management Console、AWS Command Line Interface (AWS CLI)、または Amazon RDS API を使用して、データベースログにアクセスできます。トランザクションログを表示、監視、またはダウンロードすることはできません。

> **Note**
>
> 場合によっては、ログに非表示のデータが含まれていることがあります。よって、AWS Management Console はログファイルのコンテンツを表示することがありますが、ダウンロードする際は、ログファイルは、空欄です。

トピック

データベースログファイルの表示とリスト化

AWS Management Console を使用して、Amazon Aurora DB エンジンのデータベースログファイルを表示できます。AWS CLI または Amazon RDS API を使用して、ダウンロードまたはモニタリングできるログファイルを一覧表示できます。

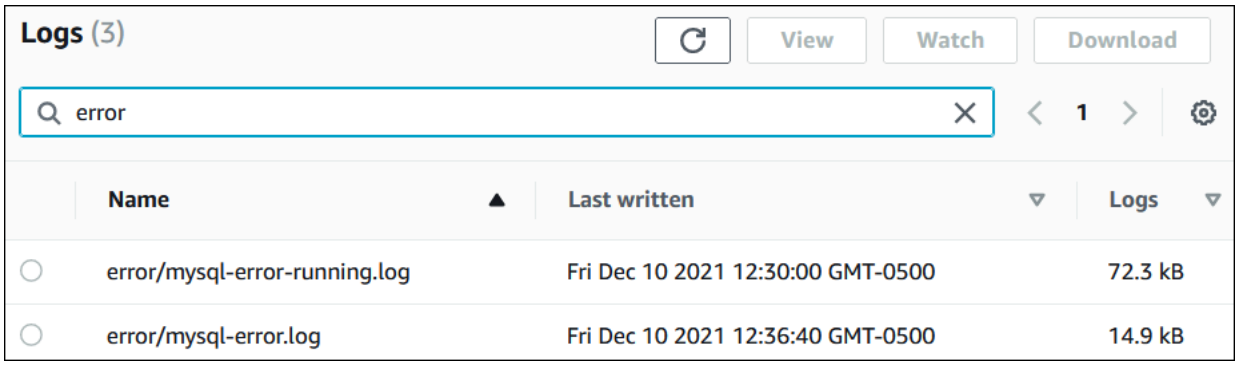
> **Note**
>
> RDS コンソールの Aurora Serverless v1 DB クラスタのログファイルは表示できません。ただし、Amazon CloudWatch コンソール https://console.aws.amazon.com/cloudwatch/ で表示できます。

コンソール

データベースログファイルを閲覧するには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. 表示するログファイルのある DB インスタンスの名前を選択します。
4. [ログとイベント] タブを選択します。
5. [ログ] セクションまで下にスクロールします。
6. (オプション) 検索語を入力して、結果をフィルタリングします。

   次の例では、テキスト **error** でフィルタリングされたログを一覧表示しています。



7. 表示するログを選択してから、[View] (表示) を選択します。

AWS CLI

DB インスタンスで使用できるデータベースログファイルを一覧表示するには、AWS CLI の `describe-db-log-files` コマンドを使用します。

次の例では、DB インスタンス (`my-db-instance`) のログファイルのリストが返ります。

Example

```
aws rds describe-db-log-files --db-instance-identifier my-db-instance
```

RDS API

DB インスタンスの使用可能なデータベースログファイルを一覧表示するには、Amazon RDS API の `DescribeDBLogFiles` アクションを使用します。

データベースログファイルのダウンロード

データベースログファイルをダウンロードするには、AWS Management Console、AWS CLI、または API を使用します。

コンソール

データベースログファイルをダウンロードするには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. 表示するログファイルのある DB インスタンスの名前を選択します。
4. [ログとイベント] タブを選択します。
5. [ログ] セクションまで下にスクロールします。
6. [ログ] セクションで、ダウンロードするログの横にあるボタンを選択し、[ダウンロード] を選択します。
7. 表示されたリンクのコンテキスト (右クリック) メニューを開き、[名前を付けて保存] を選択します。ログファイルを保存する場所を入力し、[保存] を選択します。

AWS CLI

データベースログファイルをダウンロードするには、AWS CLI の download-db-log-file-portion コマンドを使用します。デフォルトでは、このコマンドによってログファイルの最新部分のみがダウンロードされます。ただし、`--starting-token 0` パラメータを指定して、ファイル全体をダウンロードすることもできます。

以下の例では、ログファイル (log/ERROR.4) のすべての内容をダウンロードし、ローカルファイル (errorlog.txt) に格納する方法について説明します。

Example

Linux、macOS、Unix の場合:

```
aws rds download-db-log-file-portion \
    --db-instance-identifier myexampledb \
    --starting-token 0 --output text \
    --log-file-name log/ERROR.4 > errorlog.txt
```

Windows の場合:

```
aws rds download-db-log-file-portion ^
    --db-instance-identifier myexampledb ^
    --starting-token 0 --output text ^
    --log-file-name log/ERROR.4 > errorlog.txt
```

RDS API

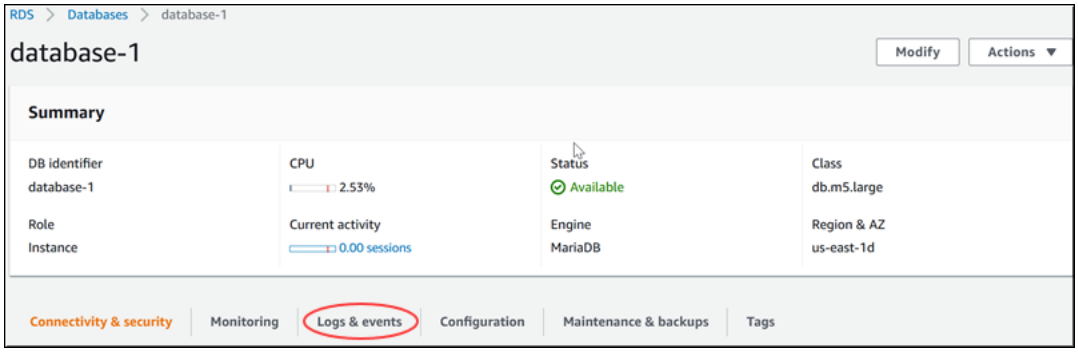
データベースログファイルをダウンロードするには、Amazon RDS API の DownloadDBLogFilePortion アクションを使用します。

データベースログファイルのモニタリング

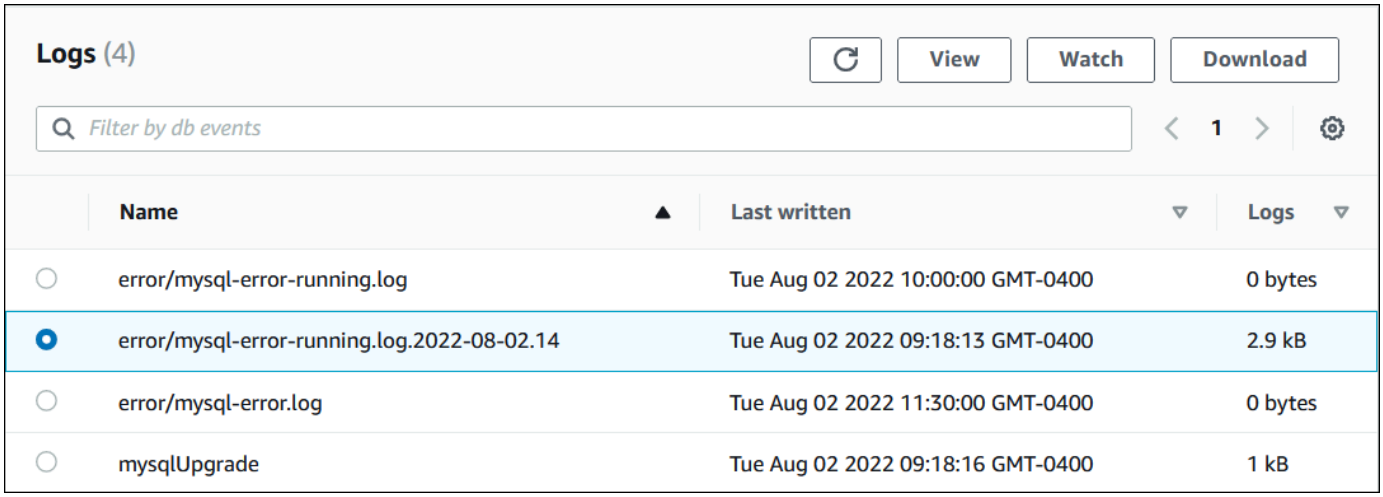
データベースログファイルを監視することは、UNIX または Linux システムでファイルをテーリングすることと同じです。AWS Management Console を使用すると、ログファイルを監視できます。RDS は 5 秒ごとにログの末尾を更新します。

データベースログファイルをモニタリングするには

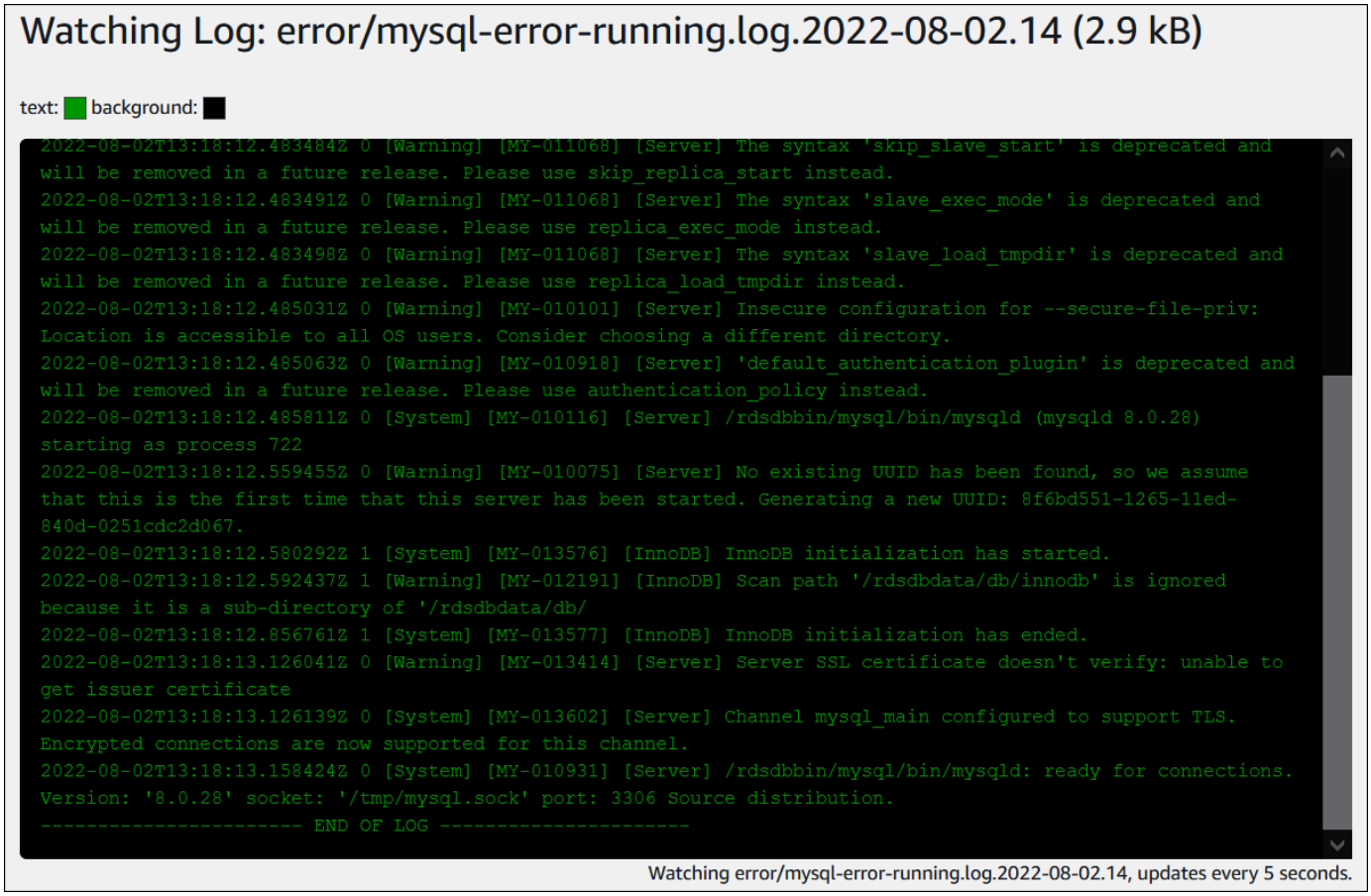
1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. 表示するログファイルのある DB インスタンスの名前を選択します。
4. [ログとイベント] タブを選択します。



5. [ログ] セクションでログファイルを選択し、[モニタリング] を選択します。



RDS には、次の MySQL の例のようにログの末尾が表示されます。



Amazon CloudWatch Logs へのデータベースログの発行

オンプレミスデータベースでは、データベースログはファイルシステムに存在します。Amazon RDS では、DB クラスターのファイルシステム上のデータベースログへのホストアクセスが許可されません。このため、Amazon RDS では、Amazon CloudWatch Logs にデータベースログをエクスポートできます。CloudWatch Logs を使用すると、ログデータのリアルタイム分析を実行できます。高い耐久性を持つストレージにデータを保存し、CloudWatch Logs エージェントを使用したデータの管理を実行できます。

トピック

- RDS と CloudWatch Logs の統合の概要
- CloudWatch Logs に発行するログの決定
- CloudWatch Logs に発行するログの指定
- CloudWatch Logs でのログの検索とフィルタリング

RDS と CloudWatch Logs の統合の概要

CloudWatch Logs では、ログストリーミングは、同じ出典を共有する一連のログイベントです。CloudWatch Logs でのログの各ソースで各ログストリームが構成されます。ロググループは、保持、モニタリング、アクセス制御について同じ設定を共有するログストリームのグループです。

Amazon Aurora は、DB クラスターログレコードをロググループに継続的にストリームします。例えば、発行した各タイプのログについて、ロググループ `/aws/rds/cluster/`*`cluster_name`*`/`*`log_type`* があることを考えます。このロググループは、ログを生成するデータベースインスタンスと同じ AWS リージョンにあります。

AWS は、CloudWatch Logs に発行されたログデータを、保持期間を指定しない限り、無期限に保持します。詳細については、「CloudWatch Logs でのログデータ保管期間の変更」を参照してください。

CloudWatch Logs に発行するログの決定

各 RDS データベースエンジンは、独自のログセットをサポートします。データベースエンジンのオプションについては、以下のトピックを確認してください。

- the section called “CloudWatch Logs への Aurora MySQL ログの発行”
- the section called “CloudWatch Logs への Aurora PostgreSQL ログの発行”

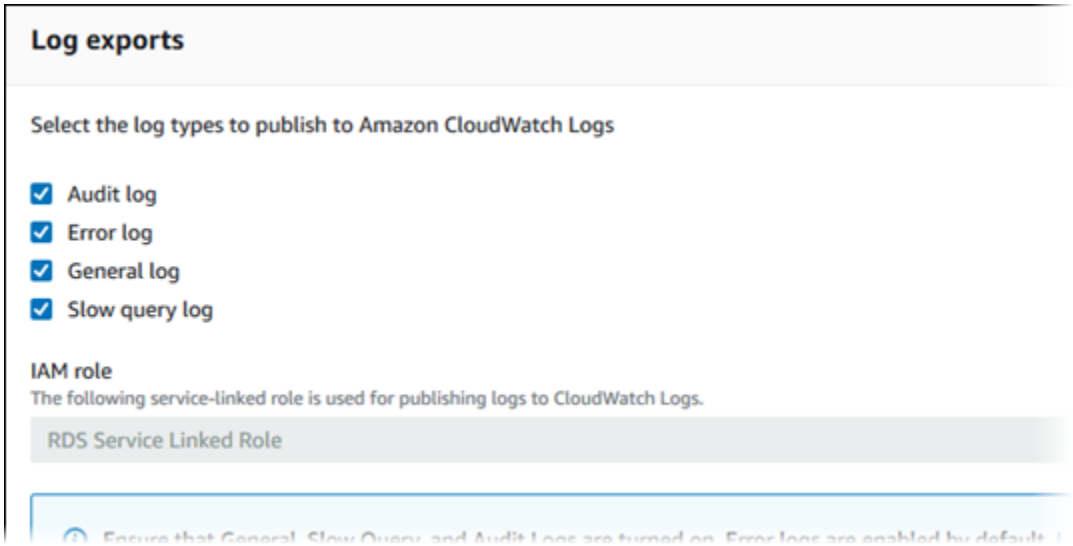
CloudWatch Logs に発行するログの指定

コンソールで発行するログを指定します。AWS Identity and Access Management (IAM) にサービスリンクロールがあることを確認します。サービスにリンクされたロールの詳細については、「Amazon Aurora のサービスにリンクされたロールの使用」を参照してください。

発行するログを指定するには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Databases] (データベース) を選択します。
3. 次のいずれかを実行します。
   - [データベースの作成] を選択します。
   - 一覧からデータベースを選択し、[Modify] (変更) を選択します。
4. [Logs exports] (ログのエクスポート) で、発行するログを選択します。

次の例では、監査ログ、エラーログ、全般ログ、スロークエリログを指定します。



CloudWatch Logs でのログの検索とフィルタリング

CloudWatch コンソールを使用して、指定した基準を満たすログエントリを検索することができます。ログには、CloudWatch Logs コンソールにつながる RDS コンソールからアクセスすることも、CloudWatch Logs コンソールから直接アクセスすることもできます。

RDS コンソールを使用して RDS ログを検索するには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Databases] (データベース) を選択します。
3. DB クラスターまたは DB インスタンスを選択します。
4. [設定] を選択します。
5. [Published logs] (発行されたログ) で、表示するデータベースログを選択します。

CloudWatch Logs コンソールを使用して RDS ログを検索するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインで、[Log groups] (ロググループ) を選択します。
3. フィルタボックスに **`/aws/rds`** と入力します。
4. [ロググループ] で、検索するログストリームを含むロググループの名前を選択します。
5. [ログストリーム] で、検索するログストリームの名前を選択します。

6. [Log Events (ログイベント)] で、使用するフィルター構文を入力します。

詳細については、Amazon CloudWatch Logs ユーザーガイドの「ログデータの検索およびフィルタリング」を参照してください。RDS ログをモニタリングする方法を説明するブログチュートリアルについては、「Amazon CloudWatch Logs、AWS Lambda、および Amazon SNS を使用して Amazon RDS のプロアクティブなデータベースモニタリングを構築する」を参照してください。

REST を用いたログファイルの内容の読み取り

Amazon RDS では、DB インスタンスのログファイルへのアクセスを許可する REST エンドポイントを使用できます。これは、Amazon RDS ログファイルの内容を取り出すアプリケーションを作成される場合に有用です。

構文は次のとおりです。

```
GET /v13/downloadCompleteLogFile/DBInstanceIdentifier/LogFileName HTTP/1.1
Content-type: application/json
host: rds.region.amazonaws.com
```

以下のパラメータは必須です。

- *DBInstanceIdentifier*— ダウンロードするログファイルを含む DB インスタンスの名前。
- *LogFileName*—ダウンロードするログファイルの名前。

このレスポンスには、ストリーミングとしてリクエストされたログファイルの内容が含まれます。

次の例では、us-west-2 リージョンの sample-sql という名前の DB インスタンスの log/ERROR.6 という名前のログファイルをダウンロードします。

```
GET /v13/downloadCompleteLogFile/sample-sql/log/ERROR.6 HTTP/1.1
host: rds.us-west-2.amazonaws.com
X-Amz-Security-Token: AQoDYXdzEIH//////////
wEa0AIXLhngC5zp9CyB1R6abwKrXHVR5efnAVN3XvR7IwqKYalFSn6UyJuEFTft9nObglx4QJ+GXV9cpACkETq=
X-Amz-Date: 20140903T233749Z
X-Amz-Algorithm: AWS4-HMAC-SHA256
X-Amz-Credential: AKIADQKE4SARGYLE/20140903/us-west-2/rds/aws4_request
X-Amz-SignedHeaders: host
X-Amz-Content-SHA256: e3b0c44298fc1c229afbf4c8996fb92427ae41e4649b934de495991b7852b855
X-Amz-Expires: 86400
```

```
X-Amz-Signature: 353a4f14b3f250142d9afc34f9f9948154d46ce7d4ec091d0cdabbcf8b40c558
```

存在しない DB インスタンスを指定した場合、レスポンスは次のエラーになります。

- DBInstanceNotFound—*DBInstanceIdentifier* が既存の DB インスタンスを参照していません。(HTTP ステータスコード: 404)

Aurora MySQL データベースのログファイル

Aurora MySQL ログは、Amazon RDS コンソール、Amazon RDS API、AWS CLI、または AWS SDK を通じて直接モニタリングできます。また、ログをメインデータベースのデータベーステーブルに書き込み、そのテーブルに対してクエリを実行することで、MySQL ログにアクセスできます。mysqlbinlog ユーティリティを使用して、バイナリログをダウンロードできます。

ファイルベースのデータベースログの表示、ダウンロード、モニタリングの詳細については、「Amazon Aurora ログファイルのモニタリング」を参照してください。

トピック

- Aurora MySQL データベースログの概要
- Amazon CloudWatch Logs への Aurora MySQL ログの発行
- テーブルベースの Aurora MySQL ログの管理
- Aurora MySQL バイナリログの設定
- MySQL バイナリログにアクセスする

Aurora MySQL データベースログの概要

次の種類の Aurora MySQL ログファイルをモニタリングできます。

- エラーログ
- スロークエリログ
- 全般ログ
- [監査ログ]

Aurora MySQL のエラーログはデフォルトで生成されます。DB パラメータグループにパラメータを設定することで、低速クエリと一般ログを生成できます。

トピック

- Aurora MySQL エラーログ
- Aurora MySQL のスロークエリと一般ログ
- Aurora MySQL の監査ログ
- Aurora MySQL のログのローテーションと保持

Aurora MySQL エラーログ

Aurora MySQL は `mysql-error.log` ファイルにエラーを書き込みます。各ログファイルには、それぞれ生成された時間 (UTC) がファイル名に付加されます。ログファイルには、タイムスタンプも付加され、ログエントリがいつ書き込まれたかを調べるために役立ちます。

Aurora MySQL では起動時、シャットダウン時、およびエラー検出時にのみ、エラーログへの書き込みが行われます。DB インスタンスでは、新しいエントリがエラーログに書き込まれないまま、数時間または数日が経過することがあります。最近のエントリがない場合、それは、サーバーにログエントリになり得るエラーが発生しなかったためです。

設計上、エラーログはフィルタリングされ、エラーなどの予期しないイベントのみが表示されます。ただし、エラーログには、クエリの進行状況など、表示されない追加のデータベース情報も含まれています。したがって、実際のエラーがなくても、継続的なデータベースアクティビティのためにエラーログのサイズが増加する可能性があります。また、AWS Management Console のエラーログには特定のサイズがバイト単位またはキロバイト単位で表示されている場合がありますが、ダウンロードすると 0 バイトになる場合があります。

Aurora MySQL は 5 分ごとに `mysql-error.log` をディスクに書き込みます。ログの内容が `mysql-error-running.log` に追加されます。

Aurora MySQL は `mysql-error-running.log` ファイルを 1 時間ごとにローテーションします。

> **Note**
>
> ログの保持期間は、Amazon RDS と Aurora で異なります。

Aurora MySQL のスロークエリと一般ログ

Aurora MySQL のスロークエリログと一般ログを、ファイルまたはデータベーステーブルに書き込めます。このためには、DB パラメータグループにパラメータを設定します。DB パラメータグループの作成と変更の詳細については、「「パラメータグループを使用する」」を参照してください。Amazon RDS コンソール、Amazon RDS API、Amazon RDS CLI、または AWS SDK を使用して、スロークエリログまたは一般ログを表示する前に、以下のパラメータを設定する必要があります。

以下のリストに示すパラメータを使用して Aurora MySQL のログ記録を制御できます。

- `slow_query_log`: スロークエリログを作成するには、1 に設定します。デフォルトは 0 です。

- general_log: 一般ログを作成するには、1 に設定します。デフォルトは 0 です。
- long_query_time: ファストクエリがスロークエリログに記録されないようにするために、ログに記録されるクエリの最短実行時間の値を秒単位で指定します。デフォルトは 10 秒で、最小値は 0 です。log_output = FILE の場合は、マイクロ秒の精度になるように、浮動小数点値を指定できます。log_output = TABLE の場合は、秒の精度になるように、整数値を指定する必要があります。実行時間が long_query_time の値を超えたクエリのみがログに記録されます。例えば、long_query_time を 0.1 に設定すると、実行時間が 100 ミリ秒未満のすべてのクエリはログに記録されなくなります。
- log_queries_not_using_indexes: インデックスを使用しないすべてのクエリをスロークエリログに記録するには、1 に設定します。インデックスを使用しないクエリは、その実行時間が long_query_time パラメータの値未満であっても口グに記録されます。デフォルトは 0 です。
- log_output *option*: log_output パラメータに指定できるオプションは、次のとおりです。
  - TABLE - 一般クエリを mysql.general_log テーブルに、スロークエリを mysql.slow_log テーブルに書き込みます。
  - FILE - 一般クエリログとスロークエリログの両方をファイルシステムに書き込みます。
  - NONE - ログ記録を無効にします。

  Aurora MySQL バージョン 2 の場合、log_output のデフォルトは FILE です。

スロークエリと一般ログの詳細については、MySQL ドキュメントの以下のトピックを参照してください。

- スロークエリログ
- 一般クエリログ

Aurora MySQL の監査ログ

Aurora MySQL の監査ログは、高度な監査と呼ばれます。高度な監査を有効にするには、特定の DB クラスターパラメータを設定します。詳細については、「Amazon Aurora MySQL DB クラスターでのアドバンストな監査の使用」を参照してください。

Aurora MySQL のログのローテーションと保持

ログ記録が有効になっている場合、Amazon Aurora は、ログファイルの削除を定期的に実行します。これは、ログファイルが大きくなることでデータベースが使用できなくなったりパフォーマンス

に影響する可能性を低く抑えるための予防措置です。Aurora MySQL は、次のようにローテーションと削除を処理します。

- Aurora MySQL エラーログファイルのサイズは、DB インスタンスのローカルストレージの 15 パーセント以下に制約されます。このしきい値を維持するために、ログは 1 時間ごとに自動的にローテーションされます。Aurora MySQL は 30 日後、またはディスク領域の 15 % が使用されると、ログを削除します。古いログファイルを削除した後、ログファイルの合計サイズがしきい値を超えている場合、ログファイルのサイズがしきい値以下になるまで、最も古いログファイルから順に削除されます。
- Aurora MySQL は、24 時間後、またはストレージの 15% が消費されると、監査、一般、および低速クエリログを削除します。
- FILE ログ記録が有効になっている場合、一般ログとスロークエリログファイルの検査が 1 時間ごとに実行され、作成後 24 時間を超えた古いログファイルは削除されます。場合によっては、削除後の残りのログファイルの合計サイズが、DB インスタンスのローカル領域のしきい値である 15 % を超えることがあります。この場合、ログファイルのサイズがしきい値以下になるまで、最も古いログファイルから順に削除されます。
- TABLE ログ記録が有効化されている場合、ログテーブルはローテーションまたは削除されません。結合されたすべてのログのサイズが大きすぎると、ログテーブルは切り捨てられます。low_free_storage イベントにサブスクライブして、ログテーブルが手動でローテーションされたり、領域を解放する t まえに削除されたりしたときに通知を受け取ることができます。詳細については、「Amazon RDS イベント通知の操作」を参照してください。

  mysql.general_log テーブルの手動ローテーションは、mysql.rds_rotate_general_log プロシージャを呼び出すことで実行できます。mysql.slow_log テーブルのローテーションは、mysql.rds_rotate_slow_log プロシージャを呼び出すことで実行できます。

  ログテーブルをローテーションすると、現在のログテーブルがバックアップのログテーブルにコピーされ、現在のログテーブル内にあるエントリは削除されます。バックアップのログテーブルが既に存在する場合は、現在のログテーブルをバックアップにコピーする前に、削除されます。バックアップのログテーブルは、必要に応じて照会することができます。mysql.general_log テーブルに対するバックアップのログテーブルは、mysql.general_log_backup という名前になります。mysql.slow_log テーブルに対するバックアップのログテーブルは、mysql.slow_log_backup という名前になります。
- Aurora MySQL 監査ログは、ファイルサイズが 100 MB に達するとローテーションされ、24 時間後に削除されます。

Amazon RDS コンソール、Amazon RDS API、Amazon RDS CLI、または AWS SDK からログを使用するには、log_output パラメータを FILE に設定します。Aurora MySQL エラーログと同様、これらのログファイルは 1 時間ごとにローテーションされます。直前 24 時間以内に生成されたログファイルが保持されます。Amazon RDS と Aurora で保持期間が異なる点に注意してください。

Amazon CloudWatch Logs への Aurora MySQL ログの発行

Aurora MySQL DB クラスターを設定して、ログデータを Amazon CloudWatch Logs のロググループに発行することができます。CloudWatch Logs を使用すると、ログデータのリアルタイム分析や、CloudWatch を使用したアラームの作成、メトリクスの表示を行うことができます。CloudWatch Logs を使用して、耐久性の高いストレージにログレコードを格納できます。詳細については、「Amazon CloudWatch Logs への Amazon Aurora MySQL ログの発行」を参照してください。

テーブルベースの Aurora MySQL ログの管理

DB パラメータグループを作成し、log_output サーバーパラメータを TABLE に設定することで、DB インスタンス上のテーブルに一般ログとスロークエリログを書き込むことができます。その後、一般クエリは mysql.general_log テーブルに記録され、スロークエリは mysql.slow_log テーブルに記録されます。それらのテーブルに対してクエリを実行することでログの情報にアクセスできます。このログ記録を有効にすると、データベースに書き込まれるデータの量が増え、パフォーマンスが低下することがあります。

一般ログもスロークエリログもデフォルトで無効になっています。テーブルへのログ記録を有効にするには、general_log と slow_query_log のサーバーパラメータを 1 に設定する必要があります。

ログテーブルは、それぞれのログ記録アクティビティのパラメータを 0 にリセットしてログ記録をオフにするまで、拡大し続けます。大量のデータが長期にわたって蓄積されることがよくあり、割り当てストレージ領域の大部分を使い果たすことがあります。Amazon Aurora では、ログテーブルを切り詰めることはできませんが、その内容を移動することはできます。テーブルのローテーションにより、その内容がバックアップテーブルに保存され、新しい空のログテーブルが作成されます。以下のコマンドラインプロシージャを使用して、ログテーブルを手動でローテーションされることができます。ここで表示されている PROMPT> はコマンドプロンプトです。

```
PROMPT> CALL mysql.rds_rotate_slow_log;
PROMPT> CALL mysql.rds_rotate_general_log;
```

以前のデータを完全に削除し、ディスク領域を再利用するには、該当するプロシージャを 2 回連続で呼び出します。

Aurora MySQL バイナリログの設定

バイナリログは、Aurora MySQL サーバーインスタンスで行われたデータ変更に関する情報を含む、一連のログファイルです。バイナリログには、以下のような情報が含まれています。

- テーブルの作成や行の変更など、データベースの変更が記述されたイベント
- データを更新した各ステートメントの実行時間に関する情報
- データを更新する可能性があったものの、それが実行されていないステートメントのイベント

バイナリログには、レプリケーション中に送信されるステートメントが記録されます。また、一部のリカバリオペレーションにもバイナリログが必要です。詳細については、MySQL ドキュメントの「バイナリログ」ならびに「バイナリログの概要」を参照してください。

バイナリログは、プライマリ DB インスタンスからのみアクセスでき、レプリカからはアクセスできません。

Amazon Aurora の MySQL では、行ベース、ステートメントベース、および混合のバイナリログ形式がサポートされています。特定バイナリログ形式が必要でない場合は、混合形式を使用することをお勧めします。Aurora MySQL の各種バイナリログ形式の詳細については、MySQL ドキュメントの「Binary logging formats」(バイナリログ記録形式) を参照してください。

レプリケーションを使用する予定の場合は、バイナリログ記録形式が重要です。ソースに記録されてレプリケーションターゲットに送信されるデータ変更記録が決定されるからです。レプリケーションのさまざまなバイナリログ記録のメリットとデメリットについては、MySQL ドキュメントの「ステートメントベースおよび行ベースレプリケーションのメリットとデメリット」を参照してください。

> **Important**
>
> バイナリログ形式を行ベースに設定すると、バイナリログファイルが巨大になることがあります。巨大なバイナリログファイルにより、DB クラスターの使用可能なストレージの量が減ります。また、DB クラスターの復元オペレーションの実行にかかる時間が長くなることがあります。
>
> ステートメントベースのレプリケーションは、ソース DB クラスターとリードレプリカの間の不整合の原因になります。詳細については、MySQL ドキュメントの「バイナリロギングでの安全および安全でないステートメントの判断」を参照してください。

バイナリログを有効にすると、DB クラスターへの書き込みディスク I/O 操作の回数が増えます。VolumeWriteIOPs CloudWatch メトリクスを使用して、IOPS の使用状況をモニタリングできます。

MySQL バイナリログ形式を設定するには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[パラメータグループ] を選択します。
3. 変更したい DB クラスターに関連付ける DB クラスターのパラメータグループを選択してください。

   デフォルトのパラメータグループを変更することはできません。DB クラスターがデフォルトのパラメータグループを使用している場合、新しいパラメータグループを作成し DB クラスターと関連付けます。

   パラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。
4. [アクション] で、[編集] を選択します。
5. binlog_format パラメータを、選択したバイナリログ形式 (ROW、STATEMENT、または MIXED) に設定します。値 OFF を使用して、バイナリログをオフにすることもできます。

   > **Note**
   >
   > DB クラスターパラメータグループで binlog_format を OFF に設定すると、log_bin セッション変数が無効になります。これにより、Aurora MySQL DB クラスターのバイナリログ記録が無効になり、binlog_format セッション変数がデータベースのデフォルト値の ROW にリセットされます。
6. [変更の保存] を選択して、更新を DB クラスターパラメータグループに保存します。

これらのステップを実行した後、変更を適用するには、DB クラスターのライターインスタンスを再起動する必要があります。Aurora MySQL バージョン 2.09 以前では、ライターインスタンスを再起動すると、DB クラスター内のすべてのリーダーインスタンスも再起動されます。Aurora MySQL バージョン 2.10 以降では、すべてのリーダーインスタンスを手動で再起動する必要があります。詳細については、「Amazon Aurora DB クラスターまたは Amazon Aurora DB インスタンスの再起動」を参照してください。

> **Important**
>
> DB クラスターパラメータグループを変更すると、そのパラメータグループを使用するすべての DB クラスターに影響を与えます。AWS リージョン内の異なる Aurora MySQL DB クラスターに異なるバイナリログ形式を指定する場合、DB クラスターは異なる DB クラスターパラメータグループを使用する必要があります。これらのパラメータグループは、さまざまなログ形式を識別します。各 DB クラスターに適切な DB クラスターパラメータグループを割り当てます。Aurora MySQL パラメータの詳細については、「Aurora MySQL 設定パラメータ」を参照してください。

MySQL バイナリログにアクセスする

mysqlbinlog ユーティリティを使用して、RDS for MySQL DB インスタンスからバイナリログをダウンロードまたはストリーミングできます。バイナリログはローカルコンピュータにダウンロードされ、mysql ユーティリティを使用してログの再生などの操作を実行できます。mysqlbinlog ユーティリティの使用の詳細については、MySQL ドキュメントの「バイナリログファイルのバックアップのための mysqlbinlog の使用」を参照してください。

Amazon RDS インスタンスに対して mysqlbinlog ユーティリティを実行するには、以下のオプションを使用します。

- `--read-from-remote-server` – 必須。
- `--host` - インスタンスのエンドポイントからの DNS 名。
- `--port` - インスタンスによって使用されるポート。
- `--user` – `REPLICATION SLAVE` アクセス許可を付与された MySQL ユーザー。
- `--password` - MySQL ユーザーのパスワード。パスワード値を省略省略した場合、ユーティリティによってパスワードの入力を求められます。
- `--raw` — バイナリ形式のファイルをダウンロードします。
- `--result-file` - raw 出力を受け取るローカルファイル。
- `--stop-never` — バイナリログファイルをストリーミングします。
- `--verbose` — `ROW` binlog 形式を使用するとき、このオプションを含めると、行イベントが疑似 SQL ステートメントとして表示されます。`--verbose` オプションの詳細については、MySQL ドキュメントの「mysqlbinlog row event display」(mysqlbinlog の行イベントの表示) を参照してください。

- 1 つ以上のバイナリログファイルの名前を指定します。使用可能なログのリストを取得するには、SQL コマンド SHOW BINARY LOGS を使用します。

mysqlbinlog のオプションの詳細については、MySQL ドキュメントの「mysqlbinlog - バイナリログファイルを処理するためのユーティリティ」を参照してください。

以下の例では、mysqlbinlog ユーティリティの使用方法を示します。

Linux、macOS、Unix の場合:

```
mysqlbinlog \
    --read-from-remote-server \
    --host=MySQLInstance1.cg034hpkmmjt.region.rds.amazonaws.com \
    --port=3306  \
    --user ReplUser \
    --password \
    --raw \
    --verbose \
    --result-file=/tmp/ \
    binlog.00098
```

Windows の場合:

```
mysqlbinlog ^
    --read-from-remote-server ^
    --host=MySQLInstance1.cg034hpkmmjt.region.rds.amazonaws.com ^
    --port=3306  ^
    --user ReplUser ^
    --password ^
    --raw ^
    --verbose ^
    --result-file=/tmp/ ^
    binlog.00098
```

Amazon RDS では、通常、バイナリログはできる限り早く消去されますが、mysqlbinlog によってアクセスされるバイナリログはインスタンスで保持される必要があります。RDS でバイナリログを保持する時間数を指定するには、mysql.rds_set_configuration ストアドプロシージャを使用して、ログのダウンロードするのに十分な期間を指定します。保持期間を設定したら、DB インスタンスのストレージ使用状況をモニタリングして、保持されたバイナリログに必要以上の容量が使用されないようにします。

以下の例では、保持期間を 1 日に設定しています。

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

現在の設定を表示するには、mysql.rds_show_configuration ストアドプロシージャを使用します。

```
call mysql.rds_show_configuration;
```

Aurora PostgreSQL データベースログファイル

Aurora PostgreSQL では、データベースアクティビティをデフォルトの PostgreSQL ログファイルに記録します。オンプレミスの PostgreSQL DB インスタンスの場合、これらのメッセージは `log/postgresql.log` にローカルに保存されます。Aurora PostgreSQL DB クラスター、の場合、ログファイルは Aurora クラスター にあります。また、Amazon RDS コンソールを使用して、コンテンツを表示またはダウンロードする必要があります。デフォルトのロギングレベルは、ログインの失敗、致命的なサーバーエラー、デッドロック、およびクエリエラーをキャプチャします。

ファイルベースのデータベースログの表示、ダウンロード、モニタリングの方法の詳細については、「Amazon Aurora ログファイルのモニタリング」を参照してください。PostgreSQL ログの詳細については、「Amazon RDS および Aurora PostgreSQL ログの操作: パート 1」および「Amazon RDS および Aurora PostgreSQL ログの操作: パート 2」を参照してください。

このトピックで説明した標準の PostgreSQL ログに加えて、Aurora PostgreSQL は PostgreSQL 監査エクステンション (`pgAudit`) もサポートしています。規制対象の業界や政府機関のほとんどは、法的要件に準拠するために、データに加えられた変更の監査ログまたは監査証跡を維持する必要があります。pgAudit のインストールおよび使用の詳細については、「pgAudit を使用してデータベースのアクティビティを記録する」を参照してください。

トピック

- ロギング動作に影響するパラメータ
- Aurora PostgreSQL DB クラスターのクエリログ記録をオンにする

ロギング動作に影響するパラメータ

さまざまなパラメータを変更することで、Aurora PostgreSQL DB クラスター のロギング動作をカスタマイズできます。次のテーブルには、ログの保存期間、ログをローテーションするタイミング、ログを CSV (カンマ区切り値) 形式で出力するかどうかなどに影響するパラメータがあります。他の設定の中でも、STDERR に送信されたテキスト出力を確認できます。変更可能なパラメータの設定を変更するには、Aurora PostgreSQL DB クラスターのカスタム DB クラスター パラメータグループを使用します。詳細については、「「パラメータグループを使用する」」を参照してください。テーブルに記載されているように、`log_line_prefix` は変更できません。

パラメータ	デフォルト	[Description] (説明)
log_destination	stderr	ログの出力形式を設定します。デフォルトは `stderr` ですが、設定に `csvlog` を追加して

パラメータ	デフォルト	[Description] (説明)
		カンマ区切り値 (CSV) を指定することもできます。詳細については、「ログの送信先の設定 (stderr、csvlog)」を参照してください。
log_filename	postgresql.log.%Y-%m-%d-%H%M	ログファイル名のパターンを指定します。デフォルトに加えて、このパラメータはファイル名パターンの `postgresql.log.%Y-%m-%d` と `postgresql.log.%Y-%m-%d-%H` をサポートします。
log_line_prefix	%t:%r:%u@%d:[%p]:	時間 (%t)、リモート ホスト (%r)、ユーザー (%u)、データベース (%d)、およびプロセス ID (%p) を記録するために、`stderr` に書き込まれる各ログ行のプレフィックスを定義します。このパラメータは変更できません。
log_rotation_age	60	ログファイルが自動的にローテーションされるまでの分数。この値は、1 ~ 1,440 分の間で変更できます。詳細については、「ログファイルのローテーションの設定」を参照してください。
log_rotation_size	–	ログが自動的にローテーションされるサイズ (kB)。この値は 50,000 ~ 1,000,000 キロバイトの範囲で変更できます。詳細については、「ログファイルのローテーションの設定」を参照してください。
rds.log_retention_period	4320	指定した時間 (分) より古い PostgreSQL ログは削除されます。デフォルト値の 4,320 分では、3 日後にログファイルが削除されます。詳細については、「ログの保持期間の設定」を参照してください。

アプリケーションの問題を特定するには、ログでクエリの失敗、ログインの失敗、デッドロック、および致命的なサーバーエラーを探すことができます。例えば、従来のアプリケーションを Oracle か

ら Aurora PostgreSQL に変換したが、一部のクエリは正しく変換されなかったとします。これらの誤った形式のクエリは、ログにエラーメッセージを生成し、ログから問題を特定することができます。クエリログの詳細については、「Aurora PostgreSQL DB クラスターのクエリログ記録をオンにする」参照してください。

次のトピックでは、PostgreSQL ログの基本的な詳細を制御するさまざまなパラメータの設定方法について説明します。

トピック

- ログの保持期間の設定
- ログファイルのローテーションの設定
- ログの送信先の設定 (stderr、csvlog)
- log_line_prefix パラメータの概要

ログの保持期間の設定

`rds.log_retention_period` パラメータは、Aurora PostgreSQL DB クラスター がログファイルを保持する期間を指定します。デフォルトの設定は 3 日 (4,320 分) ですが、この値を 1 日 (1,440 分) から 7 日 (10,080 分) までの任意の時間に設定できます。Aurora PostgreSQL DB クラスター に、一定期間ログファイルを保持するのに十分なストレージがあることを確認してください。

ログを定期的に Amazon CloudWatch Logs に公開することをお勧めします。これにより、ログが Aurora PostgreSQL DB クラスターから削除された後も、システムデータを表示して分析できます。詳細については、Amazon CloudWatch Logs への Aurora PostgreSQL ログの発行。CloudWatch での公開を設定した後、ログが CloudWatch Logs に公開されるまで、Aurora はログを削除しません。

Amazon Aurora は、DB インスタンスのストレージがしきい値に達すると、古い PostgreSQL ログを圧縮します。Aurora は、gzip 圧縮ユーティリティを使用してファイルを圧縮します。詳細については、gzip のウェブサイトを参照してください。

DB インスタンスのストレージが少なく、使用可能なすべてのログが圧縮されると、次のような警告が表示されます。

```
Warning: local storage for PostgreSQL log files is critically low for
this Aurora PostgreSQL instance, and could lead to a database outage.
```

十分なストレージがない場合、Aurora は指定した保持期間が終了する前に圧縮済みの PostgreSQL ログを削除する可能性があります。その場合は、次のようなメッセージが表示されます。

```
The oldest PostgreSQL log files were deleted due to local storage constraints.
```

ログファイルのローテーションの設定

Aurora は、デフォルトで 1 時間ごとに新しいログファイルを作成します。このタイミングは、log_rotation_age パラメータによって制御されます。このパラメータのデフォルト値は 60 (分) ですが、1 分から 24 時間 (1,440 分) までの任意の時間に設定できます。ローテーションの時期になると、新しい個別のログファイルが作成されます。ファイルには、log_filename パラメータによって指定されたパターンに従って名前が付けられます。

ログファイルは、log_rotation_size パラメータで指定されたサイズに従ってローテーションすることもできます。このパラメータは、ログが指定されたサイズ (キロバイト単位) に達したときにローテーションされるように指定します。デフォルトの log_rotation_size は、Aurora PostgreSQL DB クラスターの場合は 100,000 KB (キロバイト) ですが、この値を 50,000 ~ 1,000,000 キロバイトの任意の値に設定できます。

ログファイル名は、log_filename パラメータで指定されたファイル名のパターンに基づきます。このパラメータに使用できる設定は次のとおりです。

- postgresql.log.%Y-%m-%d — ログファイル名のデフォルトフォーマット。年、月、日をログファイルの名前に含めます。
- postgresql.log.%Y-%m-%d-%H — ログファイル名形式に時間を含めます。
- postgresql.log.%Y-%m-%d-%H%M — 時間:分をログファイル名形式に含めます。

log_rotation_age パラメータを 60 分未満に設定した場合は、log_filename パラメータを分形式に設定します。

詳細については、PostgreSQL ドキュメントの「log_rotation_age」と「log_rotation_size」を参照してください。

ログの送信先の設定 (**stderr**、**csvlog**)

デフォルトでは、Aurora PostgreSQL はスタンダードエラー (stderr) 形式でログを生成します。この形式は、log_destination パラメータのデフォルト設定です。各メッセージには、log_line_prefix パラメータで指定したパターンを使用してプレフィックスが付きます。詳細については、「log_line_prefix パラメータの概要」を参照してください。

Aurora PostgreSQL は、csvlog フォーマットでログを生成することもできます。csvlog は、ログデータをカンマ区切り値 (CSV) データとして分析する場合に便利です。例えば、log_fdw 拡張機能

を使用して外部テーブルとしてログを使用するとします。stderr ログファイルについて作成された外部テーブルには、ログイベントデータを含む 1 つの列が含まれます。log_destination パラメータに csvlog を追加すると、外部テーブルの複数の列の区切りを含む CSV 形式のログファイルが取得できます。ログをより簡単に分類して分析できるようになりました。

このパラメータに csvlog を指定する場合、stderr ファイル と csvlog ファイル の両方が生成されることに注意してください。ログのストレージと回転率に影響する rds.log_retention_period とその他の設定を考慮し、ログによって消費されるストレージに注意してください。stderr と csvlog を使用すると、ログで消費されるストレージが 2 倍以上になります。

log_destination に csvlog を追加して、stderr だけに戻す場合は、パラメータをリセットする必要があります。そのためには、Amazon RDS コンソールを開いて、インスタンスのカスタム DB クラスター パラメータグループを開きます。log_destination パラメータを選択し、[Edit parameter] (パラメータの編集) を選択し、[Reset] (リセット) を選択します。

ログの設定の詳細については、「Amazon RDS および Aurora PostgreSQL ログの操作:パート 1」を参照してください。

log_line_prefix パラメータの概要

stderr ログ形式では、log_line_prefix パラメータで指定された詳細が、以下のように各ログメッセージにプレフィックスとして付加されます。

```
%t:%r:%u@%d:[%p]:t
```

この設定は変更できません。stderr に送信される各ログエントリには次の情報が含まれます。

- %t – ログエントリの時刻。
- %r – リモートホストのアドレス。
- %u@%d – ユーザー名 @ データベース名。
- [%p] – プロセス ID (使用可能な場合)。

Aurora PostgreSQL DB クラスターのクエリログ記録をオンにする

次のテーブルに示すパラメータの一部を設定することで、クエリ、ロック待ちのクエリ、チェックポイント、その他多くの詳細を含む、データベースアクティビティに関するより詳細な情報を収集できます。このトピックでは、クエリのログ記録に焦点を当てます。

パラメータ	デフォルト	[Description] (説明)
log_connections	–	成功した各接続を口グに記録します。このパラメータを log_disconnections で使用して接続チャーンを検出する方法については、「管理する Aurora PostgreSQL 接続チャーンとプーリング」を参照してください。
log_disconnections	–	各セッションの終了とその期間を記録します。このパラメータを log_connections で使用して接続チャーンを検出する方法については、「管理する Aurora PostgreSQL 接続チャーンとプーリング」を参照してください。
log_checkpoints	1	各チェックポイントを口グに記録します。
log_lock_waits	–	長期間にわたるロックの待機を口グに記録します。デフォルトでは、このパラメータは設定されていません。
log_min_duration_sample	–	(ms) ステートメントのサンプリングに関する最小実行時間を設定します。この値を超えるとステートメントがサンプリングされて口グに記録されます。サンプルサイズは、log_statement_sample_rate パラメータを使用して設定されます。
log_min_duration_statement	–	少なくとも指定された時間以上実行された SQL ステートメントはすべて口グに記録されます。デフォルトでは、このパラメータは設定されていません。このパラメータを有効にすると、最適化されていないクエリを見つけるために役立ちます。
log_statement	–	ログに記録するステートメントのタイプを設定します。デフォルトでは、このパラメータは設定されていませんが、all、ddl、または mod

パラメータ	デフォルト	[Description] (説明)
		に変更して、ログに記録する SQL ステートメントのタイプを指定できます。このパラメータの none 以外を指定する場合は、ログファイル内のパスワードが漏洩しないように、追加の手順も実行する必要があります。詳細については、「クエリのログ記録を使用する際のパスワード漏洩リスクの軽減」を参照してください。
log_statement_sample_rate	–	log_min_duration_sample で指定された時間を超えるステートメントがログに記録される割合で、0.0 から 1.0 の間の浮動小数点値で表されます。
log_statement_stats	–	累積処理のパフォーマンスの統計情報をサーバーログに書き込みます。

ログ記録を使用してパフォーマンスの低いクエリを見つける

SQL ステートメントとクエリをログに記録すると、パフォーマンスの悪いクエリを見つけるのに役立ちます。この機能を有効にするには、このセクションで説明されているとおり、log_statement および log_min_duration パラメータの設定を変更します。Aurora PostgreSQL DB クラスター、のクエリログ記録を有効にする前に、ログにパスワードが漏洩する可能性と、そのリスクを軽減する方法について知っておく必要があります。詳細については、「クエリのログ記録を使用する際のパスワード漏洩リスクの軽減」を参照してください。

log_statement および log_min_duration のパラメータに関する参照情報は、以下を参照してください。

log_statement

このパラメータは、ログに送信する SQL ステートメントのタイプを指定します。デフォルト値は none です。このパラメータを all、ddl、または mod に変更する場合は、ログにパスワードが漏洩するリスクを軽減するために、必ず推奨アクションを適用してください。詳細については、「クエリのログ記録を使用する際のパスワード漏洩リスクの軽減」を参照してください。

すべて

すべてのステートメントを記録します。この設定はデバッグ目的での使用を推奨します。

ddl

CREATE、ALTER、DROP などのすべてのデータ定義言語 (DDL) ステートメントを口グに記録します。

mod

データを変更する DDL ステートメントと、INSERT、UPDATE、DELETE などのデータ操作言語 (DML) ステートメントをすべて口グに記録します。

なし

SQL ステートメントはログに記録されません。ログにパスワードが漏れてしまうリスクを避けるため、この設定をお勧めします。

log_min_duration_statement

少なくとも指定された時間以上実行された SQL ステートメントはすべて口グに記録されます。デフォルトでは、このパラメータは設定されていません。このパラメータを有効にすると、最適化されていないクエリを見つけるために役立ちます。

–1–2147483647

ステートメントが口グに記録される実行時間のミリ秒 (ms) 数。

クエリの口グ記録を設定するには

これらのステップは、Aurora PostgreSQL DB クラスターがカスタム DB クラスターパラメータグループを使用していることを前提としています。

1. `log_statement` パラメータを `all` に設定します。以下の例に示しているのは、このパラメータ設定で `postgresql.log` ファイルに書き込まれる情報です。

```
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:LOG: statement:
 SELECT feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:LOG: QUERY
 STATISTICS
```

```
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:DETAIL: ! system
 usage stats:
! 0.017355 s user, 0.000000 s system, 0.168593 s elapsed
! [0.025146 s user, 0.000000 s system total]
! 36644 kB max resident size
! 0/8 [0/8] filesystem blocks in/out
! 0/733 [0/1364] page faults/reclaims, 0 [0] swaps
! 0 [0] signals rcvd, 0/0 [0/0] messages rcvd/sent
! 19/0 [27/0] voluntary/involuntary context switches
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:STATEMENT: SELECT
 feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-05 22:05:56 UTC:52.95.4.1(11335):postgres@labdb:[3639]:ERROR: syntax error
 at or near "ORDER" at character 1
2022-10-05 22:05:56 UTC:52.95.4.1(11335):postgres@labdb:[3639]:STATEMENT: ORDER BY
 s.confidence DESC;
----------------------- END OF LOG ----------------------
```

2. log_min_duration_statement パラメータを設定します。以下の例に示しているのは、パラメータを postgresql.log に設定したときに 1 ファイルに書き込まれる情報です。

   log_min_duration_statement パラメータで指定された期間を超えるクエリはログに記録されます。例を以下に示します。Aurora PostgreSQL DB クラスター のログファイルは Amazon RDS コンソールで表示できます。

```
2022-10-05 19:05:19 UTC:52.95.4.1(6461):postgres@labdb:[6144]:LOG: statement: DROP
 table comments;
2022-10-05 19:05:19 UTC:52.95.4.1(6461):postgres@labdb:[6144]:LOG: duration:
 167.754 ms
2022-10-05 19:08:07 UTC::@:[355]:LOG: checkpoint starting: time
2022-10-05 19:08:08 UTC::@:[355]:LOG: checkpoint complete: wrote 11 buffers
 (0.0%); 0 WAL file(s) added, 0 removed, 0 recycled; write=1.013 s, sync=0.006 s,
 total=1.033 s; sync files=8, longest=0.004 s, average=0.001 s; distance=131028 kB,
 estimate=131028 kB
----------------------- END OF LOG ----------------------
```

クエリのログ記録を使用する際のパスワード漏洩リスクの軽減

パスワードが漏洩しないように、`log_statement` を `none` に設定したままにしておくことをお勧めします。`log_statement` を `all`、`ddl`、または `mod` に設定した場合は、次の手順を 1 つ以上実行することをお勧めします。

- クライアントの場合は、機密情報を暗号化します。詳細については、PostgreSQL ドキュメントの「暗号化オプション」を参照してください。`CREATE` および `ALTER` ステートメントの `ENCRYPTED` (および`UNENCRYPTED`) オプションを使用してください。詳細については、PostgreSQL のドキュメントの「CREATE USER」を参照してください。
- Aurora PostgreSQL DB クラスター では 、PostgreSQL 監査 (pgAudit) 拡張機能をセットアップして使用します。この拡張機能は、ログに送信された CREATE および ALTER ステートメントの機密情報を編集します。詳細については、「pgAudit を使用してデータベースのアクティビティを記録する」を参照してください。
- CloudWatch ログへのアクセスを制限します。
- IAM など、より強力な認証メカニズムを使用してください。

AWS CloudTrail での Amazon Aurora API コールのモニタリング

AWS CloudTrail は、AWS アカウントの監査に役立つ AWS のサービスです。AWS CloudTrail は、AWS アカウントを作成すると、そのアカウントでオンに切り替わります。CloudTrail の詳細については、「AWS CloudTrail ユーザーガイド」を参照してください。

トピック

CloudTrail と Amazon Aurora の統合

すべての Amazon Aurora アクションが、CloudTrail によって ログ記録されます。CloudTrail では、Amazon Aurora のユーザー、ロール、または AWS のサービスによって実行されたアクションの記録を確認できます。

CloudTrail のイベント

CloudTrail が、Amazon Aurora のAPI コールをイベントとしてキャプチャします。イベント は、任意の出典からの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストパラメータなどに関する情報が含まれます。イベントには、Amazon RDS コンソールからの呼び出しと、Amazon RDS API 操作へのコード呼び出しが含まれます。

Amazon Aurora アクティビティは、[Event history] (イベント履歴) の CloudTrail イベントに記録されます。CloudTrail コンソールを使用して、AWS リージョンの過去 90 日間に記録された API アクティビティとイベントを表示できます。詳細については、CloudTrail イベント履歴でのイベントの表示を参照してください。

CloudTrail 証跡

AWS アカウントのイベント (Amazon Aurora のイベントなど) を継続的に記録するには、証跡を作成します。証跡とは、指定した Amazon S3 バケットにイベントを配信するという設定です。CloudTrail は、通常、アカウントアクティビティから 15 分以内にログファイルを配信します。

> **Note**
>
> 追跡を設定しない場合でも、CloudTrail コンソールの Event history (イベント履歴)で最新のイベントを表示できます。

AWS アカウントには、すべてのリージョンに適用される証跡と、1 つのリージョンに適用される証跡の 2 種類の証跡を作成できます。デフォルトでは、コンソールで追跡を作成するときに、追跡がすべての リージョンに適用されます。

さらに、CloudTrail ログで収集したイベントデータをより詳細に分析し、それに基づく対応するためにその他の AWS のサービスを設定できます。詳細については、次を参照してください。

- 証跡を作成するための概要
- CloudTrail がサポートされているサービスと統合
- CloudTrail の Amazon SNS 通知の設定
- CloudTrail ログファイルを複数のリージョンから受け取る、複数のアカウントから CloudTrail ログファイルを受け取る

Amazon Aurora ログファイルエントリ

CloudTrail のログファイルには、単一か複数のログエントリがあります。CloudTrail ログファイルは、パブリック API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

次は、`CreateDBInstance` アクションを示す CloudTrail ログエントリの例です。

```
{
    "eventVersion": "1.04",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2018-07-30T22:14:06Z",
    "eventSource": "rds.amazonaws.com",
    "eventName": "CreateDBInstance",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.15.42 Python/3.6.1 Darwin/17.7.0 botocore/1.10.42",
    "requestParameters": {
        "enableCloudwatchLogsExports": [
```

```
            "audit",
            "error",
            "general",
            "slowquery"
        ],
        "dBInstanceIdentifier": "test-instance",
        "engine": "mysql",
        "masterUsername": "myawsuser",
        "allocatedStorage": 20,
        "dBInstanceClass": "db.m1.small",
        "masterUserPassword": "****"
    },
    "responseElements": {
        "dBInstanceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance",
        "storageEncrypted": false,
        "preferredBackupWindow": "10:27-10:57",
        "preferredMaintenanceWindow": "sat:05:47-sat:06:17",
        "backupRetentionPeriod": 1,
        "allocatedStorage": 20,
        "storageType": "standard",
        "engineVersion": "8.0.28",
        "dbInstancePort": 0,
        "optionGroupMemberships": [
            {
                "status": "in-sync",
                "optionGroupName": "default:mysql-8-0"
            }
        ],
        "dBParameterGroups": [
            {
                "dBParameterGroupName": "default.mysql8.0",
                "parameterApplyStatus": "in-sync"
            }
        ],
        "monitoringInterval": 0,
        "dBInstanceClass": "db.m1.small",
        "readReplicaDBInstanceIdentifiers": [],
        "dBSubnetGroup": {
            "dBSubnetGroupName": "default",
            "dBSubnetGroupDescription": "default",
            "subnets": [
                {
                    "subnetAvailabilityZone": {"name": "us-east-1b"},
                    "subnetIdentifier": "subnet-cbfff283",
```

```
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1e"},
                    "subnetIdentifier": "subnet-d7c825e8",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1f"},
                    "subnetIdentifier": "subnet-6746046b",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1c"},
                    "subnetIdentifier": "subnet-bac383e0",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1d"},
                    "subnetIdentifier": "subnet-42599426",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1a"},
                    "subnetIdentifier": "subnet-da327bf6",
                    "subnetStatus": "Active"
                }
            ],
            "vpcId": "vpc-136a4c6a",
            "subnetGroupStatus": "Complete"
        },
        "masterUsername": "myawsuser",
        "multiAZ": false,
        "autoMinorVersionUpgrade": true,
        "engine": "mysql",
        "cACertificateIdentifier": "rds-ca-2015",
        "dbiResourceId": "db-ETDZIIXHEWY5N7GXVC4SH7H5IA",
        "dBSecurityGroups": [],
        "pendingModifiedValues": {
            "masterUserPassword": "****",
            "pendingCloudwatchLogsExports": {
                "logTypesToEnable": [
                    "audit",
                    "error",
```

```
                    "general",
                    "slowquery"
                ]
            }
        },
        "dBInstanceStatus": "creating",
        "publiclyAccessible": true,
        "domainMemberships": [],
        "copyTagsToSnapshot": false,
        "dBInstanceIdentifier": "test-instance",
        "licenseModel": "general-public-license",
        "iAMDatabaseAuthenticationEnabled": false,
        "performanceInsightsEnabled": false,
        "vpcSecurityGroups": [
            {
                "status": "active",
                "vpcSecurityGroupId": "sg-f839b688"
            }
        ]
    },
    "requestID": "daf2e3f5-96a3-4df7-a026-863f96db793e",
    "eventID": "797163d3-5726-441d-80a7-6eeb7464acd4",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```


前の例の userIdentity 要素に示すように、すべてのイベントまたはログエントリには、誰がリクエストを生成したかに関する情報が含まれています。この ID 情報は以下のことを確認するのに役立ちます。

- リクエストが、ルートと IAM ユーザー認証情報のどちらを使用して送信されたか。
- リクエストが、ロールとフェデレーティッドユーザーのどちらの一時的なセキュリティ認証情報を使用して送信されたか.
- リクエストが、別の AWS のサービスによって送信されたかどうか。

userIdentity の詳細については、「CloudTrail userIdentity 要素」を参照してください。CreateDBInstance およびその他の Amazon Aurora アクションの詳細については、「Amazon RDS API リファレンス」を参照してください。

データベースアクティビティストリームを使用した Amazon Aurora のモニタリング

データベースアクティビティストリームを使用すると、データベースアクティビティのストリームをほぼリアルタイムでモニタリングできます。

トピック

データベースアクティビティストリーミングの概要

Amazon Aurora データベース管理者として、データベースを保護し、コンプライアンスおよび規制要件を満たす必要があります。1 つの戦略は、データベースアクティビティストリーミングをモニタリングツールに統合することです。このようにして、Amazon Aurora クラスターでモニタリングを行い、監査アクティビティのアラームを設定します。

セキュリティの脅威は、外部と内部の両方です。内部の脅威から保護するために、データベースアクティビティストリーミング機能を設定して、データストリームへの管理者アクセスを制御できます。DBA には、ストリームの収集、送信、保存、および処理に対するアクセス権限がありません。

トピック

データベースアクティビティストリーミングの機能

Amazon Aurora では、データベースアクティビティストリーミングをクラスターレベルで開始します。クラスター内のすべての DB インスタンスで、データベースアクティビティストリーミングが有効になっています。

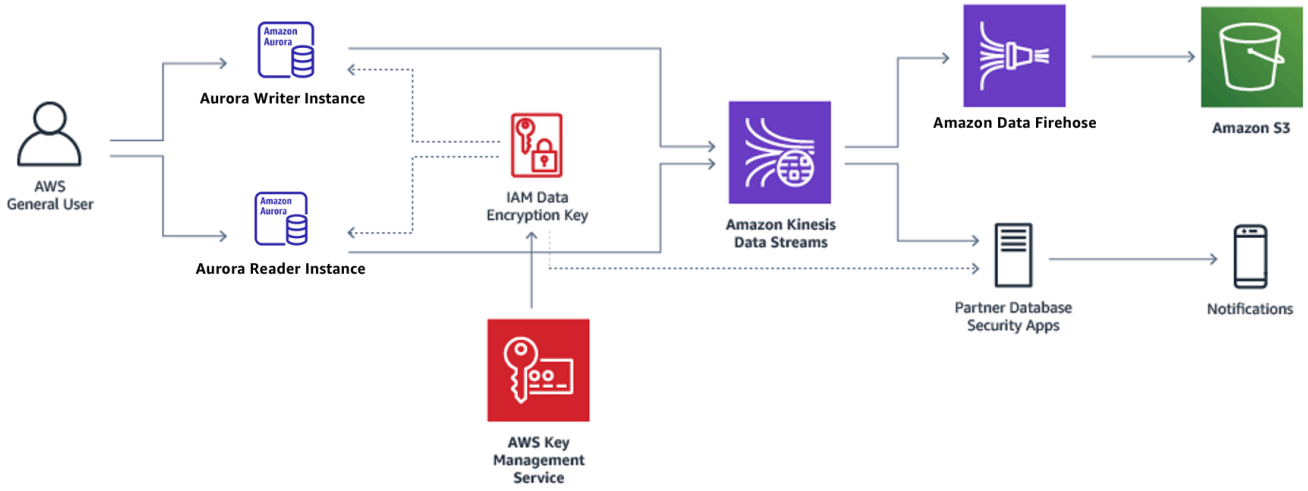
Aurora DB クラスターは、アクティビティをほぼリアルタイムで Amazon Kinesis データストリームにプッシュします。Kinesis ストリーミングが自動的に作成されます。Kinesis から、Amazon Data Firehose や AWS Lambda などの AWS サービスを設定して、ストリーミングを消費し、データを保存できます。

> **Important**
>
> Amazon Aurora のデータベースアクティビティストリーム機能は無料でご利用いただけますが、Amazon Kinesis のデータストリームに対しては課金されます。詳細については、「Amazon Kinesis Data Streams の料金表」を参照してください。

Aurora グローバルデータベースを使用する場合は、DB クラスターでデータベースアクティビティストリーミングを個別に開始してください。各クラスターは、独自の AWS リージョン 内の独自の Kinesis ストリーミングに監査データを配信します。アクティビティストリーミングは、フェイルオーバー中に異なる動作をしません。通常どおり、グローバルデータベースを監査し続けます。

コンプライアンス管理のためのアプリケーションを設定して、データベースアクティビティストリーミングを使用できます。Aurora PostgreSQLでは、コンプライアンスアプリケーションは、IBM の Security Guardium、Imperva の SecureSphere Database Audit and Protection などです。このようなアプリケーションでストリームを使用して、Aurora DB クラスターについてのアラートと監査アクティビティを生成できます。

次の図は、Amazon Data Firehose で設定された Aurora DB クラスターを示しています。



データベースアクティビティストリーミングの非同期および同期モード

次のモードのいずれかで、データベースセッションでデータベースアクティビティイベントを処理するように選択できます。

- 非同期モード - データベースセッションでアクティビティストリーミングイベントが生成されると、セッションは直ちに通常のアクティビティに戻ります。アクティビティストリーミングイベントは、バックグラウンドで永続的なレコードになります。バックグラウンドタスクでエラーが発生した場合は、RDS イベントが送信されます。このイベントは、アクティビティストリーミングのイベントレコードが失われた可能性がある時間枠のスタートと終了を示します。

  非同期モードでは、アクティビティストリーミングの精度よりもデータベースのパフォーマンスが優先されます。

  > **Note**
  >
  > 非同期モードは、Aurora PostgreSQL と Aurora MySQL の両方で使用できます。

- 同期モード - データベースセッションでアクティビティストリーミングイベントが生成されると、そのイベントが永続化されるまで、セッションによってその他のアクティビティはブロックされます。何らかの理由でイベントを永続化できない場合、データベースセッションは通常のアクティビティに戻ります。ただし、アクティビティストリーミングレコードがしばらくの間失われる可能性があることを示す RDS イベントが送信されます。システムが正常な状態に戻ったら、2 番目の RDS イベントが送信されます。

同期モードでは、データベースパフォーマンスよりもアクティビティストリーミングの精度が優先されます。

> **Note**
>
> 同期モードは、Aurora PostgreSQL で使用できます。Aurora MySQL では同期モードを使用できません。

データベースアクティビティストリームの要件と制限

Aurora では、データベースアクティビティストリームに、次の要件と制限があります。

- データベースアクティビティストリーミングには、Amazon Kinesis が必要です。
- 常に暗号化されているため、アクティビティストリーミングには AWS Key Management Service (AWS KMS) が必要です。
- Amazon Kinesis データストリームに追加の暗号化を適用することは、AWS KMS キーで暗号化済みのデータベースアクティビティストリーミングと互換性がありません。
- DB クラスターレベルでデータベースアクティビティストリーミングを開始します。DB インスタンスをクラスターに追加する場合、インスタンスでアクティビティストリーミングをスタートする必要はありません。アクティビティストリーミングは自動的に監査されます。
- Aurora グローバルデータベースでは、各 DB クラスターでアクティビティストリーミングを個別に開始してください。各クラスターは、独自の AWS リージョン 内の独自の Kinesis ストリーミングに監査データを配信します。
- Aurora PostgreSQL では、アップグレード前に必ずデータベースアクティビティストリームを停止してください。アップグレードの完了後に、データベースアクティビティストリームを開始できます。

リージョンとバージョンの可用性

利用できる機能とそのサポートは、各 Aurora データベースエンジンの特定のバージョン、および AWS リージョン によって異なります。Aurora とデータベースアクティビティストリーミングで使用できるバージョンとそのリージョンの詳細については、「データベースアクティビティストリームでサポートされているリージョンと Aurora DB エンジン」を参照してください。

データベースアクティビティストリーミングでサポートされる DB インスタンスクラス

Aurora MySQL では、次の DB インスタンスクラスでデータベースアクティビティストリーミングを使用できます。

- db.r7g.*large
- db.r6g.*large
- db.r6i.*large
- db.r5.*large
- db.x2g.*

Aurora PostgreSQL では、次の DB インスタンスクラスでデータベースアクティビティストリーミングを使用できます。

- db.r7g.*large
- db.r6g.*large
- db.r6i.*large
- db.r6id.*large
- db.r5.*large
- db.r4.*large
- db.x2g.*

Aurora MySQL データベースアクティビティストリーミングのネットワーク前提条件

次のセクションでは、データベースアクティビティストリーミングで使用するための仮想プライベートクラウド (VPC) を設定する方法について説明します。

> **Note**
>
> Aurora MySQL ネットワークの前提条件は、以下のエンジンバージョンに適用されます。
>
> - Aurora MySQL バージョン 2 から 2.11.3 まで
> - Aurora MySQL バージョン 2.12.0

- Aurora MySQL バージョン 3 から 3.04.2 まで

トピック

- AWS KMS エンドポイントの前提条件
- パブリックアベイラビリティの前提条件
- プライベートアベイラビリティの前提条件

AWS KMS エンドポイントの前提条件

アクティビティストリーミングを使用する Aurora MySQL クラスターのインスタンスは、AWS KMS エンドポイントにアクセスできる必要があります。Aurora MySQL クラスターのデータベースアクティビティストリーミングを有効にする前に、この要件が満たされていることを確認してください。Aurora クラスターが一般公開されている場合、この要件は自動的に満たされます。

Important

Aurora MySQL DB クラスターが AWS KMS エンドポイントにアクセスできない場合、アクティビティストリーミングは停止します。この場合、Aurora は RDS イベントを使用してこの問題について通知します。

パブリックアベイラビリティの前提条件

Aurora DB クラスターがパブリックの場合は、次の要件を満たしている必要があります。

- AWS Management Console クラスターの詳細ページにある [Publicly Accessible] (パブリックアクセス可能) を [Yes] (はい) にします。
- DB クラスターは Amazon VPC パブリックサブネットに存在しています。パブリックアクセス可能な DB インスタンスの詳細については、「VPC 内の DB クラスターの使用」を参照してください。Amazon VPC のパブリックサブネットの詳細については、「VPC とサブネット」を参照してください。

プライベートアベイラビリティの前提条件

Aurora DB クラスターが VPC パブリックサブネットにあり、パブリックアクセス可能でない場合は、プライベートです。クラスターをプライベートにしておき、データベースアクティビティストリーミングで使用するには、次のオプションがあります。

- VPC でネットワークアドレス変換 (NAT) を設定します。詳細については、「NAT ゲートウェイ」を参照してください。
- VPC で AWS KMS エンドポイントを作成します。設定が簡単であるため、このオプションを推奨しています。

VPC で AWS KMS エンドポイントを作成する方法

1. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
2. ナビゲーションペインで、[Endpoints] (エンドポイント) を選択します。
3. [エンドポイントの作成] を選択します。

   [Create Endpoint] (エンドポイントの作成) ページが表示されます。
4. 以下の操作を実行します。
   - [サービスカテゴリ] で、[AWS サービス] を選択します。
   - [Service Name] (サービス名) で、[com.amazonaws.***region***.kms] を選択します。この場合 *region* には、ご使用のクラスターが配置されている AWS リージョン が表示されています。
   - [VPC] では、クラスターが配置されている VPC を選択します。
5. [エンドポイントの作成] を選択します。

VPC エンドポイントの設定の詳細については、「VPC エンドポイント」を参照してください。

データベースアクティビティストリーミングのスタート

Aurora DB クラスターのすべてのインスタンスのデータベースアクティビティをモニタリングするには、クラスターレベルでアクティビティストリーミングをスタートします。クラスターに追加した DB インスタンスも自動的にモニタリングされます。Aurora グローバルデータベースを使用する場合は、DB クラスターでデータベースアクティビティストリーミングを個別に開始してください。各クラスターは、独自の AWS リージョン 内の独自の Kinesis ストリーミングに監査データを配信します。

アクティビティストリームを開始すると、監査ポリシーで設定したデータベースアクティビティイベントごとに、アクティビティストリームイベントが生成されます。アクセスイベントは CONNECT、SELECT などの SQL コマンドから生成されます。変更イベントは CREATE、INSERT などの SQL コマンドから生成されます。

コンソール

データベースアクティビティストリーミングをスタートするには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで [データベース] を選択します。
3. アクティビティストリームを有効にする DB クラスターを選択します。
4. [アクション] で [アクティビティストリーミングの開始] を選択します。

   [データベースアクティビティストリーミングの開始: *##*] ウィンドウが表示されます。ここで、*name* は DB クラスターです。
5. 以下の設定を入力します。

   - [AWS KMS key] では、 AWS KMS keys のリストからキーを選択します。

     > **Note**
     >
     > Aurora MySQL クラスターが KMS キーにアクセスできない場合は、「Aurora MySQL データベースアクティビティストリーミングのネットワーク前提条件」の手順に従って、まずそのようなアクセスを有効にします。

     Aurora は、KMS キーを使用してキーを暗号化し、それによってデータベースアクティビティを暗号化します。デフォルトキー以外の KMS キーを選択します。暗号化キーと AWS KMS の詳細については、AWS Key Management Service デベロッパーガイドの「AWS Key Management Service とは？」を参照してください。
   - [データベースアクティビティストリーミングモード] で、[非同期] または [同期] を選択します。

> **Note**
>
> この選択は Aurora PostgreSQL にのみ適用されます。Aurora MySQL では、非同期モードのみを使用できます。

- [すぐに適用] を選択します。

  [直ちに] を選択する場合直ちにとすると、DB クラスターがすぐに再起動します。を選択すると次のメンテナンス時間中とすると、DB クラスターはすぐには再起動しません。この場合、データベースアクティビティストリーミングは、次のメンテナンスウィンドウまでスタートされません。

6. [Start database activity stream] (データベースアクティビティストリームを開始) を選択します。

   DB クラスターのステータスは、アクティビティストリームが開始していることを示します。

> **Note**
>
> `You can't start a database activity stream in this configuration` というエラーが表示された場合は、データベースアクティビティストリーミングでサポートされる DB インスタンスクラス で DB クラスター がサポートされているインスタンスクラスを使用しているかを確認してください。

AWS CLI

DB クラスターのデータベースアクティビティストリームを開始するには、AWS CLI コマンド start-activity-stream を使用して DB クラスターを設定します。

- `--resource-arn` *`arn`* - DB クラスターの Amazon リソースネーム (ARN) を指定します。
- `--mode` *`sync-or-async`* - 同期 (sync) または非同期 (async) モードを指定します。Aurora PostgreSQL では、いずれかの値を選択できます。Aurora MySQL の場合は、`async` を指定します。
- `--kms-key-id` *`key`* - データベースアクティビティストリーミング内のメッセージを暗号化するための KMS キー識別子を指定します。AWS KMS キー識別子は、キー ARN、キー ID、エイリアス ARN、または AWS KMS key のエイリアス名です。

次の例では、非同期モードで DB クラスターのデータベースアクティビティストリームを開始します。

Linux、macOS、Unix の場合:

```
aws rds start-activity-stream \
    --mode async \
    --kms-key-id my-kms-key-arn \
    --resource-arn my-cluster-arn \
    --apply-immediately
```

Windows の場合:

```
aws rds start-activity-stream ^
    --mode async ^
    --kms-key-id my-kms-key-arn ^
    --resource-arn my-cluster-arn ^
    --apply-immediately
```

RDS API

DB クラスターのデータベースアクティビティストリームを開始するには、StartActivityStream オペレーションを使用してクラスターを設定します。

以下のパラメータを使用してアクションを呼び出します。

- `Region`
- `KmsKeyId`
- `ResourceArn`
- `Mode`

データベースアクティビティストリーミングのステータスの取得

アクティビティストリームのステータスを取得するには、コンソールまたは AWS CLI を使用します。

コンソール

データベースアクティビティストリーミングのステータスを取得するには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択し、DB クラスターのリンクを選択します。
3. [設定] タブを選択して、ステータスを取得する [データベースアクティビティストリーミング] をオンにします。

AWS CLI

describe-db-clusters CLI リクエストに対するレスポンスとして、DB クラスターのアクティビティストリーム設定を取得できます。

次の例は、*my-cluster* を説明します。

```
aws rds --region my-region describe-db-clusters --db-cluster-identifier my-cluster
```

以下は JSON レスポンスの例です。次のフィールドが表示されます。

- `ActivityStreamKinesisStreamName`
- `ActivityStreamKmsKeyId`
- `ActivityStreamStatus`
- `ActivityStreamMode`
-

これらのフィールドは Aurora PostgreSQL や Aurora MySQL でも同じです。ただし `ActivityStreamMode` は、常に Aurora MySQL の場合 `async` になり、Aurora PostgreSQL の場合は `sync` または `async` になります。

```
{
    "DBClusters": [
        {
      "DBClusterIdentifier": "my-cluster",
            ...
            "ActivityStreamKinesisStreamName": "aws-rds-das-cluster-
A6TSYXITZCZXJHIRVFUBZ5LTWY",
            "ActivityStreamStatus": "starting",
```

```
            "ActivityStreamKmsKeyId": "12345678-abcd-efgh-ijkl-bd041f170262",
            "ActivityStreamMode": "async",
            "DbClusterResourceId": "cluster-ABCD123456"
            ...
        }
    ]
}
```

RDS API

DescribeDBClusters オペレーションに対するレスポンスとして、DB クラスターのアクティビティストリーミングの設定を取得できます。

データベースアクティビティストリーミングの停止

アクティビティストリーミングを停止するには、コンソールまたは AWS CLI を使用します。

DB クラスターを削除すると、アクティビティストリームが停止し、基になる Amazon Kinesis ストリームが自動的に削除されます。

コンソール

アクティビティストリーミングを停止するには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. データベースアクティビティストリーミングを停止する DB クラスターを選択します。
4. [アクション] で [アクティビティストリーミングの停止] を選択します。[データベースアクティビティストリーミング] ウィンドウが表示されます。

    a. [すぐに適用] を選択します。

        [直ちに] を選択する場合直ちにとすると、DB クラスターがすぐに再起動します。を選択すると次のメンテナンス時間中とすると、DB クラスターはすぐには再起動しません。この場合、データベースアクティビティストリーミングは、次のメンテナンスウィンドウまで停止しません。

    b. [続行] を選択します。

AWS CLI

DB クラスターのデータベースアクティビティストリーミングを停止するには、AWS CLI コマンドの stop-activity-stream を使用して DB クラスターを設定します。DB クラスターのの AWS リージョンを識別するには、--region パラメータを指定します。--apply-immediately パラメータはオプションです。

Linux、macOS、Unix の場合:

```
aws rds --region MY_REGION \
    stop-activity-stream \
    --resource-arn MY_CLUSTER_ARN \
    --apply-immediately
```

Windows の場合:

```
aws rds --region MY_REGION ^
    stop-activity-stream ^
    --resource-arn MY_CLUSTER_ARN ^
    --apply-immediately
```

RDS API

DB クラスターのデータベースアクティビティストリーミングを停止するには、StopActivityStream オペレーションを使用して、クラスターを設定します。DB クラスターのの AWS リージョンを識別するには、Region パラメータを指定します。ApplyImmediately パラメータはオプションです。

データベースアクティビティストリーミングのモニタリング

データベースアクティビティストリーミングは、アクティビティをモニタリングして報告します。アクティビティのストリーミングは、収集後、Amazon Kinesis に送信されます。Kinesis から、アクティビティストリーミングをモニタリングしたり、他のサービスやアプリケーションがアクティビティストリーミングを使用して詳細な分析を行うことができます。基礎となる Kinesis ストリーム名は、AWS CLI コマンドの describe-db-clusters または RDS API DescribeDBClusters オペレーションを使用して検索できます。

Aurora は、Kinesis ストリーミングを次のように管理します。

- Aurora は、24 時間の保存期間の Kinesis ストリーミングを自動的に作成します。

- Aurora は、必要に応じて Kinesis ストリーミングをスケールします。
- データベースアクティビティストリーミングを停止したり、DB クラスターを削除したりすると、Aurora によって Kinesis ストリームが削除されます。

以下のカテゴリのアクティビティがモニタリングされ、アクティビティストリーミングの監査ログに追加されます。

- SQL コマンド - すべての SQL コマンドに加えて、準備済みステートメント、組み込み関数、および PL/SQL の関数も監査されます。ストアドプロシージャへの呼び出しが監査されます。ストアドプロシージャまたは関数内で発行された SQL ステートメントも監査されます。
- 他のデータベース情報 - モニタリングされるアクティビティには、完全な SQL ステートメント、DML コマンドから影響を受ける行の行数、アクセスされたオブジェクト、および一意のデータベース名が含まれます。Aurora PostgreSQL の場合、データベースアクティビティストリーミングは、バインド可変とストアドプロシージャパラメータもモニタリングします。

  > **Important**
  >
  > 各ステートメントの完全な SQL テキストは、機密データを含むアクティビティストリーミング監査ログに表示されます。ただし、Aurora が次の SQL ステートメントのようにコンテキストから判断できる場合、データベースユーザーのパスワードは訂正されます。
  >
  > ```
  > ALTER ROLE role-name WITH password
  > ```

- 接続情報 - モニタリングされるアクティビティには、セッションとネットワークの情報、サーバープロセス ID、および終了コードなどがあります。

DB インスタンスのモニタリング中にアクティビティストリーミングに障害が発生した場合は、RDS イベントを通じて通知されます。

トピック

- Kinesis からのアクティビティストリーミングへのアクセス
- 監査ログの内容と例
- databaseActivityEventList JSON 配列
- AWS SDK を使用したデータベースアクティビティストリーミングの処理

Kinesis からのアクティビティストリーミングへのアクセス

DB クラスターのアクティビティストリーミングを有効にすると、Kinesis ストリーミングが作成されます。データベースのアクティビティは、Kinesis からリアルタイムでモニタリングできます。データベースのアクティビティを詳細に分析するには、Kinesis ストリーミングをコンシューマーアプリケーションに接続します。また、IBM の Security Guardium または Imperva の SecureSphere Database Audit and Protection などのコンプライアンス管理アプリケーションにストリーミングを接続することもできます。

Kinesis ストリームには、RDS コンソールまたは Kinesis コンソールからアクセスできます。

RDS コンソールを使用して、Kinesis からアクティビティストリーミングにアクセスするには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. アクティビティストリームを開始する DB クラスターを選択します。
4. [設定] を選択します。
5. [Database activity stream] (データベースアクティビティストリーム) で、[Kinesis stream] (Kinesis ストリーム) の下のリンクを選択します。
6. データベースアクティビティの観察を開始するには、Kinesis コンソールで [Monitoring] (モニタリング) を選択します。

Kinesis コンソールを使用して、Kinesis からアクティビティストリーミングにアクセスするには

1. Kinesis コンソール (https://console.aws.amazon.com/kinesis) を開きます。
2. Kinesis ストリーミングのリストからアクティビティストリーミングを選択します。

   アクティビティストリーミングの名前には、プレフィックス `aws-rds-das-cluster-` が付き、その後に DB クラスターのリソース ID が続きます。次に例を示します。

   ```
   aws-rds-das-cluster-NHVOV4PCLWHGF52NP
   ```

   Amazon RDS コンソールを使用して DB クラスターのリソース ID を検索するには、データベースのリストから DB クラスターを選択した上で、[設定] タブを選択します。

AWS CLI を使用して、アクティビティストリーミングの Kinesis ストリームの完全な名前を検索するには、describe-db-clusters CLI リクエストを使用し、そのレスポンスに含まれる `ActivityStreamKinesisStreamName` の値を書き留めます。

3. データベースアクティビティの観察をスタートするには、[モニタリング] を選択します。

Amazon Kinesis の使用の詳細については、「Amazon Kinesis Data Streams とは」を参照してください。

監査ログの内容と例

モニタリングされるイベントは、データベースアクティビティストリーミングでは JSON 文字列として表されます。この構造は、`DatabaseActivityMonitoringRecord` を含む JSON オブジェクトで構成されます。このオブジェクトには、アクティビティイベントの `databaseActivityEventList` 配列が含まれます。

トピック

- アクティビティストリーミングの監査ログの例
- DatabaseActivityMonitoringRecords JSON オブジェクト
- databaseActivityEvents JSON オブジェクト

アクティビティストリーミングの監査ログの例

以下に、アクティビティイベントレコードの復号されたサンプルの JSON 監査ログを示します。

Example Aurora PostgreSQL CONNECT SQL ステートメント のアクティビティイベントレコード

次のアクティビティイベントレコードは、psql クライアント (`clientApplication`) による `CONNECT` SQL ステートメント (`command`) を使用したログインを示しています。

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":
    {
      "type":"DatabaseActivityMonitoringRecord",
      "clusterId":"cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
      "instanceId":"db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
      "databaseActivityEventList":[
        {
```

```
          "startTime": "2019-10-30 00:39:49.940668+00",
          "logTime": "2019-10-30 00:39:49.990579+00",
          "statementId": 1,
          "substatementId": 1,
          "objectType": null,
          "command": "CONNECT",
          "objectName": null,
          "databaseName": "postgres",
          "dbUserName": "rdsadmin",
          "remoteHost": "172.31.3.195",
          "remotePort": "49804",
          "sessionId": "5ce5f7f0.474b",
          "rowCount": null,
          "commandText": null,
          "paramList": [],
          "pid": 18251,
          "clientApplication": "psql",
          "exitCode": null,
          "class": "MISC",
          "serverVersion": "2.3.1",
          "serverType": "PostgreSQL",
          "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
          "serverHost": "172.31.3.192",
          "netProtocol": "TCP",
          "dbProtocol": "Postgres 3.0",
          "type": "record",
          "errorMessage": null
        }
      ]
    },
   "key":"decryption-key"
}
```

Example Aurora MySQL CONNECT SQL ステートメントのアクティビティイベントレコード

次のアクティビティイベントレコードは、mysql クライアント (CONNECT) による command SQL ステートメント (clientApplication) を使用したログインを示しています。

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
```

```
    {
      "logTime":"2020-05-22 18:07:13.267214+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"rdsadmin",
      "databaseName":"",
      "remoteHost":"localhost",
      "remotePort":"11053",
      "command":"CONNECT",
      "commandText":"",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"",
      "statementId":0,
      "substatementId":1,
      "exitCode":"0",
      "sessionId":"725121",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:13.267207+00",
      "endTime":"2020-05-22 18:07:13.267213+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"MAIN"
    }
  ]
}
```

Example Aurora PostgreSQL CREATE TABLE ステートメントのアクティビティイベントレコード

次の例は、Aurora PostgreSQL の CREATE TABLE イベントを示しています。

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":
    {
      "type":"DatabaseActivityMonitoringRecord",
```

```
      "clusterId":"cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
      "instanceId":"db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
      "databaseActivityEventList":[
        {
          "startTime": "2019-05-24 00:36:54.403455+00",
          "logTime": "2019-05-24 00:36:54.494235+00",
          "statementId": 2,
          "substatementId": 1,
          "objectType": null,
          "command": "CREATE TABLE",
          "objectName": null,
          "databaseName": "postgres",
          "dbUserName": "rdsadmin",
          "remoteHost": "172.31.3.195",
          "remotePort": "34534",
          "sessionId": "5ce73c6f.7e64",
          "rowCount": null,
          "commandText": "create table my_table (id serial primary key, name
 varchar(32));",
          "paramList": [],
          "pid": 32356,
          "clientApplication": "psql",
          "exitCode": null,
          "class": "DDL",
          "serverVersion": "2.3.1",
          "serverType": "PostgreSQL",
          "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
          "serverHost": "172.31.3.192",
          "netProtocol": "TCP",
          "dbProtocol": "Postgres 3.0",
          "type": "record",
          "errorMessage": null
        }
      ]
    },
   "key":"decryption-key"
}
```

Example Aurora MySQL CREATE TABLE ステートメントのアクティビティイベントレコード

次の例は、Aurora MySQL の CREATE TABLE ステートメントを示しています。オペレーションは、2 つの個別のイベントレコードとして表されます。1 つのイベントに "class":"MAIN" があります。他方のイベントには、"class":"AUX" があります。メッセージは任意の順序で到着する

可能性があります。logTime イベントの MAIN フィールドは、常に対応する logTime イベントの AUX フィールドよりも前にあります。

次の例は、class の値が MAIN のイベントを示しています。


```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:07:12.250221+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"QUERY",
      "commandText":"CREATE TABLE test1 (id INT)",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65459278,
      "substatementId":1,
      "exitCode":"0",
      "sessionId":"725118",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:12.226384+00",
      "endTime":"2020-05-22 18:07:12.250222+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"MAIN"
    }
  ]
}
```

次の例は、class の値 が AUX を持つ対応するイベントを示しています。

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:07:12.247182+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"CREATE",
      "commandText":"test1",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65459278,
      "substatementId":2,
      "exitCode":"",
      "sessionId":"725118",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:12.226384+00",
      "endTime":"2020-05-22 18:07:12.247182+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"AUX"
    }
  ]
}
```

Example Aurora PostgreSQL SELECT ステートメントのアクティビティイベントレコード

次の例は、 SELECT イベントを示しています。


```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":
    {
      "type":"DatabaseActivityMonitoringRecord",
      "clusterId":"cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
      "instanceId":"db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
      "databaseActivityEventList":[
        {
          "startTime": "2019-05-24 00:39:49.920564+00",
          "logTime": "2019-05-24 00:39:49.940668+00",
          "statementId": 6,
          "substatementId": 1,
          "objectType": "TABLE",
          "command": "SELECT",
          "objectName": "public.my_table",
          "databaseName": "postgres",
          "dbUserName": "rdsadmin",
          "remoteHost": "172.31.3.195",
          "remotePort": "34534",
          "sessionId": "5ce73c6f.7e64",
          "rowCount": 10,
          "commandText": "select * from my_table;",
          "paramList": [],
          "pid": 32356,
          "clientApplication": "psql",
          "exitCode": null,
          "class": "READ",
          "serverVersion": "2.3.1",
          "serverType": "PostgreSQL",
          "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
          "serverHost": "172.31.3.192",
          "netProtocol": "TCP",
          "dbProtocol": "Postgres 3.0",
          "type": "record",
          "errorMessage": null
        }
      ]
    },
```

```
   "key":"decryption-key"
}
```

```
{
    "type": "DatabaseActivityMonitoringRecord",
    "clusterId": "",
    "instanceId": "db-4JCWQLUZVFYP7DIWP6JVQ77O3Q",
    "databaseActivityEventList": [
        {
            "class": "TABLE",
            "clientApplication": "Microsoft SQL Server Management Studio - Query",
            "command": "SELECT",
            "commandText": "select * from [testDB].[dbo].[TestTable]",
            "databaseName": "testDB",
            "dbProtocol": "SQLSERVER",
            "dbUserName": "test",
            "endTime": null,
            "errorMessage": null,
            "exitCode": 1,
            "logTime": "2022-10-06 21:24:59.9422268+00",
            "netProtocol": null,
            "objectName": "TestTable",
            "objectType": "TABLE",
            "paramList": null,
            "pid": null,
            "remoteHost": "local machine",
            "remotePort": null,
            "rowCount": 0,
            "serverHost": "172.31.30.159",
            "serverType": "SQLSERVER",
            "serverVersion": "15.00.4073.23.v1.R1",
            "serviceName": "sqlserver-ee",
            "sessionId": 62,
            "startTime": null,
            "statementId": "0x03baed90412f564fad640ebe51f89b99",
            "substatementId": 1,
            "transactionId": "4532935",
            "type": "record",
            "engineNativeAuditFields": {
                "target_database_principal_id": 0,
                "target_server_principal_id": 0,
                "target_database_principal_name": "",
                "server_principal_id": 2,
```

```
                "user_defined_information": "",
                "response_rows": 0,
                "database_principal_name": "dbo",
                "target_server_principal_name": "",
                "schema_name": "dbo",
                "is_column_permission": true,
                "object_id": 581577110,
                "server_instance_name": "EC2AMAZ-NFUJJNO",
                "target_server_principal_sid": null,
                "additional_information": "",
                "duration_milliseconds": 0,
                "permission_bitmask": "0x00000000000000000000000000000001",
                "data_sensitivity_information": "",
                "session_server_principal_name": "test",
                "connection_id": "AD3A5084-FB83-45C1-8334-E923459A8109",
                "audit_schema_version": 1,
                "database_principal_id": 1,
                "server_principal_sid":
 "0x0105000000000005150000000bdc2795e2d0717901ba6998cf4010000",
                "user_defined_event_id": 0,
                "host_name": "EC2AMAZ-NFUJJNO"
            }
        }
    ]
}
```

Example Aurora MySQL SELECT ステートメントのアクティビティイベントレコード

次の例は、SELECT イベントを示しています。

次の例は、class の値が MAIN のイベントを示しています。

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:29:57.986467+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
```

```
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"QUERY",
      "commandText":"SELECT * FROM test1 WHERE id < 28",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65469218,
      "substatementId":1,
      "exitCode":"0",
      "sessionId":"726571",
      "rowCount":2,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:29:57.986364+00",
      "endTime":"2020-05-22 18:29:57.986467+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"MAIN"
    }
  ]
}
```

次の例は、class の値 が AUX を持つ対応するイベントを示しています。

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:29:57.986399+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"READ",
```

```
      "commandText":"test1",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65469218,
      "substatementId":2,
      "exitCode":"",
      "sessionId":"726571",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:29:57.986364+00",
      "endTime":"2020-05-22 18:29:57.986399+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"AUX"
    }
  ]
}
```

DatabaseActivityMonitoringRecords JSON オブジェクト

データベースアクティビティイベントレコードは、次の情報を含む JSON オブジェクトにあります。

JSON フィールド	データ型	説明
type	文字列	JSON レコードのタイプ。値は DatabaseActivityMonitoringRecords です。
version	文字列	データベースアクティビティモニタリングレコードのバージョン。 生成されたデータベースアクティビティレコードのバージョンは、次のように、DB クラスターのエンジンのバージョンによって異なります。

JSON フィールド	データ型	説明
		• バージョン 1.1 のデータベースアクティビティレコードは、エンジンバージョン 10.10 以降のマイナーバージョン、およびエンジンバージョン 11.5 以降を実行する Aurora PostgreSQL DB クラスターに対して生成されます。 • バージョン 1.0 のデータベースアクティビティレコードは、エンジンバージョン 10.7 および 11.4 を実行している Aurora PostgreSQL DB クラスターに対して生成されます。 次のフィールドは、すべてバージョン 1.0 とバージョン 1.1 の両方にあります。ただし、明記されている場合を除きます。
databaseActivityEvents	string	アクティビティイベントを含む JSON オブジェクト。
キー	string	databaseActivityEventList の復号に使用する暗号化キー。

databaseActivityEvents JSON オブジェクト

`databaseActivityEvents` JSON オブジェクトには、次の情報が含まれています。

JSON レコードの最上位フィールド

監査ログの各イベントは、JSON 形式のレコード内にラップされます。このレコードには、次のフィールドが含まれます。

type

このフィールドは常に値 `DatabaseActivityMonitoringRecords` を持ちます。

バージョン

このフィールドは、データベースアクティビティストリーミングデータプロトコルまたはコントラクトのバージョンを表します。これは、使用可能なフィールドを定義します。

バージョン 1.0 は、Aurora PostgreSQL バージョン 10.7 および 11.4 の元のデータアクティビティストリーミングのサポートを表します。バージョン 1.1 は、Aurora PostgreSQL バージョン 10.10 以降、および Aurora PostgreSQL 11.5 以降のデータアクティビティストリーミングのサポートを表します。バージョン 1.1 には、追加のフィールド `errorMessage` と `startTime` が含まれています。バージョン 1.2 は、Aurora MySQL 2.08 以降のデータアクティビティストリーミングのサポートを表します。バージョン 1.2 には、追加のフィールド `endTime` と `transactionId` が含まれています。

databaseActivityEvents

1 つ以上のアクティビティイベントを表す暗号化された文字列。これは、base64 バイト配列として表されます。文字列を復号すると、このセクションの例に示すフィールドを持つ JSON 形式のレコードが生成されます。

key

`databaseActivityEvents` 文字列の暗号化に使用される暗号化されたデータキー。これは、データベースアクティビティストリーミングをスタートしたときに指定した AWS KMS key と同じです。

以下の例は、このレコードの形式を示しています。

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":"encrypted audit records",
  "key":"encrypted key"
}
```

`databaseActivityEvents` フィールドの内容を復号化するには、次のステップを実行します。

1. データベースアクティビティストリーミングをスタートするときに指定した KMS キーを使用して、key JSON フィールドの値を復号します。これにより、データ暗号化キーがクリアテキストで返されます。
2. `databaseActivityEvents` JSON フィールドの値を Base64 デコードして、監査ペイロードの暗号化テキストをバイナリ形式で取得します。

3. 初期のステップでデコードしたデータ暗号化キーを使用して、バイナリ暗号文を復号化します。
4. 復号化されたペイロードを解凍します。
   - 暗号化されたペイロードは、databaseActivityEvents フィールドにあります。
   - databaseActivityEventList フィールドには、監査レコードの配列が含まれます。配列内の type フィールドには、 record または heartbeat を使用できます。

監査ログのアクティビティイベントレコードは、次の情報を含む JSON オブジェクトです。

JSON フィールド	データ型	説明
type	文字列	JSON レコードのタイプ。値は DatabaseActivityMonitoringRecord です。
clusterId	文字列	DB クラスターリソース識別子。これは DB クラスター属性 DbClusterResourceId に対応します。
instanceId	文字列	DB インスタンスのリソース識別子。DB インスタンス属性 DbiResourceId に対応します。
databaseActivityEventList	文字列	アクティビティ監査レコードまたはハートビートメッセージの配列。

databaseActivityEventList JSON 配列

監査ログのペイロードは、暗号化された databaseActivityEventList JSON 配列です。以下の表に、監査ログの復号された DatabaseActivityEventList 配列内の各アクティビティイベントのフィールドをアルファベット順に示します。Aurora PostgreSQL または Aurora MySQL を使用するかどうかによって、フィールドは異なります。データベースエンジンに適用される表を参照してください。

> **Important**
>
> イベントの構造は変わる場合があります。Aurora では、将来、アクティビティイベントに新しいフィールドが追加される可能性があります。JSON データを分析するアプリケーションでは、コードが未知のフィールド名に対して無視または適切なアクションを実行できることを確認します。

Aurora PostgreSQL の databaseActivityEventList フィールド

フィールド	データ型	説明
class	文字列	アクティビティイベントのクラス。Aurora PostgreSQL の有効な値は以下のとおりです。 • ALL • CONNECT - 接続イベントまたは切断イベント。 • DDL - ROLE クラスのステートメントのリストに含まれていない DDL ステートメント。 • FUNCTION - 関数の呼び出し、または DO ブロック。 • MISC - さまざまなコマンド (例: DISCARD、FETCH、CHECKPOINT 、VACUUM)。 • NONE • READ - SELECT ステートメントまたは COPY ステートメント (ソースがリレーションまたはクエリの場合)。 • ROLE - ロールと特権に関するステートメント (例: GRANT、REVOKE、CREATE、ALTER、DROP、ROLE)。 • WRITE - INSERT、UPDATE、DELETE、TRUNCATE、COPY ステートメント (送信先がリレーションの場合)。
clientApplication	文字列	クライアントのレポートどおりにクライアントが接続に使用していたアプリケーション。クライアントはこの情報を指定する必要はないため、値は null でも問題ありません。
command	文字列	SQL コマンドの名前 (コマンドの詳細は含まない)。
commandText	文字列	ユーザーによって渡された実際の SQL ステートメント。Aurora PostgreSQL の場合、値は元の SQL ステートメントと同じです。このフィールドは、接続レコードまたは切断レコードを除くすべてのタイプのレコードに使用されます。この場合、値は null です。

フィールド	データ型	説明
		Important 各ステートメントの完全な SQL テキストは、機密データを含むアクティビティストリーミング監査ログに表示されます。ただし、Aurora が次の SQL ステートメントのようにコンテキストから判断できる場合、データベースユーザーのパスワードは編集されます。 `ALTER ROLE role-name WITH password`
databaseName	文字列	ユーザーが接続したデータベース。
dbProtocol	文字列	データベースプロトコル (例: Postgres 3.0)。
dbUserName	文字列	クライアントが認証したデータベースユーザー。

フィールド	データ型	説明
errorMessage (バージョン 1.1 のデータベースアクティビティレコードのみ)	文字列	エラーがあった場合、このフィールドには DB サーバーによって生成されるはずのエラーメッセージが表示されます。エラーにならなかった通常のステートメントの場合、errorMessage の値は null です。 エラーは、重要度が ERROR 以上の、クライアントで表示される PostgreSQL エラーログイベントを生成するアクティビティとして定義されます。詳細については、「PostgreSQL メッセージの重要度」を参照してください。例えば、構文エラーやクエリのキャンセルは、エラーメッセージを生成します。 バックグラウンドのチェックポインタープロセスエラーなどの内部の PostgreSQL サーバーエラーは、エラーメッセージを生成しません。ただし、ログの重要度レベルの設定に関係なく、このようなイベントのレコードは引き続き出力されます。これにより、攻撃者がログをオフにして検出を回避することを防ぎます。 exitCode フィールドも参照してください。
exitCode	int	セッション終了レコードに使用される値。clean exit では、終了コードが含まれます。エラーシナリオによっては、終了コードが常に得られるとは限りません。例えば、PostgreSQL で exit() を実行している場合や、オペレーターが kill -9 などのコマンドを実行している場合があります。 エラーが発生した場合は、PostgreSQL エラーコードにリストされている SQL エラーコード (SQLSTATE) が exitCode フィールドに表示されます。 errorMessage フィールドも参照してください。

フィールド	データ型	説明
logTime	文字列	監査コードパスに記録されているタイムスタンプ。これは、SQL ステートメントの実行終了時刻を表します。startTime フィールドも参照してください。
netProtocol	文字列	ネットワーク通信プロトコル。
objectName	文字列	データベースオブジェクトの名前 (SQL ステートメントを使用している場合)。このフィールドは、SQL ステートメントがデータベースオブジェクトに対して機能する場合にのみ使用されます。SQL ステートメントがオブジェクトに対して機能していない場合、この値は null です。
objectType	文字列	テーブル、インデックス、ビューなどのデータベースオブジェクトタイプ。このフィールドは、SQL ステートメントがデータベースオブジェクトに対して機能する場合にのみ使用されます。SQL ステートメントがオブジェクトに対して機能していない場合、この値は null です。有効な値には次のようなものがあります。 • COMPOSITE TYPE • FOREIGN TABLE • FUNCTION • INDEX • MATERIALIZED VIEW • SEQUENCE • TABLE • TOAST TABLE • VIEW • UNKNOWN
paramList	文字列	SQL ステートメントに渡されるカンマ区切りのパラメータの配列。SQL ステートメントにパラメータがない場合、この値は空の配列です。

フィールド	データ型	説明
pid	int	クライアント接続を処理するために割り当てられているバックエンドプロセスのプロセス ID。
remoteHost	文字列	クライアントの IP アドレスまたはホスト名。Aurora PostgreSQL では、どちらが使用されるかは、データベースの log_hostname パラメータ設定によって異なります。
remotePort	文字列	クライアントのポート番号。
rowCount	int	SQL 文によって返された行数。例えば、SELECT ステートメントが 10 行を返す場合、rowCount は 10 になります。INSERT ステートメントまたは UPDATE ステートメントの場合、RowCount は 0 です。
serverHost	文字列	データベースサーバーのホスト IP アドレス。
serverType	文字列	データベースサーバーのタイプ (例: PostgreSQL)。
serverVersion	文字列	データベースサーバーのバージョン (例: Aurora PostgreSQL の 2.3.1)。
serviceName	文字列	サービスの名前 (例: Amazon Aurora PostgreSQL-Compatible edition)。
sessionId	int	一意の疑似セッション識別子。
sessionId	int	一意の疑似セッション識別子。
startTime (バージョン 1.1 のデータベースアクティビティレコードのみ)	文字列	SQL ステートメントの実行がスタートされた時刻。 SQL ステートメントのおおよその実行時間を計算するには、logTime - startTime を使用します。logTime フィールドも参照してください。
statementId	int	クライアントの SQL ステートメントの識別子。カウンターはセッションレベルであり、クライアントによって SQL ステートメントが入力される度に増加します。

フィールド	データ型	説明
substatementId	int	SQL サブステートメントの識別子。この値は、statementId フィールドで識別された各 SQL ステートメントに含まれるサブステートメントをカウントします。
type	文字列	イベントタイプ。有効な値は record または heartbeat です。

Aurora MySQL の databaseActivityEventList フィールド

フィールド	データ型	説明
class	文字列	アクティビティイベントのクラス。 Aurora MySQL の有効な値は以下のとおりです。 • MAIN - SQL ステートメントを表すプライマリイベントです。 • AUX - 追加の詳細を含む補足イベント。例えば、オブジェクトの名前を変更したステートメントには、新しい名前を反映するクラス AUX を持つイベントがあります。 同じステートメントに対応する MAIN イベントと AUX イベントを検索するには、pid フィールドと statementId フィールドの値が同じである、異なるイベントがないか確認します。
clientApplication	文字列	クライアントのレポートどおりにクライアントが接続に使用していたアプリケーション。クライアントはこの情報を指定する必要はないため、値は null でも問題ありません。
command	文字列	SQL ステートメントの一般的なカテゴリ。このフィールドの値は class の値によって異なります。 class が MAIN の場合の値には、次の値が含まれます。 • CONNECT - クライアントセッションが接続されている場合。

フィールド	データ型	説明
		• QUERY - SQL ステートメント。class の AUX 値が 1 つ以上のイベントを伴います。 • DISCONNECT - クライアントセッションが切断されている場合。 • FAILED_CONNECT - クライアントが接続を試みたが、接続できない場合。 • CHANGEUSER - 発行するステートメントではなく、MySQL ネットワークプロトコルの一部である状態の変更。 class が AUX の場合の値には、次の値が含まれます。 • READ - SELECT ステートメントまたは COPY ステートメント (ソースがリレーションまたはクエリの場合)。 • WRITE - INSERT、UPDATE、DELETE、TRUNCATE、COPY ステートメント (送信先がリレーションの場合)。 • DROP - オブジェクトの削除。 • CREATE - オブジェクトの作成。 • RENAME - オブジェクトの名前の変更。 • ALTER - オブジェクトのプロパティの変更。

フィールド	データ型	説明
commandText	文字列	class の MAIN 値を持つイベントの場合、このフィールドはユーザーが渡した実際の SQL ステートメントを表します。このフィールドは、接続レコードまたは切断レコードを除くすべてのタイプのレコードに使用されます。この場合、値は null です。 class の AUX 値を持つイベントの場合、このフィールドには、イベントに関係するオブジェクトに関する補足情報が含まれます。 Aurora MySQL では、引用符などの文字の前には、エスケープ文字を表すバックスラッシュが付きます。 **Important** 各ステートメントの SQL テキスト全体が、機密データを含む監査ログに表示されます。ただし、Aurora が次の SQL ステートメントのようにコンテキストから判断できる場合、データベースユーザーのパスワードは編集されます。 `mysql> SET PASSWORD = 'my-password ';` **Note** セキュリティ上のベストプラクティスとして、ここに示されているプロンプト以外のパスワードを指定してください。
databaseName	string	ユーザーが接続したデータベース。
dbProtocol	文字列	データベースプロトコル。現在、この値は常に Aurora MySQL の MySQL です。
dbUserName	文字列	クライアントが認証したデータベースユーザー。

フィールド	データ型	説明
endTime (バージョン 1.2 のデータベースアクティビティレコードのみ)	文字列	SQL ステートメントの実行が終了した時刻。これは、協定世界時 (UTC) 形式で表されます。 SQL ステートメントの実行時間を計算するには、endTime - startTime を使用します。startTime フィールドも参照してください。
errorMessage (バージョン 1.1 のデータベースアクティビティレコードのみ)	文字列	エラーがあった場合、このフィールドには DB サーバーによって生成されるはずのエラーメッセージが表示されます。エラーにならなかった通常のステートメントの場合、errorMessage の値は null です。 エラーは、重要度が ERROR 以上の、クライアントで表示される MySQL エラーログイベントを生成するアクティビティとして定義されます。詳細については、MySQL リファレンスマニュアルの「エラーログ」を参照してください。例えば、構文エラーやクエリのキャンセルは、エラーメッセージを生成します。 バックグラウンドのチェックポインタープロセスエラーなどの内部の MySQL サーバーエラーは、エラーメッセージを生成しません。ただし、ログの重要度レベルの設定に関係なく、このようなイベントのレコードは引き続き出力されます。これにより、攻撃者がログをオフにして検出を回避することを防ぎます。 exitCode フィールドも参照してください。
exitCode	int	セッション終了レコードに使用される値。clean exit では、終了コードが含まれます。エラーシナリオによっては、終了コードが常に得られるとは限りません。このような場合、この値はゼロになるか、空白になる可能性があります。

フィールド	データ型	説明
logTime	文字列	監査コードパスに記録されているタイムスタンプ。これは、協定世界時 (UTC) 形式で表されます。ステートメントの期間を計算する最も正確な方法については、startTime および endTime フィールドを参照してください。
netProtocol	文字列	ネットワーク通信プロトコル。現在、この値は常に Aurora MySQL の TCP です。
objectName	文字列	データベースオブジェクトの名前 (SQL ステートメントを使用している場合)。このフィールドは、SQL ステートメントがデータベースオブジェクトに対して機能する場合にのみ使用されます。SQL ステートメントがオブジェクトに対して機能していない場合、この値は空白になります。オブジェクトの完全修飾名を作成するには、databaseName と objectName を結合します。クエリに複数のオブジェクトが含まれる場合、このフィールドはカンマで区切られた名前のリストにすることができます。
objectType	文字列	テーブル、インデックスなどのデータベースオブジェクトタイプ。このフィールドは、SQL ステートメントがデータベースオブジェクトに対して機能する場合にのみ使用されます。SQL ステートメントがオブジェクトに対して機能していない場合、この値は null です。 Aurora MySQL の有効な値には次のようなものがあります。 • INDEX • TABLE • UNKNOWN
paramList	文字列	このフィールドは Aurora MySQL には使用されず、常に null です。

フィールド	データ型	説明
pid	int	クライアント接続を処理するために割り当てられているバックエンドプロセスのプロセス ID。データベースサーバー再起動すると、pid が変更され、statementId フィールドのカウンターが初期からやり直されます。
remoteHost	文字列	SQL ステートメントを発行したクライアントの IP アドレスまたはホスト名。Aurora MySQL では、どちらが使用されるかは、データベースの skip_name_resolve パラメータ設定によって異なります。この値 localhost は、rdsadmin スペシャルユーザーからのアクティビティを示します。
remotePort	文字列	クライアントのポート番号。
rowCount	int	SQL ステートメントによって影響を受けた、または取得されたテーブルの行数。このフィールドは、データ操作言語 (DML) ステートメントである SQL ステートメントでのみ使用されます。SQL ステートメントが DML ステートメントではない場合、この値は null です。
serverHost	文字列	データベースサーバーインスタンス識別子。この値は、Aurora MySQL と Aurora PostgreSQLでは異なって表されます。Aurora PostgreSQL は、識別子の代わりに IP アドレスを使用します。
serverType	文字列	データベースサーバーのタイプ (例: MySQL)。
serverVersion	文字列	データベースサーバーのバージョン。現在、この値は常に Aurora MySQL の MySQL 5.7.12 です。
serviceName	文字列	サービスの名前。現在、この値は常に Aurora MySQL の Amazon Aurora MySQL です。
sessionId	int	一意の疑似セッション識別子。

フィールド	データ型	説明
startTime (バージョン 1.1 のデータベースアクティビティレコードのみ)	文字列	SQL ステートメントの実行がスタートされた時刻。これは、協定世界時 (UTC) 形式で表されます。 SQL ステートメントの実行時間を計算するには、endTime - startTime を使用します。endTime フィールドも参照してください。
statementId	int	クライアントの SQL ステートメントの識別子。カウンターは、クライアントによって SQL ステートメントが入力される度に増加します。DB インスタンスが再起動されると、カウンターがリセットされます。
substatementId	int	SQL サブステートメントの識別子。この値は、クラス MAIN を持つイベントの場合は 1、クラス AUX を持つイベントの場合は 2 です。statementId フィールドを使用して、同じステートメントによって生成されたすべてのイベントを識別します。
transactionId (バージョン 1.2 のデータベースアクティビティレコードのみ)	int	トランザクションの識別子。
type	文字列	イベントタイプ。有効な値は record または heartbeat です。

AWS SDK を使用したデータベースアクティビティストリーミングの処理

アクティビティストリーミングをプログラムで処理するには、AWS SDK を使用します。以下は、Kinesis データストリームを処理する方法で完全に機能する Java および Python の例です。

Java

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.net.InetAddress;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.Security;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import java.util.zip.GZIPInputStream;

import javax.crypto.Cipher;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.SecretKeySpec;

import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CryptoInputStream;
import com.amazonaws.encryptionsdk.jce.JceMasterKey;
import
 com.amazonaws.services.kinesis.clientlibrary.exceptions.InvalidStateException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ShutdownException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ThrottlingException;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorCheckpointer;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorFactory;
import
 com.amazonaws.services.kinesis.clientlibrary.lib.worker.InitialPositionInStream;
import
 com.amazonaws.services.kinesis.clientlibrary.lib.worker.KinesisClientLibConfiguration;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.ShutdownReason;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker.Builder;
import com.amazonaws.services.kinesis.model.Record;
import com.amazonaws.services.kms.AWSKMS;
```

```
import com.amazonaws.services.kms.AWSKMSClientBuilder;
import com.amazonaws.services.kms.model.DecryptRequest;
import com.amazonaws.services.kms.model.DecryptResult;
import com.amazonaws.util.Base64;
import com.amazonaws.util.IOUtils;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.annotations.SerializedName;
import org.bouncycastle.jce.provider.BouncyCastleProvider;

public class DemoConsumer {

    private static final String STREAM_NAME = "aws-rds-das-[cluster-external-
resource-id]";
    private static final String APPLICATION_NAME = "AnyApplication"; //unique
 application name for dynamo table generation that holds kinesis shard tracking
    private static final String AWS_ACCESS_KEY =
 "[AWS_ACCESS_KEY_TO_ACCESS_KINESIS]";
    private static final String AWS_SECRET_KEY =
 "[AWS_SECRET_KEY_TO_ACCESS_KINESIS]";
    private static final String DBC_RESOURCE_ID = "[cluster-external-resource-id]";
    private static final String REGION_NAME = "[region-name]"; //us-east-1, us-
east-2...
    private static final BasicAWSCredentials CREDENTIALS = new
 BasicAWSCredentials(AWS_ACCESS_KEY, AWS_SECRET_KEY);
    private static final AWSStaticCredentialsProvider CREDENTIALS_PROVIDER = new
 AWSStaticCredentialsProvider(CREDENTIALS);

    private static final AwsCrypto CRYPTO = new AwsCrypto();
    private static final AWSKMS KMS = AWSKMSClientBuilder.standard()
            .withRegion(REGION_NAME)
            .withCredentials(CREDENTIALS_PROVIDER).build();

    class Activity {
        String type;
        String version;
        String databaseActivityEvents;
        String key;
    }

    class ActivityEvent {
        @SerializedName("class") String _class;
        String clientApplication;
        String command;
```

```
        String commandText;
        String databaseName;
        String dbProtocol;
        String dbUserName;
        String endTime;
        String errorMessage;
        String exitCode;
        String logTime;
        String netProtocol;
        String objectName;
        String objectType;
        List<String> paramList;
        String pid;
        String remoteHost;
        String remotePort;
        String rowCount;
        String serverHost;
        String serverType;
        String serverVersion;
        String serviceName;
        String sessionId;
        String startTime;
        String statementId;
        String substatementId;
        String transactionId;
        String type;
    }

    class ActivityRecords {
        String type;
        String clusterId;
        String instanceId;
        List<ActivityEvent> databaseActivityEventList;
    }

    static class RecordProcessorFactory implements IRecordProcessorFactory {
        @Override
        public IRecordProcessor createProcessor() {
            return new RecordProcessor();
        }
    }

    static class RecordProcessor implements IRecordProcessor {
```

```
        private static final long BACKOFF_TIME_IN_MILLIS = 3000L;
        private static final int PROCESSING_RETRIES_MAX = 10;
        private static final long CHECKPOINT_INTERVAL_MILLIS = 60000L;
        private static final Gson GSON = new
GsonBuilder().serializeNulls().create();

        private static final Cipher CIPHER;
        static {
            Security.insertProviderAt(new BouncyCastleProvider(), 1);
            try {
                CIPHER = Cipher.getInstance("AES/GCM/NoPadding", "BC");
            } catch (NoSuchAlgorithmException | NoSuchPaddingException |
NoSuchProviderException e) {
                throw new ExceptionInInitializerError(e);
            }
        }

        private long nextCheckpointTimeInMillis;

        @Override
        public void initialize(String shardId) {
        }

        @Override
        public void processRecords(final List<Record> records, final
IRecordProcessorCheckpointer checkpointer) {
            for (final Record record : records) {
                processSingleBlob(record.getData());
            }

            if (System.currentTimeMillis() > nextCheckpointTimeInMillis) {
                checkpoint(checkpointer);
                nextCheckpointTimeInMillis = System.currentTimeMillis() +
CHECKPOINT_INTERVAL_MILLIS;
            }
        }

        @Override
        public void shutdown(IRecordProcessorCheckpointer checkpointer,
ShutdownReason reason) {
            if (reason == ShutdownReason.TERMINATE) {
                checkpoint(checkpointer);
            }
        }
```

```
       private void processSingleBlob(final ByteBuffer bytes) {
           try {
               // JSON $Activity
               final Activity activity = GSON.fromJson(new String(bytes.array(),
StandardCharsets.UTF_8), Activity.class);

               // Base64.Decode
               final byte[] decoded =
Base64.decode(activity.databaseActivityEvents);
               final byte[] decodedDataKey = Base64.decode(activity.key);

               Map<String, String> context = new HashMap<>();
               context.put("aws:rds:dbc-id", DBC_RESOURCE_ID);

               // Decrypt
               final DecryptRequest decryptRequest = new DecryptRequest()

.withCiphertextBlob(ByteBuffer.wrap(decodedDataKey)).withEncryptionContext(context);
               final DecryptResult decryptResult = KMS.decrypt(decryptRequest);
               final byte[] decrypted = decrypt(decoded,
getByteArray(decryptResult.getPlaintext()));

               // GZip Decompress
               final byte[] decompressed = decompress(decrypted);
               // JSON $ActivityRecords
               final ActivityRecords activityRecords = GSON.fromJson(new
String(decompressed, StandardCharsets.UTF_8), ActivityRecords.class);

               // Iterate throught $ActivityEvents
               for (final ActivityEvent event :
activityRecords.databaseActivityEventList) {
                   System.out.println(GSON.toJson(event));
               }
           } catch (Exception e) {
               // Handle error.
               e.printStackTrace();
           }
       }

       private static byte[] decompress(final byte[] src) throws IOException {
           ByteArrayInputStream byteArrayInputStream = new
ByteArrayInputStream(src);
```

```
            GZIPInputStream gzipInputStream = new
GZIPInputStream(byteArrayInputStream);
            return IOUtils.toByteArray(gzipInputStream);
        }

        private void checkpoint(IRecordProcessorCheckpointer checkpointer) {
            for (int i = 0; i < PROCESSING_RETRIES_MAX; i++) {
                try {
                    checkpointer.checkpoint();
                    break;
                } catch (ShutdownException se) {
                    // Ignore checkpoint if the processor instance has been shutdown
(fail over).
                    System.out.println("Caught shutdown exception, skipping
checkpoint." + se);
                    break;
                } catch (ThrottlingException e) {
                    // Backoff and re-attempt checkpoint upon transient failures
                    if (i >= (PROCESSING_RETRIES_MAX - 1)) {
                        System.out.println("Checkpoint failed after " + (i + 1) +
"attempts." + e);
                        break;
                    } else {
                        System.out.println("Transient issue when checkpointing -
attempt " + (i + 1) + " of " + PROCESSING_RETRIES_MAX + e);
                    }
                } catch (InvalidStateException e) {
                    // This indicates an issue with the DynamoDB table (check for
table, provisioned IOPS).
                    System.out.println("Cannot save checkpoint to the DynamoDB table
used by the Amazon Kinesis Client Library." + e);
                    break;
                }
                try {
                    Thread.sleep(BACKOFF_TIME_IN_MILLIS);
                } catch (InterruptedException e) {
                    System.out.println("Interrupted sleep" + e);
                }
            }
        }
    }

    private static byte[] decrypt(final byte[] decoded, final byte[] decodedDataKey)
throws IOException {
```

```
        // Create a JCE master key provider using the random key and an AES-GCM
encryption algorithm
        final JceMasterKey masterKey = JceMasterKey.getInstance(new
SecretKeySpec(decodedDataKey, "AES"),
                "BC", "DataKey", "AES/GCM/NoPadding");
        try (final CryptoInputStream<JceMasterKey> decryptingStream =
CRYPTO.createDecryptingStream(masterKey, new ByteArrayInputStream(decoded));
             final ByteArrayOutputStream out = new ByteArrayOutputStream()) {
            IOUtils.copy(decryptingStream, out);
            return out.toByteArray();
        }
    }

    public static void main(String[] args) throws Exception {
        final String workerId = InetAddress.getLocalHost().getCanonicalHostName() +
":" + UUID.randomUUID();
        final KinesisClientLibConfiguration kinesisClientLibConfiguration =
                new KinesisClientLibConfiguration(APPLICATION_NAME, STREAM_NAME,
CREDENTIALS_PROVIDER, workerId);

kinesisClientLibConfiguration.withInitialPositionInStream(InitialPositionInStream.LATEST);
        kinesisClientLibConfiguration.withRegionName(REGION_NAME);
        final Worker worker = new Builder()
                .recordProcessorFactory(new RecordProcessorFactory())
                .config(kinesisClientLibConfiguration)
                .build();

        System.out.printf("Running %s to process stream %s as worker %s...\n",
APPLICATION_NAME, STREAM_NAME, workerId);

        try {
            worker.run();
        } catch (Throwable t) {
            System.err.println("Caught throwable while processing data.");
            t.printStackTrace();
            System.exit(1);
        }
        System.exit(0);
    }

    private static byte[] getByteArray(final ByteBuffer b) {
        byte[] byteArray = new byte[b.remaining()];
        b.get(byteArray);
        return byteArray;
```

```
    }
}
```

Python

```
import base64
import json
import zlib
import aws_encryption_sdk
from aws_encryption_sdk import CommitmentPolicy
from aws_encryption_sdk.internal.crypto import WrappingKey
from aws_encryption_sdk.key_providers.raw import RawMasterKeyProvider
from aws_encryption_sdk.identifiers import WrappingAlgorithm, EncryptionKeyType
import boto3

REGION_NAME = '<region>'                    # us-east-1
RESOURCE_ID = '<external-resource-id>'      # cluster-ABCD123456
STREAM_NAME = 'aws-rds-das-' + RESOURCE_ID  # aws-rds-das-cluster-ABCD123456

enc_client =
 aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.FORBID_ENCRYPT_AL

class MyRawMasterKeyProvider(RawMasterKeyProvider):
    provider_id = "BC"

    def __new__(cls, *args, **kwargs):
        obj = super(RawMasterKeyProvider, cls).__new__(cls)
        return obj

    def __init__(self, plain_key):
        RawMasterKeyProvider.__init__(self)
        self.wrapping_key =
 WrappingKey(wrapping_algorithm=WrappingAlgorithm.AES_256_GCM_IV12_TAG16_NO_PADDING,
                                          wrapping_key=plain_key,
 wrapping_key_type=EncryptionKeyType.SYMMETRIC)

    def _get_raw_key(self, key_id):
        return self.wrapping_key


def decrypt_payload(payload, data_key):
    my_key_provider = MyRawMasterKeyProvider(data_key)
    my_key_provider.add_master_key("DataKey")
```

```
    decrypted_plaintext, header = enc_client.decrypt(
        source=payload,

 materials_manager=aws_encryption_sdk.materials_managers.default.DefaultCryptoMaterialsManag
    return decrypted_plaintext


def decrypt_decompress(payload, key):
    decrypted = decrypt_payload(payload, key)
    return zlib.decompress(decrypted, zlib.MAX_WBITS + 16)


def main():
    session = boto3.session.Session()
    kms = session.client('kms', region_name=REGION_NAME)
    kinesis = session.client('kinesis', region_name=REGION_NAME)

    response = kinesis.describe_stream(StreamName=STREAM_NAME)
    shard_iters = []
    for shard in response['StreamDescription']['Shards']:
        shard_iter_response = kinesis.get_shard_iterator(StreamName=STREAM_NAME,
 ShardId=shard['ShardId'],

 ShardIteratorType='LATEST')
        shard_iters.append(shard_iter_response['ShardIterator'])

    while len(shard_iters) > 0:
        next_shard_iters = []
        for shard_iter in shard_iters:
            response = kinesis.get_records(ShardIterator=shard_iter, Limit=10000)
            for record in response['Records']:
                record_data = record['Data']
                record_data = json.loads(record_data)
                payload_decoded =
 base64.b64decode(record_data['databaseActivityEvents'])
                data_key_decoded = base64.b64decode(record_data['key'])
                data_key_decrypt_result =
 kms.decrypt(CiphertextBlob=data_key_decoded,

 EncryptionContext={'aws:rds:dbc-id': RESOURCE_ID})
                print (decrypt_decompress(payload_decoded,
 data_key_decrypt_result['Plaintext']))
            if 'NextShardIterator' in response:
                next_shard_iters.append(response['NextShardIterator'])
```

```
        shard_iters = next_shard_iters


if __name__ == '__main__':
    main()
```

データベースアクティビティストリーミングへのアクセスの管理

データベースアクティビティストリーミングに対する適切な AWS Identity and Access Management (IAM) ロールの権限が付与されているユーザーは、DB クラスターのアクティビティストリーミング設定の作成、スタート、停止、および変更ができます。これらのアクションはストリーミングの監査ログに含まれています。コンプライアンスのベストプラクティスとして、これらの権限は DBA に付与しないことをお勧めします。

データベースアクティビティストリーミングへのアクセスを設定するには、IAM ポリシーを使用します。Aurora 認証の詳細については、「Amazon Aurora での Identity and Access Management」を参照してください。IAM ポリシーの作成の詳細については、「IAM データベースアクセス用の IAM ポリシーの作成と使用」を参照してください。

Example データベースアクティビティストリーミングの設定を許可するポリシー

アクティビティストリーミングを変更するためのきめ細かいアクセスをユーザーに付与するには、IAM ポリシーでサービス固有のオペレーションコンテキストキー (`rds:StartActivityStream` および `rds:StopActivityStream`) を使用します。次の IAM ポリシーの例では、ユーザーまたはロールがアクティビティストリーミングを設定することを許可します。

```
{
   "Version":"2012-10-17",
   "Statement":[
        {
            "Sid":"ConfigureActivityStreams",
            "Effect":"Allow",
            "Action": [
                "rds:StartActivityStream",
                "rds:StopActivityStream"
            ],
            "Resource":"*",
          }
     ]
```

```
}
```

Example データベースアクティビティストリーミングのスタートを許可するポリシー

次の IAM ポリシーの例では、ユーザーまたはロールがアクティビティストリーミングをスタートすることを許可します。

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"AllowStartActivityStreams",
            "Effect":"Allow",
            "Action":"rds:StartActivityStream",
            "Resource":"*"
        }
    ]
}
```

Example データベースアクティビティストリーミングの停止を許可するポリシー

次の IAM ポリシーの例では、ユーザーまたはロールがアクティビティストリーミングを停止することを許可します。

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"AllowStopActivityStreams",
            "Effect":"Allow",
            "Action":"rds:StopActivityStream",
            "Resource":"*"
        }
     ]
}
```

Example データベースアクティビティストリーミングのスタートを拒否するポリシー

次の IAM ポリシーの例では、ユーザーまたはロールによるアクティビティストリーミングのスタートが阻止されます。

```
{
```

```
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"DenyStartActivityStreams",
            "Effect":"Deny",
            "Action":"rds:StartActivityStream",
            "Resource":"*"
        }
     ]
}
```

Example データベースアクティビティストリーミング停止を拒否するポリシー

次の IAM ポリシーの例では、ユーザーまたはロールによるアクティビティストリーミングの停止が阻止されます。

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"DenyStopActivityStreams",
            "Effect":"Deny",
            "Action":"rds:StopActivityStream",
            "Resource":"*"
        }
    ]
}
```

Amazon GuardDuty RDS Protection による脅威のモニタリング

Amazon GuardDuty は、AWS 環境内のアカウント、コンテナ、ワークロード、データを保護する脅威検知サービスです。GuardDuty は、機械学習（ML）モデル、異常および脅威検出機能を使用して、さまざまなログソースとランタイムアクティビティを継続的に監視し、環境内の潜在的なセキュリティリスクと悪意のあるアクティビティを特定して優先順位を付けます。

Amazon GuardDuty RDS Protection は、Amazon Aurora データベースに対する潜在的なアクセス脅威がないか、ログインイベントを分析してプロファイリングします。RDS Protection を有効にすると、GuardDuty は Aurora データベースの RDS ログインイベントを消費します。RDS Protection は、これらのイベントを監視し、潜在的な内部脅威や外部アクターがないかプロファイリングします。

GuardDuty RDS Protection の有効化の詳細については、「Amazon GuardDuty ユーザーガイド」の「GuardDuty RDS Protection」を参照してください。

RDS Protection が潜在的な脅威 (一連の成功、失敗、未完了のログイン試行で異常なパターンが発生するなど) を検出すると、GuardDuty は侵害された可能性のあるデータベースに関する詳細を含む新しい検出結果を生成します。Amazon GuardDuty コンソールでは、検出結果の概要セクションで検出結果の詳細を見ることができます。検出結果の詳細は検出結果のタイプによって異なります。リソースタイプとリソースロールという基本的な情報によって、検出結果にどのような情報が表示されるか決まります。検索結果の一般的に入手可能な詳細と検出結果タイプの詳細については、「Amazon GuardDuty ユーザーガイド」の「検出結果の詳細」と「GuardDuty RDS Protection の検索結果タイプ」を参照してください。

RDS Protection 機能は、この機能が使用可能な任意の AWS リージョン の任意の AWS アカウントについて、有効または無効にできます。RDS Protection が有効になっていない場合、GuardDuty は侵害の恐れがある Aurora データベースを検出したり、侵害の詳細を提供したりしません。

既存の GuardDuty アカウントでは、RDS Protection を 30 日間お試しいただけます。新しい GuardDuty アカウントでは、RDS Protection が既に有効になっており、30 日間の無料トライアル期間に含まれています。詳細については、「Amazon GuardDuty ユーザーガイド」の「GuardDuty コストの見積もり」を参照してください。

GuardDuty がまだ RDS Protection をサポートしていない場合の AWS リージョン の詳細については、「Amazon GuardDuty ユーザーガイド」の「リージョン固有機能の可用性」を参照してください。

次の表は、GuardDuty RDS Protection がサポートする Aurora データベースのバージョンを示しています。

Amazon Aurora DB エンジン	サポート対象エンジンバージョン
Aurora MySQL	• 2.10.2 以降 • 3.02.1 以降
Aurora PostgreSQL	• 10.17 以降 • 11.12 以降 • 12.7 以降 • 13.3 以降 • 14.3 以降 • 15.2 以降 • 16.1 以降

Amazon Aurora MySQL の操作

Amazon Aurora MySQL は、フルマネージド型で MySQL　互換のリレーショナルデータベースエンジンです。ハイエンドの商用データベースにあるスピードと信頼性を、オープンソースデータベースのシンプルさとコスト効率によりご提供します。MySQL を Aurora MySQL に差し替えることで、新規および既存の MySQL のデプロイを簡単に、コスト効率よく設定、操作、スケーリングできるようになり、ユーザーは本来のビジネスやアプリケーションに専念できます。Amazon RDS ではプロビジョニング、パッチ適用、バックアップ、復旧、障害検出、あるいは修復など、データベース関連の日常的なタスクを処理することで Aurora を管理します。また、Amazon RDS には、既存の Amazon RDS for MySQL アプリケーションから Aurora MySQL への変換をボタン操作のみで行える、移行ツールも用意されています。

トピック

Amazon Aurora MySQL の概要

ここで示している各セクションで、Amazon Aurora MySQL の概要を学習できます。

トピック

- Amazon Aurora MySQL パフォーマンスの拡張
- Amazon Aurora MySQL と空間データ
- Aurora MySQL バージョン 3 は MYSQL 8.0 との互換性があります。
- MySQL 5.7 互換 Aurora MySQL バージョン 2

Amazon Aurora MySQL パフォーマンスの拡張

Amazon Aurora には、ハイエンドな商用データベースのさまざまなニーズをサポートするパフォーマンスエクステンションがあります。

高速挿入

高速挿入は、プライマリキーによってソートされたパラレル挿入を加速し、特に `LOAD DATA` および `INSERT INTO ... SELECT ...` ステートメントに適用されます。高速挿入は、ステートメントの実行中にインデックストラバーサルのカーソル位置をキャッシュします。これによって、再度インデックスをトラバースする必要がなくなります。

高速挿入は、Aurora MySQL バージョン 3.03.2 以降の通常の InnoDB テーブルでのみ有効です。この最適化は、InnoDB 一時テーブルでは機能しません。Aurora MySQL バージョン 2 では、すべての 2.11 および 2.12 バージョンで無効になっています。高速挿入の最適化は、アダプティブハッシュインデックスの最適化が無効になっている場合にのみ機能します。

次のメトリクスをモニタリングして、DB クラスターに対する高速挿入の効果を判断できます。

- `aurora_fast_insert_cache_hits`: キャッシュされたカーソルが正常に取得され検証された時に増加するカウンター。
- `aurora_fast_insert_cache_misses`: キャッシュされたカーソルがすでに有効ではなく、Aurora が通常のインデックストラバーサルを実行したときに増加するカウンター。

次のコマンドを使用して、高速挿入メトリクスの現在の値を取得できます。

```
mysql> show global status like 'Aurora_fast_insert%';
```

以下のような出力結果が取得できます。

```
+---------------------------------+-----------+
| Variable_name                   | Value     |
+---------------------------------+-----------+
| Aurora_fast_insert_cache_hits   | 3598300   |
| Aurora_fast_insert_cache_misses | 436401336 |
+---------------------------------+-----------+
```

Amazon Aurora MySQL と空間データ

以下のリストでは、主要な Aurora MySQL 空間機能の概要を説明し、それらが MySQL の空間機能にどのように対応しているかを解説しています。

- Aurora MySQL バージョン 2 は、MySQL 5.7 と同じ空間データ型および空間関係関数をサポートします。これらのデータ型および関数の詳細については、MySQL 5.7 のドキュメントの空間データ型および空間関係関数を参照してください。
- Aurora MySQL バージョン 3 は、MySQL 8.0 と同じ空間データ型および空間関係関数をサポートします。これらのデータ型および関数の詳細については、MySQL 8.0 のドキュメントの空間データ型および空間関係関数を参照してください。
- Aurora MySQL は、InnoDB テーブルの空間インデックス作成をサポートします。空間インデックス作成では、空間的データのクエリにおける大規模なデータセットのクエリパフォーマンスが向上します。MySQL では、InnoDB テーブルの空間インデックス作成は MySQL 5.7 と 8.0 で使用できます。

  Aurora MySQL は、空間クエリで高度なパフォーマンスを得るために MySQL とは異なる空間インデックスの戦略を使用します。Aurora 空間インデックスの実装は、B-tree のスペースのフィルカーブを使用します。これは、R-tree よりも高度なパフォーマンスを空間範囲スキャンに提供することを目的としています。

  > **Note**
  >
  > Aurora MySQL では、空間リファレンス識別子 (SRID) が含まれているカラムで空間インデックスが定義されているテーブルのトランザクションを、別のトランザクションで更新するために選択されている領域に挿入することはできません。

次のデータ定義言語 (DDL) ステートメントは空間的なデータ型を使用する列にインデックスを作成するためにサポートされています。

CREATE TABLE

SPATIAL INDEX ステートメントの CREATE TABLE キーワードを使用して、空間インデックスを新しいテーブルの列に追加することができます。次に例を示します。

```
CREATE TABLE test (shape POLYGON NOT NULL, SPATIAL INDEX(shape));
```

ALTER TABLE

SPATIAL INDEX ステートメントの ALTER TABLE キーワードを使用して、空間インデックスを既存のテーブルの列に追加することができます。次に例を示します。

```
ALTER TABLE test ADD SPATIAL INDEX(shape);
```

CREATE INDEX

SPATIAL ステートメントの CREATE INDEX キーワードを使用して、空間インデックスを既存のテーブルの列に追加することができます。次に例を示します。

```
CREATE SPATIAL INDEX shape_index ON test (shape);
```

Aurora MySQL バージョン 3 は MYSQL 8.0 との互換性があります。

Aurora MySQL バージョン 3 を使用して、MySQL 互換の最新機能、パフォーマンスの強化、およびバグ修正を入手できます。以下では、MySQL 8.0 の互換性を持つ Aurora MySQL バージョン 3 について学ぶことができます。クラスターとアプリケーションを Aurora MySQL バージョン 3 にアップグレードする方法を学ぶことができます。

Aurora Serverless v2 など、Aurora の一部の機能は、Aurora MySQL バージョン 3 を必要とします。

トピック

- MySQL 8.0 コミュニティエディションからの機能
- Aurora MySQL サーバーレス v2 の前提条件である Aurora MySQL バージョン 3
- Aurora MySQL バージョン 3 のリリースノート
- 新しいパラレルクエリの最適化
- データベースの再起動時間を短縮するための最適化

- Aurora MySQL バージョン 3 での新しい一時テーブルの動作
- Aurora MySQL バージョン 2 と Aurora MySQL バージョン 3 の比較
- Aurora MySQL バージョン 3 と MySQL 8.0 コミュニティエディションの比較
- Aurora MySQL バージョン 3 へのアップグレード

MySQL 8.0 コミュニティエディションからの機能

Aurora MySQL バージョン 3 の初期リリースは、MySQL 8.0.23 コミュニティエディションと互換性があります。MySQL 8.0 では、以下を含むいくつかの新機能が導入されています。

- JSON 関数。使用に関する情報については、MySQL リファレンスマニュアル のJSON 関数を参照してください。
- ウィンドウ関数。使用に関する情報については、MySQL リファレンスマニュアル のWindow 関数を参照してください。
- `WITH` 句を使用した共通テーブル表現 (CTE)。使用に関する情報については、MySQL リファレンスマニュアルの WITH (共通テーブル表現)を参照してください。
- `ALTER TABLE` ステートメントの、最適化された `ADD COLUMN` と `RENAME COLUMN` 句 。これらの最適化は「インスタント DDL」と呼ばれます。Aurora MySQL バージョン 3 はコミュニティ MySQL インスタント DDL 特徴と互換性があります。旧 Aurora 高速 DDL特徴は使用されていません。インスタント DDL の使用情報については、インスタント DDL (Aurora MySQL バージョン 3) を参照してください。
- 降順、機能、不可視インデックス。使用に関する情報については、MySQL リファレンスマニュアルの非表示インデックス、降順インデックス、およびCREATE INDEX インデックスを参照してください。
- SQL 文で制御されるロールベースの権限。権限モデルの変更については、ロールベースの特権モデル を参照してください。
- `SELECT ... FOR SHARE` 文のNOWAIT と `SKIP LOCKED` 句。これらの句は、他のトランザクションが行ロックを解放するのを待つことを避けます。使用の詳細については、MySQL リファレンスマニュアルの読み取りロックを参照してください。
- バイナリログ (binlog) のレプリケーションの改善。Aurora MySQL の詳細については、バイナリログレプリケーション を参照してください。特に、フィルタリングされたレプリケーションを実行できます。使用方法については、MySQL リファレンスマニュアルのサーバがレプリケーションフィルタ規則を評価する方法を参照してください。
- ヒント。MySQL 8.0 互換ヒントのいくつかは、既に Aurora MySQL バージョン 2 にバックポートされています。Aurora MySQL でのヒントの使用については、Aurora MySQL のヒント を参照

してください。コミュニティ MySQL 8.0 でのヒントの詳細なリストは、MySQL リファレンスマニュアルのオプティマイザーヒントを参照してください。

MySQL 8.0 コミュニティエディションに追加された機能の完全なリストについては、ブログ記事 MySQL 8.0 の新機能の完全なリスト を参照してください。

Aurora MySQL バージョン 3 には、コミュニティ MySQL 8.0.26 からバックポートされた、包括的言語キーワードの変更も含まれています。これらの変更の詳細については、Aurora MySQL バージョン 3 に対する包括的な言語変更 を参照してください。

Aurora MySQL サーバーレス v2 の前提条件である Aurora MySQL バージョン 3

Aurora MySQL バージョン 3 は、Aurora MySQL サーバーレス v2 クラスター内のすべての DB インスタンスの前提条件です。Aurora MySQL サーバーレス v2 には、DB クラスター内のリーダーインスタンスのサポートと、Aurora MySQL サーバーレス v1 では利用できない Aurora 機能のサポートが含まれています。また、Aurora MySQL サーバーレス v1 よりも高速かつきめ細かなスケーリングも備えています。

Aurora MySQL バージョン 3 のリリースノート

すべての Aurora MySQL バージョン 3 リリースのリリースノートについては、Aurora MySQL のリリースノートの「Amazon Aurora MySQL バージョン 3 のデータベースエンジンの更新」を参照してください。

新しいパラレルクエリの最適化

Aurora パラレルクエリの最適化は、より多くの SQL 操作に適用されるようになりました。

- パラレルクエリは、TEXT、BLOB、JSON、GEOMETRY、VARCHAR、そして 768 バイトより長い CHAR のデータ型を含んだテーブルに適用されるようになりました。
- パラレルクエリは、パーティショニングテーブルを含むクエリを最適化できます。
- パラレルクエリは、選択リストと HAVING 句内でする集計関数の呼び出を伴うクエリを最適化できます。

強化の詳細については、Aurora MySQL バージョン 3 への パラレルクエリクラスターのアップグレード を参照してください。Aurora パラレルクエリの一般情報については、Amazon Aurora MySQL のパラレルクエリの使用 を参照してください。

データベースの再起動時間を短縮するための最適化

Aurora MySQL DB クラスターは、計画的な停止時と計画外の停止時の両方で高い可用性を備えている必要があります。

データベース管理者は時折データベースのメンテナンスを行う必要があります。このメンテナンスには、データベースのパッチ適用、アップグレード、手動での再起動が必要なデータベースパラメータの変更、インスタンスクラスの変更にかかる時間を短縮するためのフェイルオーバーの実行などが含まれます。これらの計画的なアクションには、ダウンタイムが伴います。

ただし、基盤となるハードウェア障害やデータベースリソースのスロットリングによる予期しないフェイルオーバーなど、計画外のアクションによってもダウンタイムが発生することもあります。これらの計画的なアクションでも、計画外のアクションでも、必ずデータベースの再起動が必要です。

Aurora MySQL バージョン 3.05 以降では、データベースの再起動時間を短縮するための最適化が導入されました。これらの最適化により、最適化を行わない場合と比べてダウンタイムが最大 65% 短縮され、再起動後のデータベースワークロードの中断も少なくなります。

データベースのスタートアップ時には、多くの内部メモリコンポーネントが初期化されます。これらの中で最大のものは InnoDB バッファープールで、Aurora MySQL ではデフォルトでインスタンスのメモリサイズの 75% です。テストの結果、初期化時間は InnoDB バッファプールのサイズに比例し、DB インスタンスクラスのサイズに合わせて調整されることがわかりました。この初期化フェーズでは、データベースは接続を受け付けられないため、再起動時のダウンタイムが長くなります。Aurora MySQL 高速再起動の第 1 フェーズでは、バッファープールの初期化が最適化されます。これにより、データベースの初期化時間が短縮され、全体的な再起動時間が短縮されます。

詳細については、ブログ「Amazon Aurora MySQL データベースの再起動時間の最適化によってダウンタイムを削減する」を参照してください。

Aurora MySQL バージョン 3 での新しい一時テーブルの動作

Aurora MySQL バージョン 3 では、一時テーブルの処理方法は、以前の Aurora MySQL バージョンとは異なります。この新しい動作は MySQL 8.0 コミュニティエディションから継承されています。Aurora MySQL バージョン 3 で作成できる一時テーブルには、次の 2 つのタイプがあります。

- 内部 (または黙示的) 一時テーブル — 集計の並べ替え、派生テーブル、共通テーブル式 (CTE) などの操作を処理するために Aurora MySQL エンジンによって作成されます。
- ユーザー作成 (または明示的) 一時テーブル — Aurora MySQL エンジンが`CREATE TEMPORARY TABLE`表示されます。

Aurora Reader DB インスタンスの内部およびユーザー作成一時テーブルの両方について、その他の考慮事項があります。これらについては、以降のセクションで説明します。

トピック

内部 (黙示的) 一時テーブルのストレージエンジン

中間結果セットを生成するとき、Aurora MySQL は最初にメモリ内一時テーブルへの書き込みを試みます。データ型に互換性がないか、制限が設定されていることが原因で、これがうまくいかない可能性があります。その場合、一時テーブルはメモリに保持されるのではなく、ディスク上の一時テーブルに変換されます。これについての詳細は、MySQL ドキュメントの「MySQL での内部一時テーブルの使用」を参照してください。

Aurora MySQL バージョン 3 では、内部一時テーブルの動作方法は、以前の Aurora MySQL バージョンとは異なります。このような一時テーブルの InnoDB ストレージエンジンと MyISAM ストレージエンジンのいずれかを選択する代わりに、現在は `TempTable` と InnoDB ストレージエンジンのいずれかを選択します。

`TempTable` ストレージエンジンを使用すると、特定のデータの処理方法について追加の選択を行うことができます。影響を受けるデータは、DB インスタンスのすべての内部一時テーブルを保持するメモリプールをオーバーフローします。

これらの選択は、大量の一時データを生成するクエリのパフォーマンスに影響します。例えば、ラージ テーブルの `GROUP BY` のような集約を実行している場合などです。

> **Tip**
>
> ワークロードに内部一時テーブルを生成するクエリが含まれている場合は、ベンチマークを実行し、パフォーマンス関連のメトリックをモニタリングして、この変更によるアプリケーションの動作を確認します。
> 場合によっては、一時データ量は `TempTable` メモリプールに収まるか、または少量だけメモリプールから溢れます。このような場合は、内部一時テーブルおよびメモリマップファイ

ルの TempTable 設定を使用して、オーバーフローデータを保持することをお勧めします。この設定はデフォルトです。

TempTable ストレージエンジンがデフォルトです。TempTable は、テーブルあたりの最大メモリ制限ではなく、このエンジンを使うすべての一時テーブルの共通メモリプールを使用します。このメモリプールのサイズは、temptable_max_ram パラメータで特定されます。16 GB 以上のメモリを持つ DB インスタンスでは 1 GB、メモリが 16 GB 未満の DB インスタンスでは 16 MB がデフォルトになります。メモリプールのサイズは、セッションレベルのメモリ消費に影響します。

TempTable ストレージエンジンを使用するときに、一時データがメモリプールのサイズを超えることがあります。その場合、Aurora MySQL は二次的なメカニズムを使用してオーバーフローデータを保存します。

temptable_max_mmap パラメータを設定して、メモリマップテー時ファイルまたはディスク上の InnoDB 内部一時テーブルのどちらにデータがオーバーフローするか、指定することができます。これらのオーバーフローメカニズムの異なるデータ形式とオーバーフロー基準は、クエリのパフォーマンスに影響を与える可能性があります。例えば、ディスクに書き込まれるデータ量や、ディスクストレージのスループットに対する要求に影響します。

Aurora MySQL は、データオーバーフローの送信先の選択、およびクエリがライターまたはリーダー DB インスタンスのどちらで実行されるかによって、オーバーフローデータを異なる方法で格納します。

- ライターインスタンスでは、InnoDB 内部一時テーブルにオーバーフローするデータは Aurora クラスター ボリュームに格納されます。
- ライターインスタンスでは、メモリマップされた一時ファイルにオーバーフローするデータは、Aurora MySQL バージョン 3 インスタンスのローカルストレージに存在します。
- リーダーインスタンスでは、オーバーフローデータは常にローカルストレージ上のメモリマップー時ファイルに存在します。これは、読み取り専用インスタンスでは Aurora クラスターボリュームにデータを保存できないためです。

内部一時テーブルに関連する設定パラメータは、クラスター内のライターインスタンスとリーダーインスタンスに対して異なる方法で適用されます。

- リーダーインスタンスの場合、Aurora MySQL は常に TempTable ストレージエンジンを使用します。

- temptable_max_mmap のデフォルトサイズは、DB インスタンスのメモリサイズに関係なく、ライターインスタンスとリーダーインスタンスの両方で 1 GB です。この値はライターインスタンスとリーダーインスタンスの両方で調整できます。
- temptable_max_mmap を 0 に設定すると、ライターインスタンスでのメモリマップされた一時ファイルの使用がオフになります。
- リーダーインスタンスでは、temptable_max_mmap を 0 に設定することはできません。

> **Note**
>
> temptable_use_mmap パラメーターの使用はお勧めしません。これは非推奨であり、将来の MySQL リリースでサポートが削除される予定です。

内部メモリ内一時テーブルのサイズを制限する

内部 (黙示的) 一時テーブルのストレージエンジン で説明したように、temptable_max_ram および temptable_max_mmap 設定を使用して、一時テーブルリソースをグローバルに制御できます。

また、tmp_table_size DB パラメータを使用して、個々の内部メモリ内一時テーブルのサイズを制限することもできます。この制限は、個々のクエリがグローバル一時テーブルリソースを大量に消費し、これらのリソースを必要とする同時クエリのパフォーマンスに影響を与えるのを防ぐことを目的としています。

tmp_table_size パラメータは、Aurora MySQL バージョン 3 の MEMORY ストレージエンジンによって作成される一時テーブルの最大サイズを定義します。

Aurora MySQL バージョン 3.04 以降では、tmp_table_size は、aurora_tmptable_enable_per_table_limit DB パラメータが ON に設定されているときに TempTable ストレージエンジンによって作成される一時テーブルの最大サイズも定義します。この動作はデフォルトでは無効になっています (OFF)、これは Aurora MySQL バージョン 3.03 以前のバージョンと同じ動作です。

- aurora_tmptable_enable_per_table_limit が OFF のとき、tmp_table_size は、TempTable ストレージエンジンによって作成される内部メモリ内一時テーブルでは考慮されません。

  ただし、その場合でも、グローバル TempTable リソース制限は適用されます。Aurora MySQL は、グローバル TempTable リソース制限に達すると、次のように動作します。

  - ライター DB インスタンス — Aurora MySQL は、メモリ内一時テーブルを InnoDB オンディスク一時テーブルに自動的に変換します。
  - リーダー DB インスタンス — クエリはエラーで終了します。

    ```
    ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlxx_xxx' is full
    ```

- `aurora_tmptable_enable_per_table_limit` が `ON` のとき、Aurora MySQL は、`tmp_table_size` 制限に達すると、次のように動作します。
  - ライター DB インスタンス — Aurora MySQL は、メモリ内一時テーブルを InnoDB オンディスク一時テーブルに自動的に変換します。
  - リーダー DB インスタンス — クエリはエラーで終了します。

    ```
    ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlxx_xxx' is full
    ```

    この場合、グローバル `TempTable` リソース制限とテーブルごとの制限の両方が適用されます。

> **Note**
>
> `aurora_tmptable_enable_per_table_limit` パラメータは、`internal_tmp_mem_storage_engine` が `MEMORY` に設定されたときには、効果がありません。この場合、メモリ内一時テーブルの最大サイズは、`tmp_table_size` または `max_heap_table_size` のいずれか小さい方の値によって定義されます。

以下の例は、ライター DB インスタンスとリーダー DB インスタンスについて、`aurora_tmptable_enable_per_table_limit` パラメータの動作を示しています。

Example **`aurora_tmptable_enable_per_table_limit` が `OFF` に設定されたライター DB インスタンス**

メモリ内一時テーブルは InnoDB オンディスク一時テーブルに変換されません。

```
mysql> set aurora_tmptable_enable_per_table_limit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@temptable_max_r
```

```
+--------------------+------------------+-----------------------------------------
+---------------------+----------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@temptable_max_ram | @@temptable_max_mmap |
+--------------------+------------------+-----------------------------------------
+---------------------+----------------------+
|                  0 | 3.04.0           |                                        0 |
       1073741824 |           1073741824 |
+--------------------+------------------+-----------------------------------------
+---------------------+----------------------+
1 row in set (0.00 sec)

mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 0     |
+-------------------------+-------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 60000000) SELECT max(n) FROM cte;
+----------+
| max(n)   |
+----------+
| 60000000 |
+----------+
1 row in set (13.99 sec)

mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 0     |
+-------------------------+-------+
1 row in set (0.00 sec)
```

Example **aurora_tmptable_enable_per_table_limit** が **ON** に設定されたライター DB インスタンス

メモリ内一時テーブルは InnoDB オンディスク一時テーブルに変換されます。

```
mysql> set aurora_tmptable_enable_per_table_limit=1;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@tmp_table_size;
+--------------------+-----------------+-----------------------------------------
+------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@tmp_table_size |
+--------------------+-----------------+-----------------------------------------
+------------------+
|                  0 | 3.04.0          |                                        1 |
      16777216 |
+--------------------+-----------------+-----------------------------------------
+------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 0     |
+-------------------------+-------+
1 row in set (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 6000000) SELECT max(n) FROM cte;
+---------+
| max(n)  |
+---------+
| 6000000 |
+---------+
1 row in set (4.10 sec)

mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
```

```
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 1     |
+-------------------------+-------+
1 row in set (0.00 sec)
```

Example **aurora_tmptable_enable_per_table_limit** が **OFF** に設定されたリーダー DB インスタンス

tmp_table_size は適用されず、グローバル TempTable リソースの上限に達していないため、クエリはエラーなしで終了します。

```
mysql> set aurora_tmptable_enable_per_table_limit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@temptable_max_r
+--------------------+-----------------+-----------------------------------------
+----------------------+----------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@temptable_max_ram | @@temptable_max_mmap |
+--------------------+-----------------+-----------------------------------------
+----------------------+----------------------+
|                  1 | 3.04.0           |                                        0 |
       1073741824 |           1073741824 |
+--------------------+-----------------+-----------------------------------------
+----------------------+----------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 60000000) SELECT max(n) FROM cte;
+----------+
| max(n)   |
+----------+
| 60000000 |
+----------+
1 row in set (14.05 sec)
```

Example **aurora_tmptable_enable_per_table_limit** が **OFF** に設定されたリーダー DB インスタンス

aurora_tmptable_enable_per_table_limit が OFF に設定されている場合、このクエリはグローバル TempTable リソース制限に達します。クエリはリーダーインスタンスのエラーで終了します。

```
mysql> set aurora_tmptable_enable_per_table_limit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@temptable_max_r
+--------------------+------------------+-----------------------------------------
+---------------------+----------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@temptable_max_ram | @@temptable_max_mmap |
+--------------------+------------------+-----------------------------------------
+---------------------+----------------------+
|                  1 | 3.04.0           |                                        0 |
       1073741824 |           1073741824 |
+--------------------+------------------+-----------------------------------------
+---------------------+----------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.01 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 120000000) SELECT max(n) FROM cte;
ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlfd_1586_2' is full
```

Example **aurora_tmptable_enable_per_table_limit** が **ON** に設定されたリーダー DB インスタンス

tmp_table_size 制限に達すると、クエリはエラーで終了します。

```
mysql> set aurora_tmptable_enable_per_table_limit=1;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@tmp_table_size;
+--------------------+------------------+-----------------------------------------
+------------------+
```

```
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@tmp_table_size |
+--------------------+------------------+------------------------------------------
+------------------+
|                  1 | 3.04.0           |                                        1 |
      16777216 |
+--------------------+------------------+------------------------------------------
+------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 6000000) SELECT max(n) FROM cte;
ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlfd_8_2' is full
```

Aurora レプリカの内部一時テーブルのフルネス問題の緩和

一時テーブルのサイズ制限の問題を回避するには、temptable_max_ram と temptable_max_mmapパラメータを、ワークロードの要件に適合する値を組み合わせて指定します。

temptable_max_ram パラメータの値を設定するときは注意してください。この値の設定が高すぎると、データベースインスタンスの使用可能なメモリが減少し、メモリ不足状態が発生する可能性があります。DB インスタンスの平均空きメモリ量を監視します。インスタンスには十分な量の空きメモリが残るように temptable_max_ram の適切な値を決定します。詳細については、「Amazon Aurora の解放可能なメモリの問題」を参照してください。

また、ローカルストレージのサイズと一時テーブル領域の消費量をモニタリングすることも重要です。インスタンスのローカルストレージのモニタリングの詳細については、AWSナレッジセンターの記事「Aurora MySQL 互換のローカルストレージに保存されているものと、ローカルストレージの問題のトラブルシューティング方法を教えてください」を参照してください。

> **Note**
>
> この手順は、aurora_tmptable_enable_per_table_limit パラメータが ON に設定されているときには機能しません。詳細については、「内部メモリ内一時テーブルのサイズを制限する」を参照してください。

Example 1

一時テーブルの累積サイズが 20 GiB になることがわかっています。インメモリー一時テーブルを 2 GiB に設定し、ディスク上で最大 20 GiB に拡張します。

`temptable_max_ram` を **`2,147,483,648`** に、`temptable_max_mmap` を **`21,474,836,480`** に設定します。これらの値はバイト単位です。

これらのパラメータ設定により、一時テーブルが累積合計 22 GiB に拡張できます。

Example 2

現在のインスタンスサイズは 16xlarge 以上です。必要な一時テーブルの合計サイズが不明です。最大 4 GiB のメモリと、ディスク上の使用可能な最大ストレージサイズまで使用できるようにしたいと思っています。

`temptable_max_ram` を **`4,294,967,296`** に、`temptable_max_mmap` を **`1,099,511,627,776`** に設定します。これらの値はバイト単位です。

`temptable_max_mmap` を 1 TiB に設定します。これは、16 倍の大きな Aurora DB インスタンスの最大ローカルストレージである 1.2 TiB 未満です。

小さいインスタンスサイズで、使用可能なローカルストレージがいっぱいにならないように `temptable_max_mmap` の値を調整します。例えば、2xlarge インスタンスで使用できるローカルストレージは 160 GiB のみです。したがって、値を 160 GiB 未満に設定することをお勧めします。DB インスタンスサイズで使用可能なローカルストレージの詳細については、「Aurora MySQL 用の一時ストレージの制限」を参照してください。

リーダー DB インスタンスでユーザーが作成した (明示的な) 一時テーブル

`CREATE TABLE` ステートメントの `TEMPORARY` キーワードを使用して、明示的な一時テーブルを作成できます。Aurora DB クラスター内のライター DB インスタンスでは、明示的な一時テーブルがサポートされています。リーダー DB インスタンスで明示的な一時テーブルを使用することもできますが、テーブルでは InnoDB ストレージエンジンの使用を強制することはできません。

Aurora MySQL Reader DB インスタンスで明示的な一時テーブルを作成する際のエラーを回避するには、次の方法のいずれか、または両方で、すべての `CREATE TEMPORARY TABLE` ステートメントを実行してください。

- `ENGINE=InnoDB` 句を指定しないでください。
- SQL モードを `NO_ENGINE_SUBSTITUTION` に設定しないでください。

ー時テーブル作成エラーと軽減

受け取るエラーは、プレーン CREATE TEMPORARY TABLE ステートメントまたはバリエーション CREATE TEMPORARY TABLE AS SELECT を使うかどうかによって異なります。次の例では、さまざまなタイプのエラーを示しています。

この一時テーブルの動作は、読み取り専用インスタンスにのみ適用されます。この初期の例では、セッションが接続されているインスタンスの種類を確認します。

```
mysql> select @@innodb_read_only;
+--------------------+
| @@innodb_read_only |
+--------------------+
|                  1 |
+--------------------+
```

プレーン CREATE TEMPORARY TABLE ステートメントの場合、NO_ENGINE_SUBSTITUTION SQL モードが有効になっているとステートメントは失敗します。メトリクス NO_ENGINE_SUBSTITUTION がオフ (デフォルト) の場合、適切なエンジン置換が行われ、一時テーブルの作成は成功します。

```
mysql> set sql_mode = 'NO_ENGINE_SUBSTITUTION';

mysql>  CREATE TEMPORARY TABLE tt2 (id int) ENGINE=InnoDB;
ERROR 3161 (HY000): Storage engine InnoDB is disabled (Table creation is disallowed).

mysql> SET sql_mode = '';

mysql> CREATE TEMPORARY TABLE tt4 (id int) ENGINE=InnoDB;

mysql> SHOW CREATE TABLE tt4\G
*************************** 1. row ***************************
       Table: tt4
Create Table: CREATE TEMPORARY TABLE `tt4` (
  `id` int DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

CREATE TEMPORARY TABLE AS SELECT ステートメントの場合、NO_ENGINE_SUBSTITUTION SQL モードが有効になっていると、ステートメントは失敗します。メトリクス NO_ENGINE_SUBSTITUTION がオフ (デフォルト) の場合、適切なエンジン置換が行われ、一時テーブルの作成は成功します。

```
mysql> set sql_mode = 'NO_ENGINE_SUBSTITUTION';

mysql> CREATE TEMPORARY TABLE tt1 ENGINE=InnoDB AS SELECT * FROM t1;
ERROR 3161 (HY000): Storage engine InnoDB is disabled (Table creation is disallowed).

mysql> SET sql_mode = '';

mysql> show create table tt3;
+-------+-----------------------------------------------------------------+
| Table | Create Table                                                    |
+-------+-----------------------------------------------------------------+
| tt3   | CREATE TEMPORARY TABLE `tt3` (
  `id` int DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+-------+-----------------------------------------------------------------+
1 row in set (0.00 sec)
```

Aurora MySQL バージョン 3 での一時テーブルのストレージ側面とパフォーマンスへの影響の詳細については、ブログ記事「Amazon RDS for MySQL および Amazon Aurora MySQL の TempTable ストレージエンジンを使用する」を参照してください。

Aurora MySQL バージョン 2 と Aurora MySQL バージョン 3 の比較

Aurora MySQL バージョン 2 クラスターをバージョン 3 にアップグレードする際に注意すべき変更については、以下を参照してください。

トピック

- Aurora MySQL バージョン 2 と 3 の特徴の違い
- インスタンスクラスのサポート
- Aurora MySQL バージョン 3 のパラメータ変更
- ステータス可変
- Aurora MySQL バージョン 3 に対する包括的な言語変更
- AUTO_INCREMENT 値
- バイナリログレプリケーション

Aurora MySQL バージョン 2 と 3 の特徴の違い

次の Amazon Aurora MySQL 機能は Aurora MySQL 5.7 でサポートされていますが、これらの機能は現在 MySQL 8.0 の Aurora MySQL ではサポートされていません。

- Aurora Serverless v1 クラスターに Aurora MySQL バージョン 3 を使用することはできません。Aurora MySQL バージョン 3 は Aurora Serverless v2 で動作します。
- ラボモードは Aurora MySQL バージョン 3 には適用されません。Aurora MySQL バージョン 3 には、ラボモードの機能はありません。インスタント DDL は、過去にラボモードで使用可能だった高速オンライン DDL特徴よりも優先されます。例については、「インスタント DDL (Aurora MySQL バージョン 3)」を参照してください。
- クエリキャッシュはコミュニティ MySQL 8.0 から削除され、Aurora MySQL バージョン 3 からも削除されます。
- Aurora MySQL バージョン 3 はコミュニティ MySQL ハッシュ統合特徴と互換性があります。ハッシュ結合の Aurora 固有の実装は Aurora MySQL バージョン 2 では使用されていません。ハッシュ結合を Aurora パラレルクエリで使用する方法については、パラレルクエリクラスターのハッシュ結合の有効化 および Aurora MySQL のヒント を参照してください。ハッシュ結合の一般的な使用方法については、MySQL リファレンスマニュアルのハッシュ結合の最適化を参照してください。
- Aurora MySQL バージョン 2 で非推奨であったた `mysql.lambda_async` ストアドプロシージャは、バージョン 3 で削除されます。バージョン 3 では、非同期関数 `lambda_async` を代わりに使用します。
- Aurora MySQL バージョン 3 のデフォルト文字セットは `utf8mb4` です。Aurora MySQL バージョン 2 のデフォルト文字セットは `latin1` です。この文字セットの詳細については、MySQL リファレンスマニュアルのutf8mb4 文字セット (4 バイトの UTF-8 Unicode エンコード)を参照してください。

Aurora MySQL の一部の機能は、AWS リージョンと DB エンジンのバージョンの特定の組み合わせで利用可能です。詳細については、「AWS リージョン と Aurora DB エンジンにより Amazon Aurora でサポートされている機能」を参照してください。

インスタンスクラスのサポート

Aurora MySQL バージョン 3 では、Aurora MySQL バージョン 2 とは異なるインスタンスクラスのセットがサポートされています。

- 大規模なインスタンスの場合は、`db.r5` や `db.r6g`、`db.x2g` のような最新のインスタンスクラスを使用できます。
- 小規模なインスタンスの場合は、`db.t3` や `db.t4g` のような最新のインスタンスクラスを使用できます。

> **Note**
>
> T DB インスタンスクラスは、開発サーバーおよびテストサーバー、またはその他の本稼働以外のサーバーにのみ使用することをお勧めします。T インスタンスクラスの詳細については、「開発やテストのための T インスタンスクラスの使用」を参照してください。

Aurora MySQL バージョン 2 の以下のインスタンスクラスは、Aurora MySQL バージョン 3 では使用できません。

- `db.r4`
- `db.r3`
- `db.t3.small`
- `db.t2`

Aurora MySQL DB インスタンスを作成する CLI ステートメントを、すべて作成していないかどうかを管理スクリプトでチェックします。Aurora MySQL バージョン 3 では利用できないインスタンスクラス名をハードコードします。必要に応じて、Aurora MySQL バージョン 3 がサポートするインスタンス名に、インスタンスクラス名を変更します。

> **Tip**
>
> Aurora MySQL バージョンと AWS リージョンの特定の組み合わせに使えるインスタンスクラスをチェックするには、`describe-orderable-db-instance-options` AWS CLI コマンドを使用して下さい。

Aurora インスタンスクラスの詳細については、Aurora DB インスタンスクラス を参照してください。

Aurora MySQL バージョン 3 のパラメータ変更

Aurora MySQL バージョン 3 には、新しいクラスターレベルおよびインスタンスレベルの設定パラメータが含まれています。Aurora MySQL バージョン 3 では、Aurora MySQL バージョン 2 に存在していたいくつかのパラメータも削除されます。一部のパラメータ名は、包括語のイニシアチブの結果として変更されます。下位互換性のために、古い名前または新しい名前を使用してパラメータ値を取

得できます。ただし、カスタム パラメータグループ内のパラメータ値を指定するには、新しい名前を使用する必要があります。

Aurora MySQL バージョン 3 では、lower_case_table_names パラメータ値はクラスターの作成時に永続的に設定されます。このオプションにデフォルト以外の値を使用する場合は、アップグレードする前に Aurora MySQL バージョン 3 のカスタム パラメータグループを設定します。次に、クラスターの作成またはスナップショットの復元操作中にパラメータグループを指定します。

> **Note**
>
> Aurora MySQL に基づく Aurora グローバルデータベースでは、lower_case_table_names パラメータがオンの場合、Aurora MySQL バージョン 2 からバージョン 3 へのインプレースアップグレードを実行できません。代わりに、スナップショット復元方法を使用してください。

Aurora MySQL バージョン 3 では、init_connect および read_only パラメータは CONNECTION_ADMIN 権限を持つユーザーには適用されません。これには Aurora マスターユーザーが含まれます。詳細については、「ロールベースの特権モデル」を参照してください。

すべての Aurora MySQL クラスター パラメータのリストについては、クラスターレベルのパラメータ を参照してください。この表では、Aurora MySQL バージョン 2 および 3 のすべてのパラメータについて説明します。この表には、Aurora MySQL バージョン 3 で新しく追加されたパラメータや Aurora MySQL バージョン 3 から削除されたパラメータを示す注記が含まれています。

すべての Aurora MySQL インスタンス パラメータのリストについては、インスタンスレベルのパラメータ を参照してください。この表では、Aurora MySQL バージョン 2 および 3 のすべてのパラメータについて説明します。この表には、Aurora MySQL バージョン 3 で新しく追加されたパラメータがどれか、また Aurora MySQL バージョン 3 から削除されたパラメータはどれかを示した注記が含まれています。また、Aurora MySQL バージョン 3 ではなく、以前のバージョンで変更可能なパラメータを示す注記も含まれています。

変更されたパラメータ名の詳細については、Aurora MySQL バージョン 3 に対する包括的な言語変更 を参照してください。

ステータス可変

Aurora MySQL に適用できないステータス可変の詳細については、Aurora MySQL に適応されない MySQL ステータス可変 を参照してください。

Aurora MySQL バージョン 3 に対する包括的な言語変更

Aurora MySQL バージョン 3 は MySQL コミュニティ エディションのバージョン 8.0.23 と互換性があります。Aurora MySQL バージョン 3 には、包括語のキーワードやシステムスキーマに関連した MySQL 8.0.26 からの変更も含まれています。例えば、今では `SHOW SLAVE STATUS` の代わりに`SHOW REPLICA STATUS` コマンドが優先されるようになりました。

次の Amazon CloudWatch メトリクスには、Aurora MySQL バージョン 3 の新しい名前が付けられています。

Aurora MySQL バージョン 3 では、新しいメトリクス名のみを使用できます。Aurora MySQL バージョン 3 にアップグレードするときは、メトリクス名に依存するアラームやその他のオートメーションを必ず更新してください。

現在の名前	新しい名前	
`ForwardingMasterDMLLatency`	`ForwardingWriterDMLLatency`	
`ForwardingMasterOpenSessions`	`ForwardingWriterOpenSessions`	
`AuroraDMLRejectedMasterFull`	`AuroraDMLRejectedWriterFull`	
`ForwardingMasterDMLThroughput`	`ForwardingWriterDMLThroughput`	

Aurora MySQL バージョン 3 では、次のステータス可変に新しい名前が追加されました。

Aurora MySQL バージョン 3 の初期のリリースでは、互換性のためにいずれかの名前を使用できます。古いステータス可変名は、将来のリリースで削除される予定です。

削除される名前	新しい名前または優先される名	
`Aurora_fwd_master_dml_stmt_duration`	`Aurora_fwd_writer_dml_stmt_duration`	

削除される名前	新しい名前または優先される名	
Aurora_fwd_master_dml_stmt_count	Aurora_fwd_writer_dml_stmt_count	
Aurora_fwd_master_select_stmt_duration	Aurora_fwd_writer_select_stmt_duration	
Aurora_fwd_master_select_stmt_count	Aurora_fwd_writer_select_stmt_count	
Aurora_fwd_master_errors_session_timeout	Aurora_fwd_writer_errors_session_timeout	
Aurora_fwd_master_open_sessions	Aurora_fwd_writer_open_sessions	
Aurora_fwd_master_errors_session_limit	Aurora_fwd_writer_errors_session_limit	
Aurora_fwd_master_errors_rpc_timeout	Aurora_fwd_writer_errors_rpc_timeout	

Aurora MySQL バージョン 3 では、次の設定パラメータに新しい名前が付けられました。

互換性のため、mysql クライアントのパラメータ値をチェックするには、Aurora MySQL バージョン 3 の初期のリリースでいずれかの名前を使用します。カスタムパラメータグループ内の値を変更する場合、新しい名前のみ使用できます。古いパラメータ名は、将来のリリースで削除される予定です。

削除される名前	新しい名前または優先される名
aurora_fwd_master_idle_timeout	aurora_fwd_writer_idle_timeout
aurora_fwd_master_max_connections_pct	aurora_fwd_writer_max_connections_pct
master_verify_checksum	source_verify_checksum
sync_master_info	sync_source_info
init_slave	init_replica
rpl_stop_slave_timeout	rpl_stop_replica_timeout
log_slow_slave_statements	log_slow_replica_statements
slave_max_allowed_packet	replica_max_allowed_packet
slave_compressed_protocol	replica_compressed_protocol
slave_exec_mode	replica_exec_mode
slave_type_conversions	replica_type_conversions
slave_sql_verify_checksum	replica_sql_verify_checksum
slave_parallel_type	replica_parallel_type

削除される名前	新しい名前または優先される名	
slave_preserve_commit_order	replica_preserve_commit_order	
log_slave_updates	log_replica_updates	
slave_allow_batching	replica_allow_batching	
slave_load_tmpdir	replica_load_tmpdir	
slave_net_timeout	replica_net_timeout	
sql_slave_skip_counter	sql_replica_skip_counter	
slave_skip_errors	replica_skip_errors	
slave_checkpoint_period	replica_checkpoint_period	
slave_checkpoint_group	replica_checkpoint_group	
slave_transaction_retries	replica_transaction_retries	
slave_parallel_workers	replica_parallel_workers	
slave_pending_jobs_size_max	replica_pending_jobs_size_max	
pseudo_slave_mode	pseudo_replica_mode	

Aurora MySQL バージョン 3 では、次のストアドプロシージャは新しい名前になっています。

Aurora MySQL バージョン 3 の初期のリリースでは、互換性のためにいずれかの名前を使用できます。以前のプロシージャ名は、将来のリリースで削除される予定です。

削除される名前	新しい名前または優先される名
mysql.rds_set_master_auto_position	mysql.rds_set_source_auto_position
mysql.rds_set_external_master	mysql.rds_set_external_source
mysql.rds_set_external_master_with_auto_position	mysql.rds_set_external_source_with_auto_position
mysql.rds_reset_external_master	mysql.rds_reset_external_source
mysql.rds_next_master_log	mysql.rds_next_source_log

AUTO_INCREMENT 値

Aurora MySQL バージョン 3 では、各 DB インスタンスを再起動する際、Aurora は 各テーブルの AUTO_INCREMENT 値を保持します。Aurora MySQL バージョン 2 では、再起動後に AUTO_INCREMENT 値が保持されませんでした。

スナップショットからの復元や、ポイントインタイムリカバリの実行、およびクラスターのクローン作成によって新しいクラスターを設定した場合、AUTO_INCREMENT 値は保持されません。この場合の AUTO_INCREMENT 値は、スナップショットが作成された時点のテーブル内の最大列値に基づいた値に初期化されます。この動作は、RDS for MySQL 8.0 では異なり、AUTO_INCREMENT 値はこれらのオペレーション中に保持されます。

バイナリログレプリケーション

MySQL 8.0 コミュニティエディションでは、バイナリログのレプリケーションがデフォルトで有効になっています。Aurora MySQL バージョン 3 では、バイナリログレプリケーションはデフォルトで無効になっています。

> **Tip**
>
> Aurora 組み込みレプリケーション機能によって高可用性要件が満たされている場合は、バイナリログレプリケーションをオフにしておくことができます。これにより、バイナリログレプリケーションのパフォーマンスオーバーヘッドを回避できます。また、バイナリログレプリケーションの管理に必要な、関連するモニタリングおよびトラブルシューティングを回避することもできます。

Aurora は MySQL 5.7 互換出典から Aurora MySQL バージョン 3 へのバイナリログレプリケーションをサポートしています。出典システムは、Aurora MySQL DB クラスター、RDS for MySQL DB インスタンス、またはオンプレミス MySQL インスタンスです。

コミュニティ MySQL と同様に、Aurora MySQL は特定のバージョンを実行している出典から、同じメジャーバージョンまたは 1 つ以上のメジャーバージョンを実行するターゲットへのレプリケーションをサポートします。例えば、MySQL 5.6 互換システムから Aurora MySQL バージョン 3 へのレプリケーションはサポートされていません。Aurora MySQL バージョン 3 から MySQL 5.7 互換または MySQL 5.6 互換システムへのレプリケーションはサポートされていません。バイナリログのレプリケーションの詳細については、Aurora と MySQL との間、または Aurora と別の Aurora DB クラスターとの間のレプリケーション (バイナリログレプリケーション) を参照してください。

Aurora MySQL バージョン 3 には、フィルタリングされたレプリケーションなど、コミュニティ MySQL 8.0 でのバイナリログレプリケーションの改善が含まれています。コミュニティ MySQL 8.0 の改善の詳細については、MySQL リファレンスマニュアルのサーバーがレプリケーションフィルタ規則を評価する方法を参照してください。

マルチスレッドレプリケーション

Aurora MySQL バージョン 3 では、Aurora MySQL はマルチスレッドレプリケーションをサポートします。使用に関する情報については、「マルチスレッドバイナリログレプリケーション」を参照してください。

> **Note**
>
> Aurora MySQL バージョン 2 では、マルチスレッドレプリケーションを使用しないことをお勧めします。

バイナリログレプリケーションのトランザクション圧縮

バイナリログ圧縮の使用方法については、MySQL リファレンスマニュアルのバイナリログトランザクションの圧縮を参照してください。

Aurora MySQL バージョン 3 のバイナリログ圧縮には、次の制限が適用されます。

- バイナリログデータが最大許容パケットサイズより大きいトランザクションは圧縮されません。これは、Aurora MySQL バイナリログ圧縮設定が有効になっているかどうかに関係ありません。このようなトランザクションは、圧縮されずに複製されます。
- MySQL 8.0 をまだサポートしていない変更データキャプチャ (CDC) にコネクターを使用している場合、この特徴は使用できません。サードパーティーのコネクターをバイナリログ圧縮でしっかりテストした後に、バイナリログ圧縮を有効にすることをお勧めします。また、CDC の binlog レプリケーションを使用するシステムでバイナリログ圧縮を有効にすることをお勧めします。

Aurora MySQL バージョン 3 と MySQL 8.0 コミュニティエディションの比較

次の情報を使用して、別の MySQL 8.0 互換システムから Aurora MySQL バージョン 3 に変換する際の、注意すべき変更について知ることができます。

一般に、Aurora MySQL バージョン 3 はコミュニティ MySQL 8.0.23 の特徴セットをサポートしています。MySQL 8.0 コミュニティエディションの一部の新機能は Aurora MySQL には適用されません。これらの機能の一部は、Aurora ストレージアーキテクチャなど Aurora の一部の側面と互換性がありません。Amazon RDS 管理サービスで同等の機能が提供されるため、その他の機能は必要ありません。コミュニティ MySQL 8.0 の次の機能は、Aurora MySQL バージョン 3 でサポートされていない、あるいは異なる動作をします。

すべての Aurora MySQL バージョン 3 リリースのリリースノートについては、Aurora MySQL のリリースノートの「Amazon Aurora MySQL バージョン 3 のデータベースエンジンの更新」を参照してください。

トピック

- MySQL 8.0 の機能はAurora MySQL バージョン 3 では利用できません
- ロールベースの特権モデル
- 認証

MySQL 8.0 の機能はAurora MySQL バージョン 3 では利用できません

コミュニティ MySQL 8.0 の次の機能は、Aurora MySQL バージョン 3 では利用できないか、異なる動作をします。

- リソースグループと関連付けられた SQL ステートメントは、Aurora MySQL ではサポートされていません。
- Aurora MySQL は、ユーザー定義の UNDO テーブルスペースおよび関連する SQL ステートメント (`CREATE UNDO TABLESPACE`、`ALTER UNDO TABLESPACE ... SET INACTIVE` および `DROP UNDO TABLESPACE` など) をサポートしていません。
- Aurora MySQL は 3.06 より前のバージョンの Aurora MySQL の UNDO テーブルスペースの切り捨てをサポートしていません。Aurora MySQL バージョン 3.06 以降では、UNDO テーブルスペースの自動切り捨てがサポートされています。
- MySQL プラグインの設定は変更できません。
- X プラグインはサポートされていません。
- マルチソースレプリケーションはサポートされません。

ロールベースの特権モデル

Aurora MySQL バージョン 3 では、`mysql` データベースのテーブルを直接変更できません。特に、`mysql.user` テーブルへの挿入によるユーザ設定ができません。代わりに、SQL 文を使用してロールベースの権限を付与します。また、`mysql` データベースでストアドプロシージャなど、他の種類のオブジェクトを作成することはできません。`mysql` テーブルにクエリを実行することはできます。バイナリログ レプリケーションを使用する場合、出典 クラスターの `mysql` テーブルへの直接的な変更は、ターゲット クラスターにレプリケートされません。

場合によっては、`mysql` テーブルに挿入することで、アプリケーションがショートカットを使用して、ユーザーやその他のオブジェクトを作成する場合があります。その場合は、アプリケーションコードを変更して、`CREATE USER` などの対応したステートメントを使用します。アプリケーションが `mysql` データベースでストアドプロシージャまたはその他のオブジェクトを作成する場合は、代わりに別のデータベースを使用してください。

外部 MySQL データベースからの移行中にデータベースユーザーのメタデータを エクスポートするには、`mysqldump` の代わりに MySQL Shell コマンドを使用します。詳細については、「インスタンスダンプユーティリティ、スキーマダンプユーティリティ、テーブルダンプユーティリティ」を参照してください。

多くのユーザーまたはアプリケーションの権限の管理を簡素化するには、CREATE ROLE ステートメントを使用して、一連の権限を持つロールを作成します。その後、GRANT および SET ROLE ステートメントと current_role 関数を使用して、ユーザーまたはアプリケーションにロールを割り当てたり、現在のロールを切り替えたり、有効なロールをチェックしたりできます。MySQL 8.0 でのロールベースのアクセス許可システムの詳細については、MySQL リファレンスマニュアルのロールの使用を参照してください。

> **Important**
>
> アプリケーションではマスターユーザーを直接使用しないことを強くお勧めします。代わりに、アプリケーションに必要な最小の特権で作成されたデータベースユーザーを使用するというベストプラクティスに従ってください。

Aurora MySQL バージョン 3 には、以下のすべての権限を持つ特別なロールが含まれています。ロールの名前は rds_superuser_role です。各クラスターのプライマリ管理ユーザーには、既にこのロールが付与されています。rds_superuser_role ロールには、すべてのデータベースオブジェクトに対する次の権限が含まれます。

- ALTER
- APPLICATION_PASSWORD_ADMIN
- ALTER ROUTINE
- CONNECTION_ADMIN
- CREATE
- CREATE ROLE
- CREATE ROUTINE
- CREATE TEMPORARY TABLES
- CREATE USER
- CREATE VIEW
- DELETE
- DROP
- DROP ROLE
- EVENT
- EXECUTE

- INDEX
- INSERT
- LOCK TABLES
- PROCESS
- REFERENCES
- RELOAD
- REPLICATION CLIENT
- REPLICATION SLAVE
- ROLE_ADMIN
- SET_USER_ID
- SELECT
- SHOW DATABASES
- SHOW_ROUTINE (Aurora MySQL バージョン 3.04 以降)
- SHOW VIEW
- TRIGGER
- UPDATE
- XA_RECOVER_ADMIN

ロール定義には WITH GRANT OPTION が含まれるため、管理ユーザーはそのロールを他のユーザーに付与することができます。特に、Aurora MySQL クラスターをターゲットとしてバイナリログレプリケーションを実行するために必要な権限を、管理者は付与する必要があります。

> **Tip**
>
> 権限の詳細全体を表示するには、次のステートメントを入力します。
>
> ```
> SHOW GRANTS FOR rds_superuser_role@'%';
> SHOW GRANTS FOR name_of_administrative_user_for_your_cluster@'%';
> ```

Aurora MySQL バージョン 3 には、他の AWS サービスにアクセスするために使用できるロールも含まれています。これらのロールは、GRANT ステートメントの代替として設定できます。例えば、GRANT INVOKE LAMBDA ON *.* TO *user* の代わりに GRANT AWS_LAMBDA_ACCESS TO

user を指定します。他の AWS サービスにアクセスする手順については、Amazon Aurora MySQL と他の AWS のサービスの統合 を参照してください。Aurora MySQLバージョン 3 には、他の AWS サービスへのアクセスに関連する次のロールが含まれています。

- INVOKE LAMBDA 権限の代替としての、AWS_LAMBDA_ACCESS ロール。使用に関する情報については、Amazon Aurora MySQL DB クラスターからの Lambda 関数の呼び出し を。
- LOAD FROM S3 権限の代替としての、AWS_LOAD_S3_ACCESS ロール。使用に関する情報については、「Amazon S3 バケットのテキストファイルから Amazon Aurora MySQL DB クラスターへのデータのロード」を参照してください。
- SELECT INTO S3 権限の代替としての、AWS_SELECT_S3_ACCESS ロール。使用に関する情報については、「Amazon Aurora MySQL DB クラスターから Amazon S3 バケット内のテキストファイルへのデータの保存」を参照してください。
- INVOKE SAGEMAKER 権限の代替としての、AWS_SAGEMAKER_ACCESS ロール。使用に関する情報については、「Aurora MySQL で Amazon Aurora 機械学習を使用する」を参照してください。
- INVOKE COMPREHEND 権限の代替としての、AWS_COMPREHEND_ACCESS ロール。使用に関する情報については、「Aurora MySQL で Amazon Aurora 機械学習を使用する」を参照してください。

Aurora MySQL バージョン 3 でロールを使用してアクセスを許可する場合は、SET ROLE *role_name* または SET ROLE ALL ステートメントを使用してロールも有効化します。以下の例のように指定します。AWS_SELECT_S3_ACCESS 用に適切なロール名を置き換えます。

```
# Grant role to user.
mysql> GRANT AWS_SELECT_S3_ACCESS TO 'user'@'domain-or-ip-address'

# Check the current roles for your user. In this case, the AWS_SELECT_S3_ACCESS role
 has not been activated.
# Only the rds_superuser_role is currently in effect.
mysql> SELECT CURRENT_ROLE();
+--------------------------+
| CURRENT_ROLE()           |
+--------------------------+
| `rds_superuser_role`@`%` |
+--------------------------+
1 row in set (0.00 sec)

# Activate all roles associated with this user using SET ROLE.
# You can activate specific roles or all roles.
# In this case, the user only has 2 roles, so we specify ALL.
```

```
mysql> SET ROLE ALL;
Query OK, 0 rows affected (0.00 sec)

# Verify role is now active
mysql> SELECT CURRENT_ROLE();
+-------------------------------------------------+
| CURRENT_ROLE()                                  |
+-------------------------------------------------+
| `AWS_LAMBDA_ACCESS`@`%`,`rds_superuser_role`@`%` |
+-------------------------------------------------+
```

認証

コミュニティ MySQL 8.0 では、デフォルトの認証プラグインは caching_sha2_password です。Aurora MySQL バージョン3では、mysql_native_password プラグインがまだ使用されます。default_authentication_plugin 設定は変更できません。

Aurora MySQL バージョン 3 へのアップグレード

Aurora MySQL バージョン 2 からバージョン 3 にデータベースをアップグレードする方法については、「Amazon Aurora MySQL DB クラスターのメジャーバージョンのアップグレード」を参照してください。

MySQL 5.7 互換 Aurora MySQL バージョン 2

このトピックでは、Aurora MySQL バージョン 2 と MySQL 5.7 コミュニティエディションの違いについて説明します。

Aurora MySQL バージョン 2 ではサポートされていない機能

次の機能は MySQL 5.7 ではサポートされていますが、現在、Aurora MySQL バージョン 2 ではサポートされていません。

- CREATE TABLESPACE SQL ステートメント
- グループのレプリケーションプラグイン
- ページサイズの増加
- 起動時の InnoDB バッファープールのロード
- InnoDB フルテキストパーサープラグイン
- マルチソースレプリケーション

- オンラインバッファープールのサイズ変更
- パスワード検証プラグイン – プラグインをインストールできますが、サポートされていません。プラグインをカスタマイズすることはできません。
- クエリ書き換えプラグイン
- レプリケーションフィルタリング
- X プロトコル

これらの機能のの詳細については、MySQL 5.7 のドキュメントを参照してください。

Aurora MySQL バージョン 2 での一時テーブルスペースの動作

MySQL 5.7 では、一時テーブルスペースは自動的に拡張され、ディスク上の一時テーブルに対応できるように必要に応じてサイズが大きくなります。一時テーブルが削除されると、空いたスペースを新しい一時テーブルに再利用できますが、一時テーブルスペースは拡張されたサイズのままで縮小しません。一時テーブルスペースは、エンジンを再起動すると削除され、再作成されます。

Aurora MySQL バージョン 2 では、以下の動作が適用されます。

- バージョン 2.10 以降で作成された新しい Aurora MySQL DB クラスターの場合、一時テーブルスペースは、データベースを再起動したときに削除され、再作成されます。これにより、動的なサイズ変更機能でストレージスペースを再利用できます。
- 既存の Aurora MySQL DB クラスターを以下のようにアップグレードした場合:
  - バージョン 2.10 以上 - 一時テーブルスペースは、データベースを再起動したときに削除され、再作成されます。これにより、動的なサイズ変更機能でストレージスペースを再利用できます。
  - バージョン 2.09 — データベースを再起動しても、一時テーブルスペースは削除されません。

Aurora MySQL バージョン 2 の DB クラスターの一時テーブルスペースのサイズは、次のクエリを使用して確認できます。

```
SELECT
    FILE_NAME,
    TABLESPACE_NAME,
    ROUND((TOTAL_EXTENTS * EXTENT_SIZE) / 1024 / 1024 / 1024, 4) AS SIZE
FROM
    INFORMATION_SCHEMA.FILES
WHERE
    TABLESPACE_NAME = 'innodb_temporary';
```

詳細については、MySQL ドキュメントの「一時テーブルスペース」を参照してください。

オンディスク一時テーブルのストレージエンジン

Aurora MySQL バージョン 2 は、インスタンスのロールに応じて、オンディスク内部一時テーブルに異なるストレージエンジンを使用します。

- ライターインスタンスでは、オンディスク一時テーブルはデフォルトで InnoDB ストレージエンジンを使用します。これらは Aurora クラスターボリュームの一時テーブルスペースに保存されます。

  DB パラメータ `internal_tmp_disk_storage_engine` の値を変更することで、ライターインスタンスのこの動作を変更できます。詳細については、「インスタンスレベルのパラメータ」を参照してください。
- リーダーインスタンスでは、オンディスク一時テーブルはローカルストレージを使用する MyISAM ストレージエンジンを使用します。これは、読み取り専用インスタンスでは Aurora クラスターボリュームにデータを保存できないためです。

Amazon Aurora MySQL でのセキュリティ

Amazon Aurora MySQL のセキュリティは次の 3 つのレベルで管理されます。

- Aurora MySQL DB クラスターと DB インスタンスに対する Amazon RDS 管理アクションを実行できるユーザーを管理するには、AWS Identity and Access Management (IAM) を使用します。IAM 認証情報を使用して AWS に接続している場合、AWS アカウントには、Amazon RDS の管理オペレーションを実行するためのアクセス許可を付与する IAM ポリシーが必要です。詳細については、「Amazon Aurora での Identity and Access Management」を参照してください。

  IAM を使用して Amazon RDS コンソールにアクセスする場合は、まず、IAM ユーザーの認証情報を使用して AWS Management Console にサインインしてください。次に、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) に移動します。

- Amazon VPC サービスに基づいて仮想パブリッククラウド (VPC) に Aurora MySQL DB クラスターを作成します。VPC 内の Aurora MySQL DB クラスター用の DB インスタンスのエンドポイントとポートに対して接続を開くことができるデバイスと Amazon EC2 インスタンスを制御するには、VPC セキュリティグループを使用します。これらのエンドポイントおよびポート接続には、Transport Layer Security (TLS) を使用できます。さらに、会社のファイアウォールルールでも、社内のいずれのデバイスが DB インスタンスへの接続を開くことができるかを制御できます。VPC の詳細については、「Amazon VPC VPC とAmazon Aurora」を参照してください。

  サポートされている VPC テナンシーは、Aurora MySQL DB クラスターで使用している DB インスタンスクラスによって異なります。`default` VPC テナントでは、VPC は共有ハードウェアで実行されます。`dedicated` VPC テナンシーでは、VPC は専有のハードウェアインスタンスで実行されます。バースト可能パフォーマンス DB インスタンスクラスは、デフォルト VPC テナンシーのみをサポートしています。バースト可能パフォーマンス DB インスタンスクラスには、db.t2、db.t3、および db.t4g DB インスタンスクラスが含まれます。その他すべての Aurora MySQL DB インスタンスクラスでは、デフォルトおよび専用 VPC テナンシーの両方がサポートされています。

> **Note**
>
> T DB インスタンスクラスを、開発サーバーおよびテストサーバー、または他の本稼働以外のサーバーにのみ使用することをお勧めします。T インスタンスクラスの詳細については、「開発やテストのための T インスタンスクラスの使用」を参照してください。

インスタンスクラスの詳細については、「Aurora DB インスタンスクラス」を参照してください。default および dedicated VPC テナントの詳細については、Amazon Elastic Compute Cloud ユーザーガイドの「ハードウェア専有インスタンス」を参照してください。

- Amazon Aurora MySQL DB クラスターへのログインとアクセス権限を認証するには、以下の方法のいずれかを使用するか、複数を組み合わせて使用します。
  - MySQL のスタンドアロンインスタンスの場合と同じ方法を使用できます。

    CREATE USER、RENAME USER、GRANT、REVOKE、SET PASSWORD などのコマンドは、オンプレミスデータベースでの方法と同様に、データベーススキーマテーブルを直接変更します。詳細については、MySQL ドキュメントの「アクセス制御およびアカウントマネジメント」を参照してください。
  - IAM データベース認証を使用することもできます。

    IAM データベース認証を使用する場合は、IAM ユーザーと IAM ロールのいずれかと認証トークンを使用して、DB クラスターを認証します。認証トークン は、署名バージョン 4 の署名プロセスを使用して生成されている一意の値です。IAM データベース認証では、同一の認証情報を使用して AWS リソースおよびデータベースへのアクセスを制御できます。詳細については、「 の IAM データベース認証」を参照してください。

> **Note**
>
> 詳細については、「Amazon Aurora でのセキュリティ」を参照してください。

Amazon Aurora MySQL でのマスターユーザー特権

Amazon Aurora MySQL DB インスタンスを作成すると、マスターユーザーには マスターユーザーアカウント権限 にリストされているデフォルト権限が付与されます。

各 DB クラスターに管理サービスを提供するために、DB クラスターの作成時に admin および rdsadmin ユーザーが作成されます。rdsadmin アカウントの削除、名前の変更、パスワードの変更、または権限の変更を行おうとすると、エラーになります。

Aurora MySQL バージョン 2 の DB クラスターでは、DB クラスターの作成時に admin および rdsadmin ユーザーが作成されます。Aurora MySQL バージョン 3 の DB クラスターでは、admin、rdsadmin、および rds_superuser_role ユーザーが作成されます。

> **Important**
>
> アプリケーションではマスターユーザーを直接使用しないことを強くお勧めします。代わりに、アプリケーションに必要な最小の特権で作成されたデータベースユーザーを使用するというベストプラクティスに従ってください。

Aurora MySQL DB クラスターの管理では、スタンダード的なコマンド `kill` と `kill_query` の使用は制限されています。Aurora MySQL DB インスタンスのユーザーセッションまたはクエリを終了するには、Amazon RDS の `rds_kill` コマンドと `rds_kill_query` コマンドを代わりに使用します。

> **Note**
>
> データベースインスタンスおよびスナップショットの暗号化は、中国 (寧夏) リージョンではサポートされていません。

Aurora MySQL DB クラスターでの TLS の使用

Amazon Aurora MySQL DB クラスターは、RDS for MySQL DB インスタンスと同じプロセスと公開鍵を使用したアプリケーションからの Transport Layer Security (TLS) 接続をサポートします。

Amazon RDS は、Amazon RDS がインスタンスをプロビジョニングするときに、TLS 証明書を作成し、その証明書を DB インスタンスにインストールします。これらの証明書は認証局によって署名されます。TLS 証明書には、なりすまし攻撃から保護するために、TLS 証明書の共通名 (CN) として DB インスタンスのエンドポイントが含まれています。その結果、クライアントがサブジェクト代替名 (SAN) をサポートしている場合は、DB クラスターエンドポイントのみを使用して、TLS を使用する DB クラスターに接続できます。それ以外の場合は、書き込みインスタンスのインスタンスを使用する必要があります。

証明書のダウンロードについては、SSL/TLS を使用した DB クラスターへの接続の暗号化 を参照してください。

TLS での SAN をサポートするクライアントとして、AWS JDBC ドライバーを推奨します。AWS JDBC ドライバーおよびその使用方法の詳細については、「Amazon Web Services (AWS) JDBC ドライバー GitHub リポジトリ」を参照してください。

トピック

Aurora MySQL DB クラスターへの TLS 接続の要求

`require_secure_transport` DB クラスターパラメータを使用することによって、Aurora MySQL DB クラスターへのすべてのユーザー接続が TLS を使用するように要求できます。デフォルトでは、`require_secure_transport` パラメータが `OFF` に設定されています。`require_secure_transport` パラメータを `ON` に設定して、DB クラスターへの接続で TLS を必須にすることができます。

`require_secure_transport` パラメータの値は、DB クラスターの DB クラスターパラメータグループを更新することで設定できます。変更を有効にするために、DB クラスターを再起動する必要はありません。パラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。

> **Note**
>
> `require_secure_transport` パラメータは、Aurora MySQL バージョン 2 および 3 で使用できます。このパラメータは、カスタム DB クラスターのパラメータグループで設定できます。このパラメータは、DB インスタンスパラメータグループでは使用できません。

DB クラスターに対して `require_secure_transport` パラメータが `ON` に設定されている場合、データベースクライアントが暗号化された接続を確立できれば、データベースクライアントはそのクラスターに接続できます。それ以外の場合は、次のようなエラーメッセージがクライアントに返されます。

```
MySQL Error 3159 (HY000): Connections using insecure transport are prohibited while --
require_secure_transport=ON.
```

Aurora MySQL の TLS バージョン

Aurora MySQL は Transport Layer Security (TLS) バージョン 1.0、1.1、1.2、および 1.3 をサポートしています。Aurora MySQL バージョン 3.04.0 以降では、TLS 1.3 プロトコルを使用して接続を保護できます。次の表は、Aurora MySQL バージョンの TLS サポートを示しています。

Aurora MySQL バージョン	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	デフォルト値
Aurora MySQL バージョン 2	サポート	サポート	サポート	サポートされていません	サポートされているすべての TLS バージョン
Aurora MySQL バージョン 3 (3.04.0 より前のバージョン)	サポート	サポート	サポート	サポートされていません	サポートされているすべての TLS バージョン
Aurora MySQL バージョン 3 (3.04.0 以降)	サポートされていません	サポートされていません	サポート	サポート	サポートされているすべての TLS バージョン

> **Important**
>
> バージョン 2 と 3.04.0 より前のバージョンの Aurora MySQL クラスターでカスタムパラメータグループを使用している場合、TLS 1.0 と 1.1 は安全性が低いため、TLS 1.2 を使用することをお勧めします。MySQL 8.0.26 と Aurora MySQL 3.03 のコミュニティエディションとそのマイナーバージョンでは、TLS バージョン 1.1 と 1.0 のサポートが廃止されました。MySQL 8.0.28 のコミュニティエディションと互換性のある Aurora MySQL バージョン 3.04.0 以降は、TLS 1.1 と TLS 1.0 をサポートしていません。Aurora MySQL バージョン 3.04.0 以降を使用している場合は、カスタムパラメータグループで TLS プロトコルを 1.0 と 1.1 に設定しないでください。
>
> Aurora MySQL バージョン 3.04.0 以降では、デフォルト設定は TLS 1.3 と TLS 1.2 です。

`tls_version` DB クラスターパラメータを使用して、許可されたプロトコルバージョンを指定できます。ほとんどのクライアントツールまたはデータベースドライバには、同様のクライアントパラ

メータがあります。古いクライアントの中には、新しい TLS バージョンをサポートしていないものもあります。デフォルトでは、DB クラスターは、サーバーとクライアントの設定の両方で許可されている最高の TLS プロトコルバージョンを使用しようとします。

`tls_version` DB クラスターパラメータを次のいずれかの値に設定します。

- `TLSv1.3`
- `TLSv1.2`
- `TLSv1.1`
- `TLSv1`

`tls_version` パラメータは、カンマで区切られたリストの文字列として設定することもできます。TLS 1.2 プロトコルと TLS 1.0 プロトコルの両方を使用する場合は、`tls_version` パラメータに最下位プロトコルから最上位プロトコルまでのすべてのプロトコルを含める必要があります。この場合、`tls_version` は次のように設定されます。

```
tls_version=TLSv1,TLSv1.1,TLSv1.2
```

DB クラスターのパラメータグループの変更については、「DB クラスターパラメータグループのパラメータの変更」を参照してください。AWS CLI を使用する場合、`tls_version` DB クラスターのパラメータの変更には、`ApplyMethod` に `pending-reboot` を設定している必要があります。アプリケーションのメソッドが `pending-reboot` の場合は、パラメータグループに関連付けられている DB クラスターを停止および再起動した後に、パラメータの変更が適用されます。

Aurora PostgreSQL DB クラスターへの接続用暗号スイートを設定する

設定可能な暗号スイートを使用すると、データベース接続のセキュリティをより詳細に制御できます。データベースへのクライアント TLS 接続を保護するために許可する暗号スイートのリストを指定できます。設定可能な暗号スイートを使用すると、データベースサーバーが受け入れる接続暗号化を制御できます。これにより、安全でない暗号や非推奨の暗号の使用を防ぐことができます。

設定可能な暗号スイートは、Aurora MySQL バージョン 3 および Aurora MySQL バージョン 2 でサポートされています。接続を暗号化するために許容できる TLS 1.2、TLS 1.1、TLS 1.0 暗号のリストを指定するには、`ssl_cipher` クラスターパラメータを変更します。AWS Management Console、AWS CLI、または RDS API を使用して、クラスターパラメータグループ内の `ssl_cipher` パラメータを設定します。

ssl_cipher パラメータを、TLS バージョンのカンマ区切りの暗号値の文字列に設定します。クライアントアプリケーションでは、データベースに接続するときに --ssl-cipher オプションを使用することによって、暗号化接続に使用する暗号を指定できます。データベースへの接続の詳細については、「Amazon Aurora MySQL DB クラスターへの接続」を参照してください。

Aurora MySQL バージョン 3.04.0 以降では、TLS 1.3 暗号スイートを指定できるようになりました。許容される TLS 1.3 暗号スイートを指定するには、パラメータグループの tls_ciphersuites パラメータを変更します。TLS 1.3 では、命名規則が変更され、鍵交換メカニズムと証明書が使用されなくなったため、使用可能な暗号スイートの数が減少しました。tls_ciphersuites パラメータを、カンマ区切りの TLS 1.3 の暗号値の文字列に設定します。

次の表に、サポートされている暗号と、各暗号の TLS 暗号化プロトコルおよび有効な Aurora MySQL エンジンのバージョンを示します。

暗号	暗号化プロトコル	サポートされている Aurora MySQL バージョン
DHE-RSA-AES128-SHA	TLS 1.0	3.01.0 以上、2.11.0 未満のすべて
DHE-RSA-AES128-SHA256	TLS 1.2	3.01.0 以上、2.11.0 未満のすべて
DHE-RSA-AES128-GCM-SHA256	TLS 1.2	3.01.0 以上、2.11.0 未満のすべて
DHE-RSA-AES256-SHA	TLS 1.0	3.03.0 以下、2.11.0 未満のすべて
DHE-RSA-AES256-SHA256	TLS 1.2	3.01.0 以上、2.11.0 未満のすべて
DHE-RSA-AES256-GCM-SHA384	TLS 1.2	3.01.0 以上、2.11.0 未満のすべて
ECDHE-RSA-AES128-SHA	TLS 1.0	3.01.0 以降、2.09.3 以降、2.10.2 以降

暗号	暗号化プロトコル	サポートされている Aurora MySQL バージョン
ECDHE-RSA-AES128-SHA256	TLS 1.2	3.01.0 以降、2.09.3 以降、2.10.2 以降
ECDHE-RSA-AES128-GCM-SHA256	TLS 1.2	3.01.0 以降、2.09.3 以降、2.10.2 以降
ECDHE-RSA-AES256-SHA	TLS 1.0	3.01.0 以降、2.09.3 以降、2.10.2 以降
ECDHE-RSA-AES256-SHA384	TLS 1.2	3.01.0 以降、2.09.3 以降、2.10.2 以降
ECDHE-RSA-AES256-GCM-SHA384	TLS 1.2	3.01.0 以降、2.09.3 以降、2.10.2 以降
TLS_AES_128_GCM_SHA256	TLS 1.3	3.04.0 以上
TLS_AES_256_GCM_SHA384	TLS 1.3	3.04.0 以上
TLS_CHACHA20_POLY1305_SHA256	TLS 1.3	3.04.0 以上

> **Note**
>
> DHE-RSA 暗号は、2.11.0 より前のバージョンの Aurora MySQL でのみサポートされています。バージョン 2.11.0 以上では、ECDHE 暗号のみがサポートされています。

DB クラスターのパラメータグループの変更については、「DB クラスターパラメータグループのパラメータの変更」を参照してください。CLI を使用して ssl_cipher DB クラスターのパラメータを変更する場合には、ApplyMethod を pending-reboot に設定してください。アプリケーションのメソッドが pending-reboot の場合は、パラメータグループに関連付けられている DB クラスターを停止および再起動した後に、パラメータの変更が適用されます。

また、CLI コマンドの describe-engine-default-cluster-parameters を使用して、特定のパラメータグループファミリーで現在サポートされている暗号スイートを特定することもできます。次の例は、Aurora MySQL バージョン 2 の ssl_cipher クラスターパラメータで許可される値を取得する方法を示しています。

```
aws rds describe-engine-default-cluster-parameters --db-parameter-group-family aurora-
mysql5.7

        ...some output truncated...
 {
  "ParameterName": "ssl_cipher",
  "ParameterValue": "DHE-RSA-AES128-SHA,DHE-RSA-AES128-SHA256,DHE-RSA-AES128-GCM-
SHA256,DHE-RSA-AES256-SHA,DHE-RSA-AES256-SHA256,DHE-RSA-AES256-GCM-SHA384,ECDHE-RSA-
AES128-SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-AES128-GCM-SHA256,ECDHE-RSA-AES256-
SHA,ECDHE-RSA-AES256-SHA384,ECDHE-RSA-AES256-GCM-SHA384",
  "Description": "The list of permissible ciphers for connection encryption.",
  "Source": "system",
  "ApplyType": "static",
  "DataType": "list",
  "AllowedValues": "DHE-RSA-AES128-SHA,DHE-RSA-AES128-SHA256,DHE-RSA-AES128-GCM-
SHA256,DHE-RSA-AES256-SHA,DHE-RSA-AES256-SHA256,DHE-RSA-AES256-GCM-SHA384,ECDHE-
RSA-AES128-SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-AES128-GCM-SHA256,ECDHE-RSA-AES256-
SHA,ECDHE-RSA-AES256-SHA384,ECDHE-RSA-AES256-GCM-SHA384",
  "IsModifiable": true,
  "SupportedEngineModes": [
   "provisioned"
  ]
 },
       ...some output truncated...
```

暗号の詳細については、MySQL ドキュメントの「ssl_cipher」を参照してください。暗号スイートの形式の詳細については、OpenSSL ウェブサイトの「openssl 暗号リスト形式」と「openssl 暗号文字列形式」のドキュメントを参照してください。

Aurora MySQL DB クラスターへの接続の暗号化

デフォルトの mysql クライアントを使用して接続を暗号化するには、以下の例のように、--ssl-ca パラメータを使用して mysql クライアントを起動します。

MySQL 5.7 および 8.0 向け:

```
mysql -h myinstance.123456789012.rds-us-east-1.amazonaws.com
```

```
--ssl-ca=full_path_to_CA_certificate --ssl-mode=VERIFY_IDENTITY
```

MySQL 5.6 向け:

```
mysql -h myinstance.123456789012.rds-us-east-1.amazonaws.com
--ssl-ca=full_path_to_CA_certificate --ssl-verify-server-cert
```

full_path_to_CA_certificate を、認証局 (CA) の証明書に対する完全なパスに置き換えます。証明書のダウンロードについては、「SSL/TLS を使用した DB クラスターへの接続の暗号化」を参照してください。

特定のユーザーアカウントに対して TLS 接続を要求できます。例えば、MySQL バージョンに応じて以下のいずれかのステートメントを使用し、ユーザーアカウント encrypted_user に対して TLS 接続を要求できます。

MySQL 5.7 および 8.0 向け:

```
ALTER USER 'encrypted_user'@'%' REQUIRE SSL;
```

MySQL 5.6 向け:

```
GRANT USAGE ON *.* TO 'encrypted_user'@'%' REQUIRE SSL;
```

RDS プロキシを使用する場合は、通常のクラスターエンドポイントではなく、プロキシエンドポイントに接続します。Aurora DB クラスターへの直接接続の場合と同様に、プロキシへの SSL/TLS 接続を必須またはオプションにすることができます。RDS プロキシの使用の詳細については、「Amazon RDS Proxy for Aurora の使用」を参照してください。

> **Note**
>
> MySQL での TLS 接続の詳細については、MySQL ドキュメントを参照してください。

新しい TLS 証明書を使用して Aurora MySQL DB クラスターに接続するようにアプリケーションを更新する

202323 年 1 月 13 日に、Amazon RDS は、Transport Layer Security（TLS）を使用して Aurora DB クラスターに接続するための新しい認証局（CA）証明書を公開しました。ここでは、新しい証明書を使用するためのアプリケーションの更新について説明します。

このトピックでは、クライアントアプリケーションが DB クラスターへの接続に TLS を使用するかどうかを判断する方法を説明します。該当する場合はさらに、それらのアプリケーションの接続に証明書の検証が必要かどうかを確認できます。

> **Note**
>
> 一部のアプリケーションは、サーバー上の証明書を正常に検証できる場合にのみ、Aurora MySQL DB クラスターに接続されるように設定されています。
> そのようなアプリケーションの場合は、新しい CA 証明書を含むように、クライアントアプリケーションの信頼ストアを更新する必要があります。

クライアントアプリケーションの信頼ストアで CA 証明書を更新した後、DB クラスターで証明書をローテーションできます。これらの手順を開発環境またはステージング環境でテストしてから、本番環境で実装することを強くお勧めします。

証明書のローテーションの詳細については、「SSL/TLS 証明書のローテーション」を参照してください。証明書のダウンロードの詳細については、SSL/TLS を使用した DB クラスターへの接続の暗号化 を参照してください。Aurora MySQL DB クラスターで TLS を使用する方法については、「Aurora MySQL DB クラスターでの TLS の使用」を参照してください。

トピック

- アプリケーションが TLS を使用して Aurora MySQL DB クラスターに接続しているかどうかの確認
- クライアントが接続するために証明書の検証が必要かどうかの確認
- アプリケーション信頼ストアの更新
- TLS 接続を確立するための Java コードの例

アプリケーションが TLS を使用して Aurora MySQL DB クラスターに接続しているかどうかの確認

Aurora MySQL バージョン 2 (MySQL 5.7 と互換性あり) を使用しており、パフォーマンススキーマが有効になっている場合は、次のクエリを実行すると、接続で TLS が使用されているかどうかを確認できます。Performance Schemaを有効にする方法については、MySQL ドキュメントの「Performance Schemaクイックスタート」を参照してください。

```
mysql> SELECT id, user, host, connection_type
       FROM performance_schema.threads pst
       INNER JOIN information_schema.processlist isp
       ON pst.processlist_id = isp.id;
```

この出力例では、webapp1 が TLS を使用しているため、ユーザー独自のセッション (admin) と、ログインしているアプリケーションの両方を確認できます。

```
+----+-----------------+------------------+-----------------+
| id | user            | host             | connection_type |
+----+-----------------+------------------+-----------------+
|  8 | admin           | 10.0.4.249:42590 | SSL/TLS         |
|  4 | event_scheduler | localhost        | NULL            |
| 10 | webapp1         | 159.28.1.1:42189 | SSL/TLS         |
+----+-----------------+------------------+-----------------+
3 rows in set (0.00 sec)
```

クライアントが接続するために証明書の検証が必要かどうかの確認

JDBC クライアントおよび MySQL クライアントを接続するために証明書の検証が必要かどうかを確認できます。

JDBC

MySQL Connector/J 8.0 を使用した次の例では、アプリケーションの JDBC 接続プロパティをチェックして、正常な接続に有効な証明書が必要かどうかを確認する 1 つの方法を示します。MySQL のすべての JDBC 接続オプションの詳細については、MySQL ドキュメントの「設定プロパティ」を参照してください。

MySQL Connector/J 8.0 を使用するとき、次の例のように、接続プロパティで sslMode が VERIFY_CA または VERIFY_IDENTITY に設定されている場合、TLS 接続にはサーバー CA 証明書に対する検証が必要です。

```
Properties properties = new Properties();
properties.setProperty("sslMode", "VERIFY_IDENTITY");
properties.put("user", DB_USER);
properties.put("password", DB_PASSWORD);
```

> **Note**
>
> MySQL Java Connector v5.1.38 以降または MySQL Java Connector v8.0.9 以降を使用してデータベースに接続する場合、データベースへの接続時に TLS を使用するようにアプリケーションを明示的に設定しなくても、これらのクライアントドライバーはデフォルトで TLS を使用します。また、TLS の使用時に部分的な証明書検証が実行され、データベースサーバー証明書の有効期限が切れている場合、接続に失敗します。

MySQL

MySQL クライアントの以下の例では、スクリプトの MySQL 接続をチェックして、正常な接続に有効な証明書が必要かどうかを確認する 2 つの方法を示します。MySQL クライアントで使用するすべての接続オプションの詳細については、MySQL ドキュメントの「Client-Side Configuration for Encrypted Connections」を参照してください。

MySQL 5.7 または MySQL 8.0 クライアントを使用しているとき、次の例のように、`--ssl-mode` オプションに `VERIFY_CA` または `VERIFY_IDENTITY` を指定した場合、TLS 接続にはサーバー CA 証明書に対する検証が必要です。

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem
 --ssl-mode=VERIFY_CA
```

MySQL 5.6 クライアントを使用するとき、次の例のように、`--ssl-verify-server-cert` オプションを指定する場合、SSL 接続にはサーバー CA 証明書に対する検証が必要です。

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem
 --ssl-verify-server-cert
```

アプリケーション信頼ストアの更新

MySQL アプリケーションの信頼ストアの更新については、MySQL ドキュメントの「Installing SSL Certificates」を参照してください。

> **Note**
>
> 信頼ストアを更新するとき、新しい証明書を追加できるだけでなく、古い証明書を保持できます。

JDBC のアプリケーション信頼ストアの更新

TLS 接続に JDBC を使用するアプリケーションの信頼ストアを更新できます。

ルート証明書のダウンロードについては、SSL/TLS を使用した DB クラスターへの接続の暗号化 を参照してください。

証明書をインポートするサンプルスクリプトについては、証明書を信頼ストアにインポートするためのサンプルスクリプト を参照してください。

アプリケーションで mysql JDBC ドライバーを使用している場合は、アプリケーションで以下のプロパティを設定します。

```
System.setProperty("javax.net.ssl.trustStore", certs);
System.setProperty("javax.net.ssl.trustStorePassword", "password");
```

> **Note**
>
> セキュリティ上のベストプラクティスとして、ここに示されているプロンプト以外のパスワードを指定してください。

アプリケーションを起動するとき、以下のプロパティを設定します。

```
java -Djavax.net.ssl.trustStore=/path_to_truststore/MyTruststore.jks -
Djavax.net.ssl.trustStorePassword=my_truststore_password com.companyName.MyApplication
```

TLS 接続を確立するための Java コードの例

次のコード例では、JDBC を使用してサーバー証明書を検証する SSL 接続のセットアップ方法を示します。

```
public class MySQLSSLTest {
```

```
        private static final String DB_USER = "user name";
        private static final String DB_PASSWORD = "password";
        // This key store has only the prod root ca.
        private static final String KEY_STORE_FILE_PATH = "file-path-to-keystore";
        private static final String KEY_STORE_PASS = "keystore-password";

    public static void test(String[] args) throws Exception {
        Class.forName("com.mysql.jdbc.Driver");


        System.setProperty("javax.net.ssl.trustStore", KEY_STORE_FILE_PATH);
        System.setProperty("javax.net.ssl.trustStorePassword", KEY_STORE_PASS);

        Properties properties = new Properties();
        properties.setProperty("sslMode", "VERIFY_IDENTITY");
        properties.put("user", DB_USER);
        properties.put("password", DB_PASSWORD);

        Connection connection = DriverManager.getConnection("jdbc:mysql://jagdeeps-ssl-
test.cni62e2e7kwh.us-east-1.rds.amazonaws.com:3306",properties);
        Statement stmt=connection.createStatement();

        ResultSet rs=stmt.executeQuery("SELECT 1 from dual");

        return;
    }
}
```

Important

データベース接続で TLS を使用することを決定し、アプリケーションの信頼ストアを更新したら、rds-ca-rsa2048-g1 証明書を使用するようにデータベースを更新できます。ステップについては、「DB インスタンスの変更による CA 証明書の更新」のステップ 3 を参照してください。

Aurora MySQL での Kerberos 認証の使用

Aurora MySQL DB クラスターに接続するユーザーを Kerberos 認証を使用して認証できるようになりました。そのためには、Kerberos 認証に AWS Directory Service for Microsoft Active Directory を使用するように DB クラスターを設定します。AWS Directory Service for Microsoft Active Directory は AWS Managed Microsoft AD とも呼ばれます。これは、AWS Directory Service で利用できる機能です。詳細については、「AWS Directory Service 管理ガイド」の「AWS Directory Service とは」を参照してください。

まず、ユーザー認証情報を格納する AWS Managed Microsoft AD ディレクトリを作成します。次に、Active Directory のドメインおよびその他の情報を Aurora MySQL DB クラスターに提供します。ユーザーが Aurora MySQL クラスターを使用して認証を実行すると、認証要求は AWS Managed Microsoft AD ディレクトリに転送されます。

同じディレクトリにすべての認証情報を保持することで時間と労力を節約できます。この方法により、複数の DB クラスターの認証情報を一元的に保存および管理できます。また、ディレクトリを使用することで、セキュリティプロファイル全体を向上できます。

また、独自のオンプレミスの Microsoft Active Directory から認証情報にアクセスできます。そのためには、信頼するドメイン関係を作成して、AWS Managed Microsoft AD ディレクトリがオンプレミスの Microsoft Active Directory を信頼するようにします。これにより、ユーザーは、オンプレミスネットワークのワークロードにアクセスするときと同じ Windows シングルサインオン (SSO) の使い方で、Aurora MySQL クラスターにアクセスできます。

データベースは Kerberos、AWS Identity and Access Management (IAM)、または Kerberos 認証と IAM 認証の両方。ただし、Kerberos 認証と IAM 認証では異なる認証方法が提供されるため、特定のユーザーは、どちらか一方の認証方法のみを使用してデータベースにログインできますが、両方を使用することはできません。IAM 認証の詳細については、「 の IAM データベース認証」を参照してください。

目次

Aurora MySQL DB クラスターの Kerberos 認証の概要

Aurora MySQL DB クラスターに Kerberos 認証を設定するには、以下の一般的なステップを完了します。これらのステップについては、後で詳細を説明します。

1. AWS Managed Microsoft AD を使用して AWS Managed Microsoft AD ディレクトリを作成します。AWS Management Console、AWS CLI、AWS Directory Service を使用して、ディレクトリを作成できます。詳しい手順については、AWS Directory Service 管理ガイドの「AWS Managed Microsoft AD ディレクトリの作成」を参照してください。
2. マネージド IAM ポリシー `AmazonRDSDirectoryServiceAccess` を使用する AWS Identity and Access Management (IAM) ロールの作成 このロールにより Amazon Aurora はディレクトリを呼び出すことができます。

   ロールによるアクセスを許可するには、AWS Security Token Service (AWS STS) エンドポイントを AWS アカウントの AWS リージョン でアクティベートする必要があります。AWS STS エンドポイントは、すべての AWS リージョン でデフォルトでアクティブになっているため、他のアクションを実行せずに、エンドポイントを使用することができます。詳細については、IAM ユーザーガイドの「AWS リージョン でのアクティブ化と非アクティブ化」を参照してください。

3. Microsoft Active Directory のツールを使用して、AWS Managed Microsoft AD ディレクトリでユーザーとグループを作成し、設定します。Active Directory にユーザーを作成する方法の詳細については、AWS 管理ガイドの「AWS Directory Service マネージド Microsoft AD でユーザーとグループを管理する」を参照してください。
4. Aurora MySQL DB クラスターを作成または変更する 作成リクエストで CLI または RDS API を使用する場合は、`Domain` パラメータでドメイン識別子を指定します。ディレクトリの作成時に生成された `d-*` 識別子と、作成した IAM ロールの名前を使用します。

   既存の Aurora MySQL DB クラスターを変更して Kerberos 認証を使用する場合は、DB クラスターのドメインパラメータと IAM ロールパラメータを設定します。ドメインディレクトリと同じ VPC で DB クラスターを見つけます。
5. Amazon RDS プライマリユーザー認証情報を使用して、Aurora MySQL DB クラスターに接続します。ステップ 6: Kerberos 認証を使用する Aurora MySQL ユーザーを作成する の手順に従って、Aurora MySQL でデータベースユーザーを作成します。

   この方法で作成したユーザーは、Kerberos 認証を使用して Aurora MySQL DB クラスターにログインできます。詳細については、「Kerberos 認証を使用した Aurora MySQL への接続」を参照してください。

オンプレミスまたはセルフホスト型の Microsoft Active Directory を使用して Kerberos 認証を取得するには、フォレストの信頼関係を確立する必要があります。フォレストの信頼関係とは、2 つのドメイングループ間の信頼関係です。信頼は、一方向または双方向にすることができます。AWS Directory Service を使用してフォレストの信頼関係を設定する方法の詳細については、AWS Directory Service 管理ガイドの「信頼関係を作成する場合」を参照してください。

Aurora MySQL の Kerberos 認証の制限事項

Aurora MySQL の Kerberos 認証には、以下の制限が適用されます。

- Kerberos 認証は、Aurora MySQL バージョン 3.03 以降でサポートされています。

  AWS リージョン でのサポートの詳細については、「Aurora MySQL で Kerberos 認証を使用する」を参照してください。
- Aurora MySQL で Kerberos 認証を使用するには、MySQL クライアントまたはコネクタが Unix プラットフォームではバージョン 8.0.26 以上、Windows では 8.0.27 以降を使用する必要があります。それ以外の場合は、クライアント側の `authentication_kerberos_client` プラグインが利用できず、認証できません。

- AWS Managed Microsoft AD は、Aurora MySQL でのみサポートされています。ただし、同じ AWS リージョン の異なるアカウントによって所有されている共有の Managed Microsoft AD ドメインに、Aurora MySQL DB クラスターを接続できます。

  また、独自のオンプレミスの Active Directory を使用できます。詳細については、「ステップ 2: (オプション) オンプレミスの Active Directory の信頼を作成する」を参照してください。
- MySQL クライアントから、または Windows オペレーティングシステムのドライバーから Aurora MySQL クラスターに接続するユーザーを Kerberos を使用して 認証する場合、デフォルトでは、データベースユーザー名の大文字と小文字は Active Directory のユーザーのものと一致する必要があります。例えば、Active Directory のユーザーが `Admin` として表示されている場合、データベースユーザー名は `Admin` である必要があります。

  ただし、`authentication_kerberos` プラグインでは、大文字と小文字を区別しないユーザー名の比較が可能になりました。詳細については、「ステップ 8: (オプション) 大文字と小文字を区別しないユーザー名比較の設定」を参照してください。
- `authentication_kerberos` プラグインをインストールする機能を有効にした後は、リーダー DB インスタンスを再起動する必要があります。
- `authentication_kerberos` プラグインをサポートしていない DB インスタンスに複製すると、複製が失敗する可能性があります。
- Aurora グローバルデータベースで Kerberos 認証を使用するには、グローバルデータベース内のすべての DB クラスターに認証を設定する必要があります。
- ドメイン名は 62 文字未満にする必要があります。
- Kerberos 認証を有効にした後は、DB クラスターポートを変更しないでください。ポートを変更すると、Kerberos 認証が機能しなくなります。

Aurora MySQL DB クラスターの Kerberos 認証の設定

AWS Managed Microsoft AD を使用して、Aurora MySQL DB クラスターに Kerberos 認証を設定します。Kerberos 認証をセットアップするには、次のステップに従います。

トピック

- ステップ 1: AWS Managed Microsoft AD を使用してディレクトリを作成する
- ステップ 2: (オプション) オンプレミスの Active Directory の信頼を作成する
- ステップ 3: Amazon Aurora 用の IAM ロールを作成する
- ステップ 4: ユーザーを作成して設定する

- ステップ 5: Aurora MySQL DB クラスターを作成または変更する
- ステップ 6: Kerberos 認証を使用する Aurora MySQL ユーザーを作成する
- ステップ 7: MySQL クライアントを設定する
- ステップ 8: (オプション) 大文字と小文字を区別しないユーザー名比較の設定

ステップ 1: AWS Managed Microsoft AD を使用してディレクトリを作成する

AWS Directory Service はフルマネージド型の Active Directory を AWS クラウド内に作成します。AWS Managed Microsoft AD ディレクトリを作成すると、AWS Directory Service がユーザーに代わって 2 つのドメインコントローラーと 2 つのドメインネームシステム (DNS) サーバーを作成します。ディレクトリサーバーは、VPC 内の異なるサブネットで作成されます。この冗長性によって、障害が発生してもディレクトリにアクセス可能な状態を維持できます。

AWS Managed Microsoft AD ディレクトリを作成すると、AWS Directory Service がユーザーに代わって自動的に以下のタスクを実行します。

- VPC 内に Active Directory を設定します。
- ユーザー名 `Admin` と指定されたパスワードで、ディレクトリ管理者アカウントを作成します。このアカウントはディレクトリの管理に使用します。

> **Note**
>
> このパスワードは必ず保存してください。AWS Directory Service は保存しません。パスワードはリセットできますが、取得することはできません。

- ディレクトリコントローラー用セキュリティグループを作成します。

AWS Managed Microsoft AD を起動すると、AWS は組織単位 (OU) を作成します。OU にはディレクトリのオブジェクトがすべて含まれています。この OU は、ディレクトリの作成時に入力した NetBIOS 名です。AWS が所有および管理するドメインルートにあります。

`Admin` ディレクトリに作成された AWS Managed Microsoft AD アカウントには、OU に対して頻繁に実行される、次のような管理行為の権限が含まれています。

- ユーザーを作成、更新、削除する
- ファイルやプリントサーバーなどのドメインにリソースを追加して、追加したリソースへのアクセス許可を OU のユーザーとグループに割り当てる

- 追加の OU やコンテナを作成する
- 権限を委譲する
- 削除されたオブジェクトを Active Directory のごみ箱から元に戻す
- Active Directory Web Service で AD と DNS Windows PowerShell モジュールを実行する

Admin アカウントには、ドメイン全体に関係するアクティビティを実行する権限もあります。

- DNS 設定 (レコード、ゾーン、フォワーダーの追加、削除、更新) を管理する
- DNS イベントログを参照する
- セキュリティイベントログを参照する

AWS Managed Microsoft AD でディレクトリを作成するには

1. AWS Management Console にサインインし、AWS Directory Service コンソール (https://console.aws.amazon.com/directoryservicev2/)を開きます。
2. ナビゲーションペインで、[Directories]、[Set up Directory] の順に選択します。
3. [AWS Managed Microsoft AD] を選択します。現状では、AWS Managed Microsoft AD が Amazon RDS で使用できる唯一のオプションです。
4. 次の情報を入力します。

   ディレクトリの DNS 名

   ディレクトリの完全修飾名 (例: **corp.example.com**)。

   [Directory NetBIOS name] (ディレクトリの NetBIOS 名)

   ディレクトリの短縮名 (例: **CORP**)。

   ディレクトリの説明

   (オプション) ディレクトリの説明。

   Admin パスワード

   ディレクトリ管理者のパスワードです。ディレクトリの作成プロセスでは、ユーザー名 Admin とこのパスワードで管理者アカウントが作成されます。

ディレクトリ管理者のパスワードに「admin」の単語を含めることはできません。パスワードは大文字と小文字を区別し、8-64 文字にします。また、以下の 4 つのカテゴリうち 3 つから少なくとも 1 文字を含める必要があります。

- 小文字 (a ~ z)
- 大文字 (A ~ Z)
- 数字 (0 ~ 9)
- アルファベット以外の文字 (~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/)

[Confirm password] (パスワードを確認)

管理者パスワードが再入力されました。

5. [次へ] をクリックします。
6. [Networking] セクションに次の情報を入力し、[Next] を選択します。

   VPC

   ディレクトリ用の VPC。この同じ VPC に Aurora MySQL DB クラスターを作成します。

   サブネット

   ディレクトリサーバーのサブネット。2 つのサブネットは、異なるアベイラビリティーゾーンに存在している必要があります。
7. ディレクトリ情報を確認し、必要な変更を加えます。情報が正しい場合は、[Create directory (ディレクトリの作成)] を選択します。

ディレクトリが作成されるまで、数分かかります。正常に作成されると、[Status] 値が [Active] に変わります。

ディレクトリに関する情報を表示するには、ディレクトリの一覧で、ディレクトリ名を選択します。[Directory ID] の値を書き留めます。この値は、Aurora MySQL DB クラスターを作成または変更するときに必要になります。

ステップ 2: (オプション) オンプレミスの Active Directory の信頼を作成する

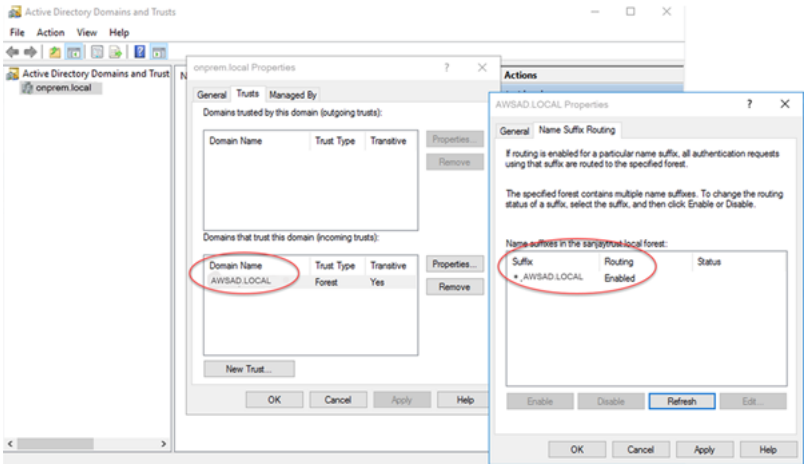
独自のオンプレミスの Microsoft Active Directory を使用する予定がない場合は、ステップ 3: Amazon Aurora 用の IAM ロールを作成する に進みます。

オンプレミスの Active Directory を使用して Kerberos 認証を取得するには、オンプレミスの Microsoft Active Directory と (AWS Managed Microsoft AD で作成された) ステップ 1: AWS Managed

Microsoft AD を使用してディレクトリを作成する ディレクトリとの間に、フォレストの信頼を使用して、信頼するドメイン関係を作成する必要があります。信頼は一方向にすることができます。この場合、AWS Managed Microsoft AD ディレクトリはオンプレミスの Microsoft Active Directory を信頼します。信頼は、両方の Active Directory が相互に信頼する双方向にすることもできます。AWS Directory Service を使用して信頼関係を設定する方法の詳細については、AWS Directory Service 管理ガイドの「信頼関係を作成する場合」を参照してください。

> **Note**
>
> オンプレミスの Microsoft Active Directory を使用している場合:
>
> - Windows クライアントは、rds.amazonaws.com ではなく、エンドポイントの AWS Directory Service のドメイン名を使用して接続する必要があります。詳細については、「Kerberos 認証を使用した Aurora MySQL への接続」を参照してください。
> - Windows クライアントは、Aurora カスタムエンドポイントを使用すると接続できません。詳細については、「Amazon Aurora 接続管理」を参照してください。
> - グローバルデータベースの場合:
>   - Windows クライアントは、グローバルデータベースのプライマリ AWS リージョン リージョンにあるインスタンスエンドポイントまたはクラスターエンドポイントを使用する場合に限り接続できます。
>   - Windows クライアントは、セカンダリ AWS リージョン のクラスターエンドポイントを使用して接続できません。

オンプレミスの Microsoft Active Directory ドメイン名に、新しく作成された信頼関係に対応する DNS サフィックスルーティングが含まれていることを確認してください。次のスクリーンショットは、例を示しています。



ステップ 3: Amazon Aurora 用の IAM ロールを作成する

Amazon Aurora が AWS Directory Service を呼び出すには、マネージド IAM ポリシー `AmazonRDSDirectoryServiceAccess` を使用する AWS Identity and Access Management (IAM) ロールが必要です。このロールにより、Aurora は AWS Directory Service を呼び出すことができます。

AWS Management Console を使用して DB クラスターを作成し、`iam:CreateRole` アクセス許可を持っている場合、コンソールではこのロールを自動的に作成します。この場合、ロール名は `rds-directoryservice-kerberos-access-role` です。それ以外の場合は、IAM ロールを手動で作成する必要があります。IAM ロールを作成する場合、[Directory Service] を選択し、それに AWS マネージドポリシー `AmazonRDSDirectoryServiceAccess` をアタッチします。

サービス用の IAM ロールを作成する方法の詳細については、IAM ユーザーガイドの「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。

マネージド IAM ポリシー `AmazonRDSDirectoryServiceAccess` を使用する代わりに、必要なアクセス許可を使用してポリシーを作成することもできます。これを行うには、IAM ロールに次の IAM 信頼ポリシーが必要です。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.amazonaws.com",
          "rds.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

また、ロールには、以下の IAM ロールポリシーも必要です。

```
{
  "Version": "2012-10-17",
```

```
  "Statement": [
    {
      "Action": [
        "ds:DescribeDirectories",
        "ds:AuthorizeApplication",
        "ds:UnauthorizeApplication",
        "ds:GetAuthorizedApplicationDetails"
      ],
    "Effect": "Allow",
    "Resource": "*"
    }
  ]
}
```

ステップ 4: ユーザーを作成して設定する

Active Directory ユーザーとコンピュータツールを使用してユーザーを作成できます。このツールは、Active Directory Domain Services ツールおよび Active Directory Lightweight Directory Services ツールの一部です。ユーザーとは、ディレクトリにアクセスできる個別の人物、またはエンティティのことです。

AWS Directory Service ディレクトリにユーザーを作成するには、AWS Directory Service ディレクトリに接続している Microsoft Windows ベースのオンプレミスまたは Amazon EC2 インスタンスを使用します。ユーザーを作成する権限を持つユーザーとして、インスタンスにログインする必要があります。詳細については、AWS Managed Microsoft AD Directory Service 管理ガイドのAWS のユーザーとグループの管理を参照してください。

ステップ 5: Aurora MySQL DB クラスターを作成または変更する

ディレクトリ用の Aurora MySQL DB クラスターを作成または変更します。コンソール、AWS CLI、RDS API を使用して DB クラスターとディレクトリを関連付けることができます。このタスクを実行するには、以下の 2 つの方法があります。

- コンソール、create-db-cluster CLI コマンド、または CreateDBCluster RDS API オペレーションを使用して、新しい Aurora MySQL DB クラスターを作成します。

  手順については、「Amazon Aurora DB クラスターの作成」を参照してください。
- コンソール、modify-db-cluster CLI コマンド、または ModifyDBCluster RDS API オペレーションを使用して、既存の Aurora MySQL DB クラスターを変更します。

  手順については、「Amazon Aurora DB クラスターの変更」を参照してください。

- コンソール、restore-db-cluster-from-snapshot CLI コマンド、または RestoreDBClusterFromSnapshot RDS API オペレーションを使用して、DB スナップショットから Aurora MySQL DB クラスターを復元します。

  手順については、「DB クラスターのスナップショットからの復元」を参照してください。

- コンソール、restore-db-cluster-to-point-in-time CLI コマンド、RestoreDBClusterToPointInTime RDS API オペレーションを使用して、Aurora MySQL DB クラスターをポイントインタイムに復元します。

  手順については、「DB クラスターを指定の時点の状態に復元する」を参照してください。

Kerberos 認証は、VPC 内の Aurora MySQL DB クラスターでのみサポートされています。DB クラスターは、ディレクトリと同じ VPC または異なる VPC 内にあります。DB クラスターの VPC には、ディレクトリへのアウトバウンド通信を許可する VPC セキュリティグループが必要です。

コンソール

DB クラスターを作成、変更または復元するためにコンソールを使用する場合は、[データベースの認証] セクションの [Kerberos 認証] を選択します。[ディレクトリの参照] を選択してディレクトリを選択するか、[新しいディレクトリの作成] を選択します。

AWS CLI

コンソール、AWS CLI、RDS API を使用する場合は、DB クラスターとディレクトリを関連付けます。作成したドメインディレクトリを使用するには、DB クラスターに以下のパラメータが必要です。

- `--domain` パラメータには、ディレクトリの作成時に生成されたドメイン識別子 ("d-*" 識別子) を使用します。
- `--domain-iam-role-name` パラメータには、マネージド IAM ポリシー `AmazonRDSDirectoryServiceAccess` を使用する作成済みのロールを使用します。

例えば、以下の CLI コマンドはディレクトリを使用するように DB クラスターを変更します。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
```

```
    --domain d-ID \
    --domain-iam-role-name role-name
```

Windows の場合:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --domain d-ID ^
    --domain-iam-role-name role-name
```

> **Important**
>
> DB クラスターを変更して Kerberos 認証を有効にした場合、変更後、その DB クラスターを再起動します。

ステップ 6: Kerberos 認証を使用する Aurora MySQL ユーザーを作成する

DB クラスターは、AWS Managed Microsoft AD ドメインに接続されています。したがって、このドメインの Active Directory ユーザーから Aurora MySQL ユーザーを作成できます。データベースへのアクセス許可は、これらのユーザーに対して付与および取り消される標準の Aurora MySQL アクセス許可を通じて管理されます。

Active Directory ユーザーに対して Aurora MySQL での認証を許可できます。これを行うには、まず Amazon RDS プライマリユーザーの認証情報を使用して、他の DB クラスターと同様に Aurora MySQL DB クラスターに接続します。ログインしたら、次に示すように、Aurora MySQL で Kerberos 認証された外部認証ユーザーを作成します。

```
CREATE USER user_name@'host_name' IDENTIFIED WITH 'authentication_kerberos' BY
 'realm_name';
```

- *user_name* をユーザー名で置き換えます。ドメインのユーザー (人とアプリケーションの両方) は、Kerberos 認証を使用して、クライアントマシンを結合したドメインから DB クラスターに接続できるようになりました。
- *host_name* をホスト名に置き換えます。ワイルドカードとして % を使用できます。ホスト名には、特定の IP アドレスを使用することもできます。

- *realm_name* をドメインディレクトリの領域名に置き換えます。領域名は、通常 CORP.EXAMPLE.COM のように大文字の DNS ドメイン名と同じです。領域とは、同じ Kerberos Key Distribution Center を使用するシステムのグループです。

次の例では、領域名 MYSQL.LOCAL の Active Directory に対して認証する名前 Admin のデータベースユーザーを作成します。

```
CREATE USER Admin@'%' IDENTIFIED WITH 'authentication_kerberos' BY 'MYSQL.LOCAL';
```

既存の Aurora MySQL ログインの変更

また、次の構文を使用して、既存の Aurora MySQL ログインを Kerberos 認証を使用するように変更できます。

```
ALTER USER user_name IDENTIFIED WITH 'authentication_kerberos' BY 'realm_name';
```

ステップ 7: MySQL クライアントを設定する

MySQL クライアントを設定するには、次のステップを実行します。

1. ドメインを指す krb5.conf ファイル (または同等) を作成します。
2. クライアントホストと AWS Directory Service 間でトラフィックが流れることを確認します。次の目的で Netcat などのネットワークユーティリティを使用します。
   - ポート 53 の DNS 経由のトラフィックを確認します。
   - ポート 53 および Kerberos の TCP/UDP 上のトラフィックを確認します。これには、AWS Directory Service の場合ポート 88 および 464 が含まれます。
3. データベースポートを介してクライアントホストと DB インスタンス間でトラフィックが流れることを確認します。例えば、mysql を使用してデータベースに接続し、アクセスします。

以下は、AWS Managed Microsoft AD の krb5.conf の内容のサンプルです。

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
```

```
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
```

以下は、オンプレミスの Microsoft Active Directory 向けの `krb5.conf` の内容のサンプルです。

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
 ONPREM.COM = {
  kdc = onprem.com
  admin_server = onprem.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
 .onprem.com = ONPREM.COM
 onprem.com = ONPREM.COM
 .rds.amazonaws.com = EXAMPLE.COM
 .amazonaws.com.cn = EXAMPLE.COM
 .amazon.com = EXAMPLE.COM
```

ステップ 8: (オプション) 大文字と小文字を区別しないユーザー名比較の設定

デフォルトでは、MySQL データベースユーザー名の大文字と小文字は、Active Directory ログインのものと一致する必要があります。ただし、`authentication_kerberos` プラグインでは、大文字と小文字を区別しないユーザー名の比較が可能になりました。そのためには、`authentication_kerberos_caseins_cmp` DB クラスターパラメーターを `true` に設定します。

大文字と小文字を区別しないユーザー名比較を使用するには

1. カスタム DB クラスターのパラメータグループを作成します。「DB クラスターのパラメータグループの作成」の手順に従います。
2. 新しいパラメータグループを編集して、`authentication_kerberos_caseins_cmp` の値を `true` に設定します。「DB クラスターパラメータグループのパラメータの変更」の手順に従います。

3. DB クラスターパラメータグループを Aurora MySQL DB クラスターに関連付けます。「DB クラスターパラメータグループと DB クラスターの関連付け」の手順に従います。
4. DB クラスターを再起動します。

Kerberos 認証を使用した Aurora MySQL への接続

エラーを回避するため、Unix プラットフォームではバージョン 8.0.26 以降、Windows では 8.0.27 以降の MySQL クライアントを使用してください。

Aurora MySQL Kerberos ログインを使用して DB クラスターに接続する

Kerberos 認証で Aurora MySQL に接続するには、ステップ 6: Kerberos 認証を使用する Aurora MySQL ユーザーを作成する の手順に従って作成したデータベースユーザーとしてログインします。

コマンドプロンプトで、Aurora MySQL DB クラスターに関連付けられているエンドポイントの 1 つに接続します。パスワードの入力を求められたら、そのユーザー名に関連付けられている Kerberos パスワードを入力します。

Kerberos で認証すると、ticket-granting ticket (TGT) がまだ存在しない場合は、TGT が生成されます。`authentication_kerberos` プラグインによって TGT を使用してサービスチケットを取得し、そのチケットが Aurora MySQL データベースサーバーに提示されます。

MySQL クライアントを使用すると、Windows または Unix を使用して Kerberos 認証によって Aurora MySQL に接続できます。

Unix

接続するには、次のいずれかの方法を使用します。

- TGT を手動で入手します。この場合、MySQL クライアントにパスワードを指定する必要はありません。
- Active Directory ログイン用のパスワードを MySQL クライアントに直接指定します。

クライアント側のプラグインは、Unix プラットフォームでは MySQL クライアントバージョン 8.0.26 以降でサポートされています。

TGT を手動で取得して接続するには

1. コマンドラインインターフェイスで、次のコマンドを使用して TGT を取得します。

```
kinit user_name
```

2. 次の mysql コマンドを使用して、DB クラスターの DB インスタンスエンドポイントにログインします。

```
mysql -h DB_instance_endpoint -P 3306 -u user_name -p
```

> **Note**
>
> DB インスタンスでキータブがローテーションされると、認証が失敗する可能性があります。この場合は、kinit を再実行して新しい TGT を取得します。

直接接続するには

1. コマンドラインインターフェイスで、次の mysql コマンドを使用して、DB クラスターの DB インスタンスエンドポイントにログインします。

```
mysql -h DB_instance_endpoint -P 3306 -u user_name -p
```

2. Active Directory ユーザーのパスワードを入力します。

Windows

Windows では、通常、認証はログイン時に行われるため、Aurora MySQL DB クラスターに接続するために TGT を手動で取得する必要はありません。データベースユーザー名の大文字と小文字は、Active Directory のユーザーのものと一致する必要があります。例えば、Active Directory のユーザーが Admin として表示されている場合、データベースユーザー名は Admin である必要があります。

クライアント側のプラグインは、Windows では MySQL クライアントバージョン 8.0.27 以降でサポートされています。

直接接続するには

- コマンドラインインターフェイスで、次の mysql コマンドを使用して、DB クラスターの DB インスタンスエンドポイントにログインします。

```
mysql -h DB_instance_endpoint -P 3306 -u user_name
```

Aurora グローバルデータベースで Kerberos 認証を使用する

Aurora MySQL の Kerberos 認証は、Aurora グローバルデータベースでサポートされています。プライマリ DB クラスターの Active Directory を使用してセカンダリ DB クラスターのユーザーを認証するには、Active Directory をセカンダリ AWS リージョン に複製します。プライマリクラスターと同じドメイン ID を使用して、セカンダリクラスターの Kerberos 認証を有効にします。AWS Managed Microsoft AD レプリケーションは、エンタープライズバージョンの Active Directory でのみサポートされます。詳細については、AWS Directory Service 管理ガイドの「マルチリージョンレプリケーション」を参照してください。

RDS for MySQL から Aurora MySQL への移行

Kerberos 認証を有効にした RDS for MySQL から Aurora MySQL への移行後、`auth_pam` プラグインで作成されたユーザーを `authentication_kerberos` プラグインを使用するように変更します。次に例を示します。

```
ALTER USER user_name IDENTIFIED WITH 'authentication_kerberos' BY 'realm_name';
```

チケットキャッシュの防止

MySQL クライアントアプリケーションの起動時に有効な TGT が存在しない場合、アプリケーションは TGT を取得してキャッシュできます。TGT がキャッシュされないようにするには、`/etc/krb5.conf` ファイルに設定パラメータを設定します。

> **Note**
>
> この設定は UNIX を実行しているクライアントホストにのみ適用され、Windows には適用されません。

TGT キャッシュを防止するには

- 次のように、`/etc/krb5.conf` に `[appdefaults]` セクションを追加します。

```
[appdefaults]
```

```
mysql = {
  destroy_tickets = true
}
```

Kerberos 認証用のログ記録

AUTHENTICATION_KERBEROS_CLIENT_LOG 環境変数によって、Kerberos 認証のログ記録レベルを設定します。ログはクライアント側のデバッグに使用できます。

指定できる値は 1~5 です。ログメッセージは、標準エラー出力に書き込まれます。次の表に各ログ記録レベルの説明を示します。

ログ記録レベル	説明
1 または未設定	ログ記録なし
2	エラーメッセージ
3	エラーと警告メッセージ
4	エラー、警告、情報メッセージ
5	エラー、警告、情報メッセージ、デバッグメッセージ

ドメイン内の DB クラスターの管理

AWS CLI または RDS API を使用して、DB クラスターとマネージド Active Directory との関係を管理できます。例えば、Kerberos 認証用に Active Directory を関連付けたり、Active Directory の関連付けを解除して Kerberos 認証を有効にしたりできます。さらに、DB クラスターを外部認証する Active Directory を別の Active Directory に変更することもできます。

例えば、Amazon RDS API を使用して次を実行できます。

- メンバーシップが失敗した Kerberos 認証を再度有効化するには、ModifyDBInstance API オペレーションを使用し、現在のメンバーシップのディレクトリ ID を指定します。
- メンバーシップの IAM ロール名を更新するには、ModifyDBInstance API オペレーションを使用し、現在のメンバーシップのディレクトリ ID と新しい IAM ロールを指定します。

- DB クラスターの Kerberos 認証を無効にするには、ModifyDBInstance API オペレーションを使用し、ドメインパラメータとして none を指定します。
- ドメイン間で DB クラスターを移動するには、ModifyDBInstance API オペレーションを使用し、新しいドメインのドメイン識別子をドメインパラメータとして指定します。
- 各 DB クラスターのメンバーシップを一覧表示するには、DescribeDBInstances API オペレーションを使用します。

ドメインのメンバーシップを理解する

DB クラスターを作成または変更すると、そのインスタンスはドメインのメンバーになります。DB クラスターのドメインメンバーシップのステータスを表示するには、describe-db-clusters CLI コマンドを実行します。DB クラスターのステータスは、次のいずれかです。

- kerberos-enabled — DB クラスターでは Kerberos 認証が有効になっています。
- enabling-kerberos — AWS は、この DB クラスターで Kerberos 認証を有効化中です。
- pending-enable-kerberos — この DB クラスターでは、Kerberos 認証の有効化が保留になっています。
- pending-maintenance-enable-kerberos - AWS は、次に予定されているメンテナンス期間で、DB クラスターの Kerberos 認証の有効化を試みます。
- pending-disable-kerberos — この DB クラスターでは、Kerberos 認証の無効化が保留になっています。
- pending-maintenance-disable-kerberos - AWS は、次に予定されているメンテナンス期間で、DB クラスターの Kerberos 認証の無効化を試みます。
- enable-kerberos-failed - 設定の問題により、AWS は DB クラスター上の Kerberos 認証を有効化できませんでした。DB クラスターの変更コマンドを再発行する前に、設定を確認して修正してください。
- disabling-kerberos — AWS は、この DB クラスターで Kerberos 認証を無効化中です。

ネットワーク接続の問題や正しくない IAM ロールのために、Kerberos 認証を有効化するリクエストは失敗する可能性があります。例えば、DB クラスターを作成するか、既存の DB クラスターを変更し、Kerberos 認証を有効化しようとして失敗したとします。この場合は、変更コマンドを再発行するか、新しく作成した DB クラスターを変更してドメインに参加させます。

Amazon Aurora MySQL DB クラスターへのデータの移行

既存のデータベースから Amazon Aurora MySQL DB クラスターにデータを移行するために、複数のオプションがあります。また、移行オプションは、移行元のデータベースおよび移行するデータのサイズによっても異なります。

移行には物理と論理の 2 つのタイプがあります。物理移行とは、データベースファイルの物理コピーを使用してデータベースを移行することです。論理的な移行とは、挿入、更新、削除などの論理的な変更を適用することでデータベースを移行することです。

物理移行には以下の利点があります。

- 物理移行は、特にデータベースのサイズが大きい場合、論理的な移行よりも高速です。
- 物理移行用にバックアップを作成するとき、データベースのパフォーマンスが低下しません。
- 物理移行では、複雑なデータベースコンポーネントを含め、出典データベースのすべての内容を移行できます。

物理移行には以下の制限があります。

- innodb_page_size パラメータは、デフォルト値 (16KB) に設定する必要があります。
- innodb_data_file_path パラメータは、デフォルトのデータファイル名 "ibdata1:12M:autoextend" を使用するデータファイル 1 つのみで設定する必要があります。2 つのデータファイルを持つデータベースや名前が異なるデータファイルを持つデータベースは、この方法では移行できません。

  "innodb_data_file_path=ibdata1:50M; ibdata2:50M:autoextend" や "innodb_data_file_path=ibdata01:50M:autoextend" などのファイル名は使用できません。
- innodb_log_files_in_group パラメータは、デフォルト値 (2) に設定する必要があります。

論理的な移行には以下の利点があります。

- 特定のテーブルやテーブルの一部など、データベースのサブセットを移行できます。
- データは、物理ストレージ構造に関係なく移行できます。

論理的な移行には以下の制限があります。

- 論理的な移行は通常、物理移行よりも低速です。
- 複雑なデータベースコンポーネントにより、論理的な移行プロセスの速度が遅くなることがあります。場合によっては、複雑なデータベースコンポーネントにより、論理的な移行がブロックされることさえあります。

以下の表に示しているのは、移行のオプションと各オプションで使用する移行のタイプです。

移行元	移行タイプ	ソリューション
RDS for MySQL DB インスタンス	物理	まず MySQL DB インスタンスの Aurora MySQL リードレプリカを作成することで、RDS for MySQL DB インスタンスから移行できます。MySQL DB インスタンスと Aurora MySQL リードレプリカとの間のレプリカラグが 0 の場合は、クライアントアプリケーションで Aurora リードレプリカから読み取り、レプリケーションを停止することで、Aurora MySQL リードレプリカを読み取りと書き込み用のスタンドアロンの Aurora MySQL DB クラスターにすることができます。詳細については、「Aurora リードレプリカを使用した、RDS for MySQL DB インスタンスから Amazon Aurora MySQL DB クラスターへのデータの移行」を参照してください。
RDS for MySQL DB スナップショット	物理	RDS for MySQL DB スナップショットから Amazon Aurora MySQL DB クラスターに直接データを移行できます。詳細については、「Aurora への RDS for MySQL スナップショットの移行」を参照してください。
Amazon RDS 外部の MySQL データベース	論理	`mysqldump` ユーティリティを使用してデータのダンプを作成し、そのデータを既存の Amazon Aurora MySQL DB クラスターにインポートできます。詳細については、「mysqldump を使用した MySQL から Amazon Aurora MySQL

移行元	移行タイプ	ソリューション
		への論理的移行」を参照してください。 外部 MySQL データベースからの移行中にデータベースユーザーのメタデータをエクスポートするには、`mysqldump` の代わりに MySQL Shell コマンドを使用することもできます。詳細については、「インスタンスダンプユーティリティ、スキーマダンプユーティリティ、テーブルダンプユーティリティ」を参照してください。 **Note** MySQL 8.0.34 以降、mysqlpump ユーティリティは廃止されました。
Amazon RDS 外部の MySQL データベース	物理	データベースから Amazon Simple Storage Service (Amazon S3) バケットにバックアップファイルをコピーし、これらのファイルから Amazon Aurora MySQL DB クラスターを復元できます。このオプションは、`mysqldump` を使用したデータの移行よりもかなり高速になる場合があります。詳細については、「Percona XtraBackup と Amazon S3 を使用した MySQL からの物理的な移行」を参照してください。

移行元	移行タイプ	ソリューション
Amazon RDS 外部の MySQL データベース	論理	データベースのデータをテキストファイルとして保存し、そのファイルを Amazon S3 バケットにコピーできます。次に、`LOAD DATA FROM S3` MySQL コマンドを使用して、そのデータを既存の Aurora MySQL DB クラスター内にロードできます。詳細については、「Amazon S3 バケットのテキストファイルから Amazon Aurora MySQL DB クラスターへのデータのロード」を参照してください。
MySQL と互換性がないデータベース	論理	AWS Database Migration Service (AWS DMS) は、MySQL との互換性がないデータベースからのデータを移行するのに使用できます。AWS DMS の詳細については、「AWS Database Migration Service とは」を参照してください。

> **Note**
>
> 外部の MySQL データベースを Amazon RDS に移行する場合、表で説明している移行のオプションは、データベースが InnoDB または MyISAM テーブルスペースをサポートしている場合にのみサポートされます。
> Aurora MySQL に移行中の MySQL データベースで `memcached` が使用されている場合は、移行前に `memcached` を削除します。
> 8.0.11、8.0.13、8.0.15 などの一部の古い MySQL 8.0 バージョンからは Aurora MySQL バージョン 3.05 以上に移行できません。移行する前に MySQL バージョン 8.0.28 にアップグレードすることをお勧めします。

外部の MySQL データベースから Amazon Aurora MySQL DB クラスターへのデータ移行

データベースが InnoDB または MyISAM のテーブルスペースをサポートしている場合、これらのオプションを使用して、データを Amazon Aurora MySQL DB クラスターに移行できます。

- mysqldump ユーティリティを使用してデータのダンプを作成し、そのデータを既存の Amazon Aurora MySQL DB クラスターにインポートできます。詳細については、「mysqldump を使用した MySQL から Amazon Aurora MySQL への論理的移行」を参照してください。
- これで、データベースから Amazon S3 バケットに完全および増分バックアップファイルをコピーし、これらのファイルから Amazon Aurora MySQL DB クラスターを復元できます。このオプションは、mysqldump を使用したデータの移行よりもかなり高速になる場合があります。詳細については、「Percona XtraBackup と Amazon S3 を使用した MySQL からの物理的な移行」を参照してください。

トピック

- Percona XtraBackup と Amazon S3 を使用した MySQL からの物理的な移行
- mysqldump を使用した MySQL から Amazon Aurora MySQL への論理的移行

Percona XtraBackup と Amazon S3 を使用した MySQL からの物理的な移行

完全バックアップファイルと増分バックアップファイルをソース MySQL バージョン 5.7 または 8.0 データベースから AmazonAmazon S3 バケットにコピーします。その後、それらのファイルから同じメジャー DB エンジンバージョンの Amazon Aurora MySQL DB クラスターに復元できます。

このオプションは、mysqldump を使用したデータ移行よりもかなり高速になる場合があります。これは、mysqldump を使用することですべてのコマンドが再生され、新しい Aurora MySQL DB クラスターの出典データベースからスキーマとデータが再作成されるためです。出典 MySQL データファイルをコピーすることで、Aurora MySQL はこれらのファイルを即座に Aurora MySQL DB クラスター用のデータとして使用できます。

また、移行プロセス中にバイナリログレプリケーションを使用してダウンタイムを最小化できます。バイナリログのレプリケーションを使用すると、Aurora MySQL DB クラスターにデータ移動中に外部 MySQL データベースがオープンのままになります。Aurora MySQL DB クラスターが作成されたら、バイナリログレプリケーションを使用して、Aurora MySQL DB クラスターとバックアップ後のトランザクションを同期します。Aurora MySQL DB クラスターが MySQL データベースに追いついたら、Aurora MySQL DB を新規のトランザクションに完全に切り替えて、移行を終了します。詳

細については、「レプリケーションを使用して、Amazon Aurora MySQL DB クラスターと MySQL データベースを同期する」を参照してください。

目次

制約事項と考慮事項

Amazon S3 バケットから Amazon Aurora MySQL DB クラスターに移行する場合は、以下の制限と考慮事項が当てはまります。

- データは既存の DB クラスターではなく、新しい DB クラスターにのみ移行できます。
- Percona XtraBackup を使用してデータを S3 にバックアップする必要があります。詳細については、「Percona XtraBackup のインストール」を参照してください。

- Amazon S3 バケットと Aurora MySQL DB クラスターは同じ AWS リージョンに存在する必要があります。
- 以下から復元することはできません。
  - DB クラススナップショットは、Amazon S3 にスナップショットをエクスポートします。また、DB クラススナップショットエクスポートから、S3 にデータを移行することはできません。
  - 暗号化されたソースデータベース、しかし移行するデータを暗号化することはできます。移行プロセス中にデータを非暗号化のまま維持することもできます。
  - MySQL 5.5 または 5.6 データベース
- Percona Server for MySQL は、`mysql` スキーマに `compression_dictionary*` テーブルが含まれる場合があるため、ソースデータベースとしてはサポートされていません。
- Aurora Serverless DB クラスターに復元することはできません。
- 後方移行はメジャーバージョンとマイナーバージョンのどちらでもサポートされていません。例えば、MySQL バージョン 8.0 から Aurora MySQL バージョン 2 (MySQL 5.7 と互換性あり) への移行はできませんし、MySQL バージョン 8.0.32 から MySQL コミュニティバージョン 8.0.26 と互換性のある Aurora MySQL バージョン 3.03 への移行もできません。
- 8.0.11、8.0.13、8.0.15 などの一部の古い MySQL 8.0 バージョンからは Aurora MySQL バージョン 3.05 以上に移行できません。移行する前に MySQL バージョン 8.0.28 にアップグレードすることをお勧めします。
- Amazon S3 からのインポートは、db.t2.micro DB インスタンスクラスでサポートされていません。ただし、1 つの DB インスタンスクラスに復元し、後で別の DB インスタンスクラスを変更できます。DB インスタンスクラスの詳細については、「Aurora DB インスタンスクラス」を参照してください。
- Amazon S3 では、S3 バケットにアップロードするファイルのサイズが 5 TB に制限されます。バックアップファイルが 5 TB を超える場合は、バックアップファイルを小さいファイルに分割する必要があります。
- Amazon RDS では、S3 バケットにアップロードするファイルの数が百万までに制限されます。データベースのバックアップデータ (すべての完全および増分バックアップを含む) が百万ファイルを超える場合は、Gzip (.gz)、tar (.tar.gz)、Percona xbstream (.xbstream) ファイルを使用して完全および増分バックアップファイルを S3 バケットに保存します。Percona XtrabackUp 8.0 は Percona xbstream のみを圧縮用にサポートしています。
- 各 DB クラスターに管理サービスを提供するために、DB インスタンスの作成時に `rdsadmin` ユーザーが作成されます。これは RDS の予約ユーザーであるため、以下の制限が適用されます。

  - 'rdsadmin'@'localhost' が定義子である関数、プロシージャ、ビュー、イベント、トリガーは、インポートされません。詳細については、定義者として 'rdsadmin'@'localhost' を使用して保存されたオブジェクトおよびAmazon Aurora MySQL でのマスターユーザー特権を参照してください。
  - Aurora MySQL DB クラスターを作成すると、サポートされている最大の権限を持つマスターユーザーが作成されます。バックアップから復元する際、インポート中のユーザーに割り当てられたサポートされていない権限は、インポート中に自動的に削除されます。

    この影響を受ける可能性のあるユーザーを特定するには、「サポートされていない権限を持つユーザーアカウント」を参照してください。Aurora MySQL でサポートされる権限の詳細については、「ロールベースの特権モデル」を参照してください。
- Aurora MySQL バージョン 3 の場合、動的権限はインポートされません。Aurora がサポートする動的権限は、移行後にインポートできます。詳細については、「Aurora MySQL バージョン 3 に対する動的権限」を参照してください。
- mysql スキーマのユーザー作成のテーブルは移行されません。
- innodb_data_file_path パラメータは、デフォルトのデータファイル名 ibdata1:12M:autoextend を使用するデータファイル 1 つのみで設定する必要があります。2 つのデータファイルを持つデータベースや名前が異なるデータファイルを持つデータベースは、この方法では移行できません。

  innodb_data_file_path=ibdata1:50M、ibdata2:50M:autoextend、および innodb_data_file_path=ibdata01:50M:autoextend などのファイル名は使用できません。
- デフォルトの MySQL データディレクトリ外で定義されたテーブルを含むソースデータベースから移行することはできません。
- この方法を使用した非圧縮バックアップでサポートされる最大サイズは、現在 64 TiB に制限されています。圧縮バックアップの場合、圧縮解除に必要な容量を考慮してこの制限は小さくなります。このような場合、サポートされる最大バックアップサイズは (64 TiB - compressed backup size) です。
- Aurora MySQL は MySQL やその他の外部コンポーネントやプラグインのインポートをサポートしていません。
- Aurora MySQL は、データベースからすべてを復元するわけではありません。ソース MySQL データベースからデータベーススキーマと以下の項目の値を保存してから、作成後に復元された Aurora MySQL DB クラスターに追加することをお勧めします。
  - ユーザーアカウント

- 関数
- ストアドプロシージャ
- タイムゾーン情報。タイムゾーン情報は、Aurora MySQL DB クラスターのローカルオペレーティングシステムからロードされます。詳細については、「Amazon Aurora DB クラスターのローカルタイムゾーン」を参照してください。

開始する前に

Amazon S3 バケットにデータをコピーし、それらのファイルから DB クラスターを復元するには、事前に以下の作業を行う必要があります。

- Percona XtraBackup をローカルサーバーにインストールします。
- お客様に代わって Aurora MySQL が Amazon S3 バケットにアクセスすることを許可します。

Percona XtraBackup のインストール

Amazon Aurora では、Percona XtraBackup を使用して作成されたファイルから DB クラスターを復元できます。Percona XtraBackup は、「ソフトウェアのダウンロード - Percona」からインストールできます。

MySQL 5.7 の移行では、Percona XtraBackup 2.4 を使用します。

MySQL 8.0 の移行では、Percona XtraBackup 8.0 を使用します。Percona XtraBackup バージョンがソースデータベースのエンジンバージョンと互換性があることを確認してください。

必要なアクセス許可

MySQL データを Amazon Aurora MySQL DB クラスターに移行するには、複数のアクセス権限が必要です。

- Amazon S3 バケットから新しいクラスターを作成することを Aurora に要求するユーザーは、AWS アカウントのバケットを一覧表示するためのアクセス許可を必要とします。このアクセス許可は、AWS Identity and Access Management (IAM) ポリシーを使用してユーザーに付与します。
- Aurora は、ユーザーに代わって Amazon Aurora MySQL DB クラスターを作成するために必要なファイルが保存されている Amazon S3 バケットにアクセスするためのアクセス許可を必要とします。必要なアクセス許可を Aurora に付与するには、IAM サービスロールを使用します。

- リクエストを実行するユーザーには、AWS アカウントの IAM ロールをリストするアクセス許可も必要です。
- リクエスト元のユーザーが IAM サービスロールを自分で作成したり、(コンソールから) Aurora に IAM サービスロールを作成することを要求したりする場合、そのユーザーには AWS アカウントの IAM ロールを作成するためのアクセス許可が必要です。
- 移行プロセス中にデータを暗号化する場合には、移行を実行するユーザーの IAM ポリシーを更新して、バックアップの暗号化に使用した AWS KMS keys へのアクセス権限を RDS に付与します。手順については、AWS KMS リソースにアクセスするための IAM ポリシーの作成 を参照してください。

例えば、次の IAM ポリシーでは、コンソールを使用して IAM ロールの一覧表示、IAM ロールの作成、アカウントの Amazon S3 バケットの一覧表示、および KMS キーの一覧表示を行うために必要な最小限のアクセス許可をユーザーに付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListRoles",
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy",
                "s3:ListBucket",
                "kms:ListKeys"
            ],
            "Resource": "*"
        }
    ]
}
```

さらに、ユーザーが IAM ロールを Amazon S3 バケットに関連付けるためには、IAM ユーザーに、その IAM ロールの `iam:PassRole` アクセス権限が必要です。このアクセス権限により、ユーザーが Amazon S3 バケットに関連付けることができる IAM ロールを管理者が制限できます。

例えば、次の IAM ポリシーでは、`S3Access` という名前のロールをユーザーが Amazon S3 バケットに関連付けることができます。

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"AllowS3AccessRole",
            "Effect":"Allow",
            "Action":"iam:PassRole",
            "Resource":"arn:aws:iam::123456789012:role/S3Access"
        }
    ]
}
```

IAM ユーザーのアクセス許可の詳細については、「ポリシーを使用したアクセスの管理」を参照してください。

IAM サービスロールの作成

[新規ロールの作成] オプション (このトピックで後述します) を選択して、AWS Management Console でロールを作成することができます。このオプションを選択して新しいロールの名前を指定すると、この指定した名前を使用して Aurora から Amazon S3 バケットにアクセスするために必要な IAM サービスロールが Aurora で作成されます。

または、次の手順を使用して手動でロールを作成できます。

Aurora から Amazon S3 にアクセスするための IAM ロールを作成するには

1. 「Amazon S3 リソースにアクセスするための IAM ポリシーの作成」の各ステップを実行します。
2. 「Amazon Aurora が AWS のサービスにアクセスすることを許可する IAM ロールの作成」の各ステップを実行します。
3. 「IAM ロールと Amazon Aurora MySQL DB クラスターの関連付け」の各ステップを実行します。

Amazon Aurora MySQL DB クラスターとして復元するファイルのバックアップ

Percona XtraBackup を使用して MySQL データベースファイルの完全バックアップを作成し、そのバックアップファイルを Amazon S3 バケットにアップロードできます。または、Percona XtraBackup を使用して MySQL データベースファイルをバックアップ済みである場合は、既存の完全および増分バックアップディレクトリおよびファイルを Amazon S3 バケットにアップロードできます。

トピック

- Percona XtraBackup での完全バックアップの作成
- Percona XtraBackup での増分バックアップの使用
- バックアップに関する考慮事項

Percona XtraBackup での完全バックアップの作成

Aurora MySQL DB クラスターを作成するために Amazon S3 から復元できる、MySQL データベースファイルのフルバックアップを作成するには、Percona XtraBackup ユーティリティ (`xtrabackup`) を使用してデータベースをバックアップします。

例えば、次のコマンドは MySQL データベースのバックアップを作成し、ファイルを `/on-premises/s3-restore/backup` フォルダに保存します。

```
xtrabackup --backup --user=<myuser> --password=<password> --target-dir=</on-premises/
s3-restore/backup>
```

バックアップを 1 つのファイル (必要に応じて分割できます) に圧縮する場合、以下のいずれかの形式を使用して `--stream` オプションを使用してバックアップを保存できます。

- Gzip (.gz)
- tar (.tar)
- Percona xbstream (.xbstream)

次のコマンドでは、複数の Gzip ファイルに分割された MySQL データベースのバックアップを作成します。

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=tar \
   --target-dir=</on-premises/s3-restore/backup> | gzip - | split -d --bytes=500MB \
   - </on-premises/s3-restore/backup/backup>.tar.gz
```

次のコマンドでは、複数の tar ファイルに分割された MySQL データベースのバックアップを作成します。

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=tar \
   --target-dir=</on-premises/s3-restore/backup> | split -d --bytes=500MB \
   - </on-premises/s3-restore/backup/backup>.tar
```

例えば、次のコマンドでは、複数の xbstream ファイルに分割された MySQL データベースのバックアップを作成します。

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=xbstream \
   --target-dir=</on-premises/s3-restore/backup> | split -d --bytes=500MB \
   - </on-premises/s3-restore/backup/backup>.xbstream
```

> **Note**
>
> 次のエラーが表示される場合は、コマンドにファイル形式が混在していることが原因となっている可能性があります。
>
> ```
> ERROR:/bin/tar: This does not look like a tar archive
> ```

Percona XtraBackup ユーティリティを使用して MySQL データベースをバックアップしたら、バックアップディレクトリおよびファイルを Amazon S3 バケットにコピーできます。

ファイルを作成して Amazon S3 バケットにアップロードする方法については、Amazon S3 入門ガイドの「Amazon Simple Storage Service のスタート方法」を参照してください。

Percona XtraBackup での増分バックアップの使用

Amazon Aurora MySQL は、Percona XtraBackup を使用して作成された完全バックアップおよび増分バックアップの両方をサポートしています。Percona XtraBackup を使用して MySQL データベースファイルの完全および増分バックアップを作成済みである場合は、完全バックアップを作成して Amazon S3 にアップロードする必要はありません。代わりに、既存の完全および増分バックアップのディレクトリおよびファイルを Amazon S3 バケットにコピーして、多大な時間を節約できます。詳細については、Percona のウェブサイトで「インクリメンタルバックアップを作成する」を参照してください。

既存の完全および増分バックアップファイルを Amazon S3 バケットにコピーするときは、ベースディレクトリのコンテンツを再帰的にコピーする必要があります。これらのコンテンツには、完全バックアップと、すべての増分バックアップディレクトリおよびファイルが含まれます。このコピーには、Amazon S3 バケットのディレクトリ構造を維持する必要があります。Aurora は、すべてのファイルとディレクトリを反復処理します。Aurora は、増分バックアップを含む `xtrabackup_checkpoints` ファイルを使用し、ベースディレクトリを識別します。また、ログシーケンス番号 (LSN) の範囲に従って増分バックアップを整列させます。

ファイルを作成して Amazon S3 バケットにアップロードする方法については、Amazon S3 入門ガイドの「Amazon Simple Storage Service のスタート方法」を参照してください。

バックアップに関する考慮事項

Aurora では、Percona XtraBackup を使用して作成された部分バックアップがサポートされていません。データベースの出典ファイルをバックアップするときに、`--tables`、`--tables-exclude`、`--tables-file`、`--databases`、`--databases-exclude`、または `--databases-file` オプションを使用して部分バックアップを作成することはできません。

Percona XtraBackup を使用したデータベースのバックアップの詳細については、Percona ウェブサイトの「Percona XtraBackup - ドキュメント」および「バイナリログの使用」を参照してください。

Aurora では、Percona XtraBackup を使用して作成された差分バックアップがサポートされています。詳細については、Percona のウェブサイトで「インクリメンタルバックアップを作成する」を参照してください。

Aurora では、ファイル名に基づいてバックアップファイルを使用します。ファイル形式に基づいた適切なファイル拡張子でバックアップファイルの名前を付けてください。例えば、Percona xbstream 形式を使用して保存されるファイルでは、`.xbstream` のようにします。

Aurora では、アルファベット順および通常の数値順にバックアップファイルを使用します。バックアップファイルが適切な順序で書き込まれ、名前が付けられるように、`split` コマンドを発行するときは必ず `xtrabackup` オプションを使用します。

Amazon S3 では、Amazon S3 バケットにアップロードするファイルのサイズが 5 TB に制限されます。データベースのバックアップデータが 5 TB を超える場合は、`split` コマンドを使用して、それぞれが 5 TB 未満の複数のファイルにバックアップファイルを分割します。

Aurora から Amazon S3 バケットにアップロードする出典ファイルの数は百万までに制限されています。場合によっては、すべての完全および増分バックアップを含むデータベースのバックアップデータには多数のファイルがあることがあります。この場合、tarball (.tar.gz) ファイルを使用して完全および増分バックアップを Amazon S3 バケットに保存します。

Amazon S3 バケットにファイルをアップロードしたら、サーバー側の暗号化を使用してデータを暗号化します。その後、この暗号化されたファイルから Amazon Aurora MySQL DB クラスターを復元できます。Amazon Aurora MySQL は、以下のタイプのサーバー側の暗号化を使用して、暗号化されたファイルを含む DB クラスターを復元できます。

- Amazon S3 管理キー (SSE-S3) によるサーバー側の暗号化。各オブジェクトは、強力な多要素暗号化を採用した一意のキーで暗号化されています。
- AWS KMS 管理キー (SSE-KMS) によるサーバー側の暗号化 - SSE-S3 に類似していますが、ユーザーによる暗号キーの作成と管理オプションや、その他点で異なります。

Amazon S3 バケットにファイルをアップロードするときにサーバー側の暗号化を使用する方法については、Amazon S3 デベロッパーガイドの「サーバー側の暗号化を使用したデータの保護」を参照してください。

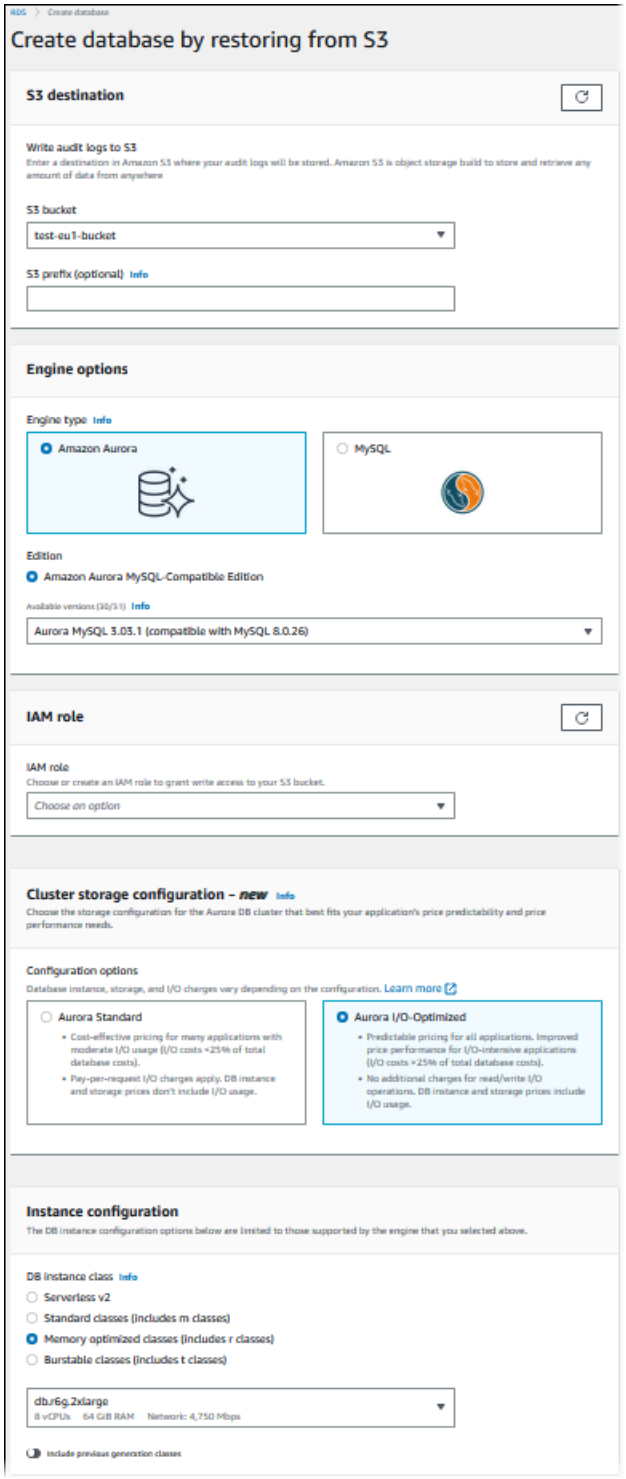
Amazon S3 バケットからの Amazon Aurora MySQL DB クラスターの復元

Amazon RDS コンソールを使用して、Amazon S3 バケットからバックアップファイルを復元して新しい Amazon Aurora MySQL DB クラスターを作成できます。

Amazon S3 バケットのファイルから Amazon Aurora MySQL DB クラスターを復元するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. Amazon RDS コンソールの右上で、DB クラスターを作成する AWS リージョンを選択します。データベースバックアップを含む Amazon S3 バケットと同じ AWS リージョンを選択します。
3. ナビゲーションペインで、[データベース]、[S3 から復元] の順に選択します。
4. [S3 から復元する] を選択します。

   [S3 から復元してデータベースを作成する] ページが表示されます。



5. S3 送信先で

   a. バックアップファイルを含む S3 バケットを選択します。

   b. (オプション) [S3 フォルダパスプレフィックス] で、Amazon S3 バケットに保存されているファイルのファイルパスのプレフィックスを入力します。

プレフィックスを指定しない場合、RDS は S3 バケットのルートフォルダにあるすべてのファイルとフォルダを使用して DB インスタンスを作成します。プレフィックスを指定すると、RDS はファイルのパスが指定されたプレフィックスで始まる S3 バケットのファイルとフォルダを使用して DB インスタンスを作成します。

例えば、複数のバックアップファイルのセットを S3 のサブフォルダ (backup) 内に保存し、各セットは独自のディレクトリ (gzip_backup1、gzip_backup2 など) 内にあるとします。この場合、プレフィックス backups/gzip_backup1 を指定して、gzip_backup1 フォルダのファイルから復元することができます。

6. [エンジンオプション]で

   a. [エンジンのタイプ] で [Amazon Aurora] を選択します。

   b. [バージョン] で、復元した DB インスタンスの Aurora MySQL エンジンのバージョンを選択します。

7. IAM ロールでは、既存の IAM ロールを選択します。

8. (オプション) [新しいロールを作成する]を選択して、新しい IAM ロールを持つこともできます。その場合、

   a. IAM ロール名を入力します。

   b. KMS キーへのアクセスを許可するかどうかを選択します。

      - バックアップファイルを暗号化していない場合には、[いいえ] を選択します。
      - Amazon S3 にバックアップファイルをアップロードした際に、このファイルを AES-256 (SSE-S3) で暗号化している場合には、[いいえ] を選択します。この設定により、データは自動的に復号化されます。
      - Amazon S3 にバックアップファイルをアップロードした際に、このファイルで AWS KMS (SSE-KMS) によるサーバー側の暗号化を実行した場合には、[はい] を選択します。次に、AWS KMS key の適切な KMS キーを選択します。

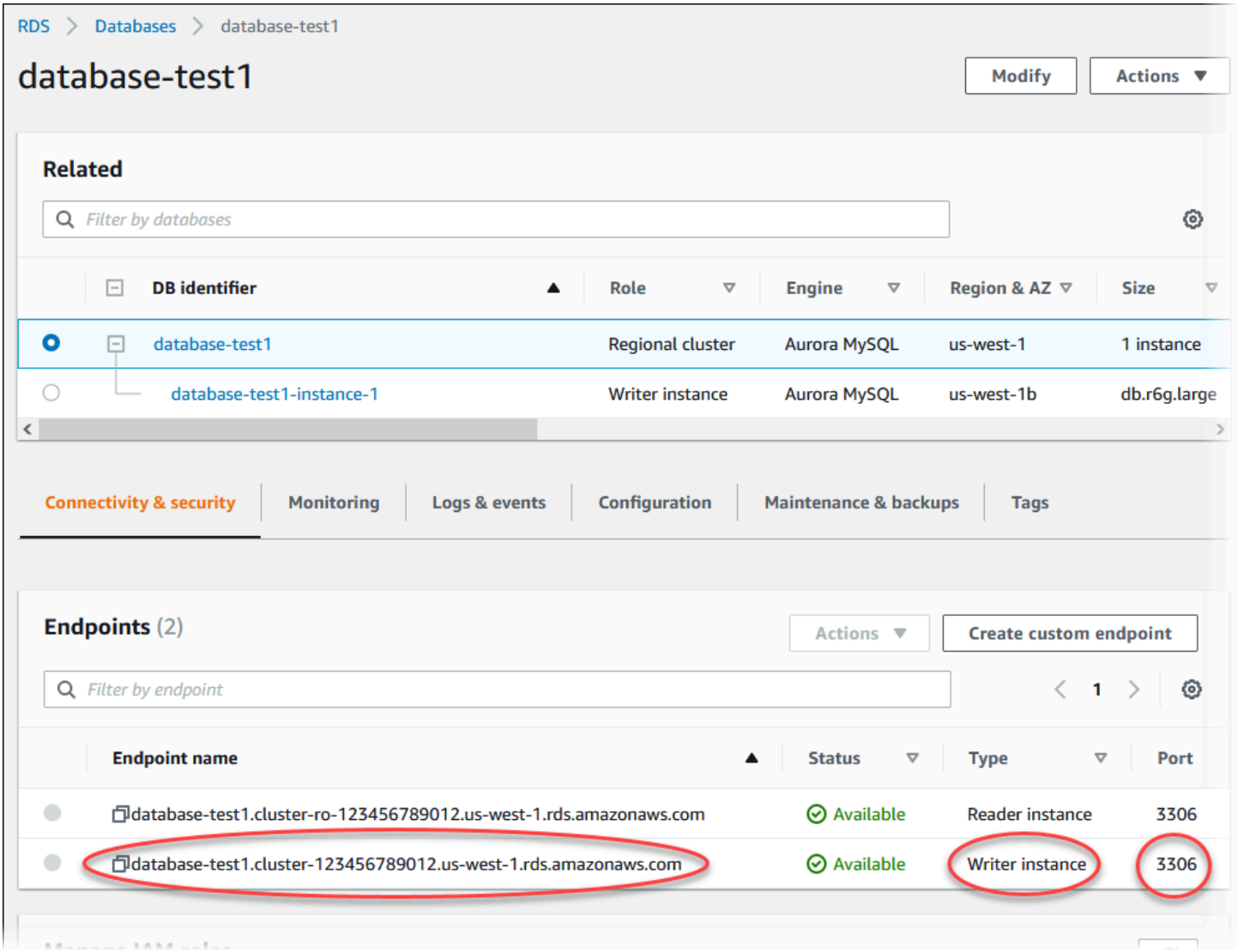
        AWS Management Console は、Aurora によるデータの復号を許可する IAM ポリシーを作成します。

詳細については、Amazon S3 デベロッパーガイドの「サーバー側の暗号化を使用したデータの保護」を参照してください。

9. DB クラスターストレージ設定、DB インスタンスクラス、DB クラスター識別子やログイン認証情報など、DB クラスターの設定を選択します。各設定の詳細については、「Aurora DB クラスターの設定」を参照してください。
10. 必要に応じて、Aurora MySQL DB クラスターの追加設定をカスタマイズします。
11. [データベースの作成] を選択して Aurora DB インスタンスを起動します。

Amazon RDS コンソールでは、新しい DB インスタンスが DB インスタンスのリストに表示されます。DB インスタンスが作成されて使用できるようになるまで、DB インスタンスのステータスは [作成中] となります。ステータスが [available] に変わったら、DB クラスターのプライマリインスタンスに接続できます。DB インスタンスクラスと割り当てられたストレージによっては、新しいインスタンスを使用できるようになるまで数分かかることがあります。

新しく作成したクラスターを表示するには、Amazon RDS コンソールで [データベース] ビューを選択し、DB クラスターを選択します。詳細については、「Amazon Aurora DB クラスターの表示」を参照してください。



DB クラスターのポートと書き込みエンドポイントをメモします。DB クラスターの書き込みエンドポイントとポートは、書き込みまたは読み取りオペレーションを実行するすべてのアプリケーションの JDBC 接続文字列と ODBC 接続文字列で使用します。

レプリケーションを使用して、Amazon Aurora MySQL DB クラスターと MySQL データベースを同期する

移行中のダウンタイムを減少あるいはなくすためには、Aurora MySQL DB クラスターに対して MySQL データベースで行われるトランザクションをレプリケートできます。レプリケーションは、移行中に発生する MySQL データベース上のトランザクションに DB クラスターが追いつくようにします。DB クラスターが完全に追いついたら、レプリケーションを停止し、Aurora MySQL への移行を終了できます。

トピック

- 暗号化レプリケーション用に外部の MySQL データベースと Aurora MySQL DB クラスターを設定する
- Amazon Aurora MySQL DB クラスターと外部の MySQL データベースを同期する

暗号化レプリケーション用に外部の MySQL データベースと Aurora MySQL DB クラスターを設定する

データを安全に複製するには、暗号化されたレプリケーションを使用できます。

> **Note**
>
> 暗号化レプリケーションを使う必要がない場合、このステップをスキップして、「Amazon Aurora MySQL DB クラスターと外部の MySQL データベースを同期する」のステップに進むことができます。

暗号化レプリケーションを使用するための前提条件は次のとおりです。

- Secure Sockets Layer (SSL) は、外部の MySQL プライマリデータベースで有効になっている必要があります。
- クライアントのキーとクライアントの証明書が Aurora MySQL DB クラスター用に準備されている必要があります。

暗号化のレプリケーション中、Aurora MySQL DB クラスターはクライアントとして MySQL データベースサーバーに動作します。Aurora MySQL 用の証明書およびキーは、.pem 形式のファイルにあります。

暗号化レプリケーションのために、外部の MySQL データベースおよび Aurora MySQL DB クラスターを設定するには

1. 暗号化レプリケーションのための準備があることを確認してください。
   - 外部の MySQL プライマリデータベースで SSL が有効になっておらず、クライアントキーとクライアント証明書が準備されていない場合、MySQL データベースサーバーで SSL を有効にし、必要なクライアントキーとクライアントの証明書を生成します。
   - SSL が外部プライマリで有効になっている場合は、Aurora MySQL DB クラスターにクライアントキーおよび証明書を提供します。これらがない場合は、Aurora MySQL DB クラスター用

に新しいキーと証明書を生成します。クライアント証明書に署名するには、外部の MySQL プライマリデータベースで SSL の設定に使用した認証局キーが必要です。

詳細については、MySQL ドキュメントの「openssl を使って SSL 証明書とキーを作成する」を参照してください。

認証局証明書、クライアントキーおよびクライアント証明書が必要となります。

2. SSL を使用して、プライマリユーザーとして Aurora MySQL DB クラスターに接続します。

   SSL で Aurora MySQL DB クラスターに接続する詳細については、「Aurora MySQL DB クラスターでの TLS の使用」を参照してください。

3. mysql.rds_import_binlog_ssl_material ストアドプロシージャを実行して、Aurora MySQL DB クラスターに SSL 情報をインポートします。

   `ssl_material_value` パラメータには、正しい JSON ペイロードで Aurora MySQL DB クラスター用の .pem 形式から情報を挿入します。

   次の例では、SSL 情報を Aurora MySQL DB クラスターにインポートします。.pem 形式ファイルでは、通常の場合、コード本文に例に示されるコード本文より長くなっています。

```
call mysql.rds_import_binlog_ssl_material(
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

詳細については、mysql.rds_import_binlog_ssl_materialおよびAurora MySQL DB クラスターでの TLS の使用を参照してください。

> **Note**
>
> 手順を実行したあと、シークレットはファイルに保存されます。ファイルを後で消去するには、mysql.rds_remove_binlog_ssl_material ストアドプロシージャを実行できます。

Amazon Aurora MySQL DB クラスターと外部の MySQL データベースを同期する

レプリケーションを使用して、Amazon Aurora MySQL DB クラスターと MySQL データベースを同期できます。

レプリケーションを使用して、Aurora MySQL DB クラスターと MySQL データベースを同期するには

1. 外部の MySQL データベース用の /etc/my.cnf ファイルに関連するエントリがあることを確認します。

   暗号化レプリケーションが求められない場合、外部 MySQL データベースがバイナリログ (binlogs) が有効化されて、SSL が無効化された状態でスタートすることを確認します。以下に示すのは、非暗号化データ用の /etc/my.cnf ファイルの関連するエントリです。

   ```
   log-bin=mysql-bin
   server-id=2133421
   innodb_flush_log_at_trx_commit=1
   sync_binlog=1
   ```

   暗号化レプリケーションが求められる場合、外部 MySQL データベースが SSL およびバイナリログ有効化された状態でスタートすることを確認します。/etc/my.cnf ファイルのエントリには、MySQL データベースサーバーのための .pem ファイルの場所が含まれます。

   ```
   log-bin=mysql-bin
   server-id=2133421
   innodb_flush_log_at_trx_commit=1
   sync_binlog=1
   ```

```
# Setup SSL.
ssl-ca=/home/sslcerts/ca.pem
ssl-cert=/home/sslcerts/server-cert.pem
ssl-key=/home/sslcerts/server-key.pem
```

次のコマンドを使って、SSL が有効であることを確認できます。

```
mysql> show variables like 'have_ssl';
```

出力は次のようになります。

```
+~-~-~-~-~-~-~-~-~-~-~-~-~-~-~---+~-~-~-~-~-~---+
| Variable_name | Value |
+~-~-~-~-~-~-~-~-~-~-~-~-~-~-~---+~-~-~-~-~-~---+
| have_ssl      | YES   |
+~-~-~-~-~-~-~-~-~-~-~-~-~-~-~---+~-~-~-~-~-~---+
1 row in set (0.00 sec)
```

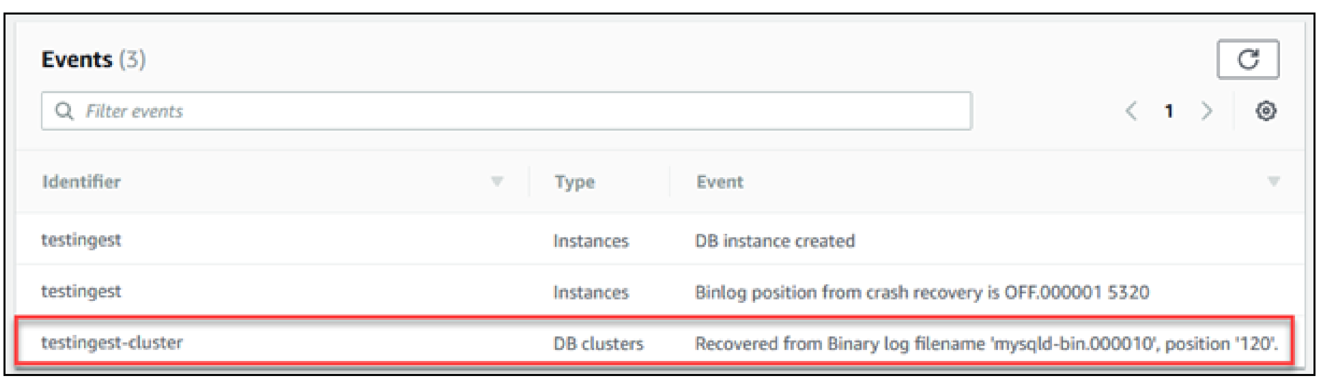
2. レプリケーションをスタートするバイナリログの位置を決定します。レプリケーションをスタートする位置は後のステップで指定します。

   AWS Management Console の使用

   a. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。

   b. ナビゲーションペインの [Events] (イベント) を選択します。

   c. [イベント] リストで、[Recovered from Binary log filename (バイナリログのファイル名から復旧済み)] イベントの位置をメモします。



AWS CLI の使用

describe-events AWS CLI コマンドを使用sるうことにより、binlog ファイルの名前と場所を取得することもできます。describe-events コマンドの例を以下に示します。

```
PROMPT> aws rds describe-events
```

出力で、バイナリログの位置を示すイベントを特定します。

3. 外部の MySQL データベースに接続したら、レプリケーションに使用するユーザーを作成します。このアカウントはレプリケーション専用に使用され、セキュリティを強化するためにドメインに制限する必要があります。次に例を示します。

```
mysql> CREATE USER '<user_name>'@'<domain_name>' IDENTIFIED BY '<password>';
```

ユーザーには REPLICATION CLIENT および REPLICATION SLAVE 権限が必要です。ユーザーのこれらの権限を付与します。

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO
 '<user_name>'@'<domain_name>';
```

暗号化レプリケーションを使用する必要がある場合は、レプリケーションのユーザーに対して SSL 接続を要求します。例えば、以下のステートメントを使用して、ユーザーアカウント *<user_name>* に SSL 接続を要求できます。

```
GRANT USAGE ON *.* TO '<user_name>'@'<domain_name>' REQUIRE SSL;
```

> **Note**
>
> REQUIRE SSL が含まれていない場合には、レプリケーション接続がメッセージの表示なしで非暗号化接続に戻る場合があります。

4. Amazon RDS コンソールで、外部 MySQL データベースをホストするサーバーの IP アドレスを、Aurora MySQL DB クラスターの VPC セキュリティグループに追加します。VPC セキュ

リティグループの変更方法の詳細については、Amazon Virtual Private Cloudユーザーガイドの「VPC のセキュリティグループ」を参照してください。

外部の MySQL データベースと通信できるようにするために、Aurora MySQL DB クラスターの IP アドレスからの接続を許可するようにローカルネットワークを設定することも必要になる場合があります。Aurora MySQL DB クラスターの IP アドレスを確認するには、host コマンドを使用します。

```
host <db_cluster_endpoint>
```

このホスト名は、Aurora MySQL DB クラスターのエンドポイントからの DNS 名です。

5. mysql.rds_reset_external_master (Aurora MySQL バージョン 2) または mysql.rds_reset_external_source (Aurora MySQL バージョン 3) ストアプロシージャを実行して、バイナリログレプリケーションを有効化します。このストアプロシージャの構文は次のとおりです。

```
CALL mysql.rds_set_external_master (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);

CALL mysql.rds_set_external_source (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);
```

パラメータの詳細については、「mysql.rds_reset_external_master (Aurora MySQL バージョン 2)」および「mysql.rds_reset_external_source (Aurora MySQL バージョン 3)」を参照してください。

mysql_binary_log_file_name および mysql_binary_log_file_location には、前のステップでメモした [Recovered from Binary log filename (バイナリログのファイル名から復旧済み)] イベントの位置を使用します。

Aurora MySQL DB クラスターのデータが暗号化されていない場合には、ssl_encryption パラメータを 0 に設定する必要があります。このデータが暗号化されている場合、ssl_encryption パラメータを 1 に設定する必要があります。

次の例では、暗号化されたデータがある Aurora MySQL DB クラスターの手順を実行します。

```
CALL mysql.rds_set_external_master(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'password',
  'mysql-bin.000010',
  120,
  1);

CALL mysql.rds_set_external_source(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'password',
  'mysql-bin.000010',
  120,
  1);
```

このストアプロシージャは、外部の MySQL データベースに接続し、そのバイナリログを読み取るために Aurora MySQL DB クラスターが使用するパラメータを設定します。データが暗号化されている場合には、SSL 認証局証明書、クライアント証明書およびクライアントキーもローカルディスクにダウンロードされます。

6. mysql.rds_start_replication ストアプロシージャを実行して、バイナリログレプリケーションをスタートします。

```
CALL mysql.rds_start_replication;
```

7. Aurora MySQL DB クラスターが MySQL レプリケーションプライマリーデータベースよりどれだけ遅れているかをモニタリングします。そのためには、Aurora MySQL DB クラスターに接続し、次のコマンドを実行します。

```
Aurora MySQL version 2:
SHOW SLAVE STATUS;

Aurora MySQL version 3:
SHOW REPLICA STATUS;
```

このコマンドの出力の `Seconds Behind Master` フィールドには、Aurora MySQL DB クラスターがどれだけ MySQL プライマリより遅れているかが示されます。この値が `0` (ゼロ) の場合、Aurora MySQL DB クラスターがMySQL プライマリに追いついたことを示し、レプリケーションを停止するための次のステップに進むことができます。

8. MySQL レプリケーションのMySQL プライマリデータベースに接続して、レプリケーションを停止します。これを行うには、mysql.rds_stop_replication ストアドプロシージャを実行します。

```
CALL mysql.rds_stop_replication;
```

Amazon Aurora MySQL への物理的な移行にかかる時間の短縮

Amazon Aurora MySQL へのデータベース移行プロセスをスピードアップするために、以下のデータベース修正を行うことができます。

> **Important**
>
> これらのアップデートは、本稼働データベースではなく、そのコピーに対して実行するようにしてください。このコピーをバックアップし、Aurora MySQL DB クラスターに復元することで、本稼働データベースのサービス停止を回避できます。

サポートされていないテーブルタイプ

Aurora MySQL はデータベーステーブル用の InnoDB エンジンのみをサポートしています。データベース内に MyISAM テーブルがある場合は、Aurora MySQL に移行する前にそれらのテーブルを変

換する必要があります。移行中の MyISAM から InnoDB への変換プロセスには、追加領域が必要です。

領域不足が発生する可能性を低く抑えて移行プロセスを高速化するには、すべての MyISAM テーブルを移行前に InnoDB テーブルに変換しておきます。処理後の InnoDB テーブルのサイズは、Aurora MySQL がそのテーブルに対して必要とするサイズと同じになります。MyISAM テーブルを InnoDB に変換するには、次のコマンドを実行します。

```
ALTER TABLE schema.table_name engine=innodb, algorithm=copy;
```

Aurora MySQL では、圧縮テーブルまたはページ (ROW_FORMAT=COMPRESSED または COMPRESSION = {"zlib"|"lz4"} を使用して作成されたテーブル) をサポートしていません。

スペースが不足する可能性を減らしたり、移行処理を高速化するには、ROW_FORMAT を DEFAULT、COMPACT、DYNAMIC または REDUNDANT に設定して圧縮テーブルを展開します。圧縮ページの場合は、COMPRESSION="none" を設定します。

詳細については、MySQL ドキュメントの「InnoDB 行形式」および「InnoDB テーブルおよびページ圧縮」を参照してください。

既存の MySQL DB インスタンスで以下の SQL スクリプトを使用して、データベースの MyISAM テーブルまたは圧縮テーブルのリストを表示できます。

```
-- This script examines a MySQL database for conditions that block
-- migrating the database into Aurora MySQL.
-- It must be run from an account that has read permission for the
-- INFORMATION_SCHEMA database.

-- Verify that this is a supported version of MySQL.

select msg as `==> Checking current version of MySQL.`
from
  (
  select
    'This script should be run on MySQL version 5.6 or higher. ' +
    'Earlier versions are not supported.' as msg,
    cast(substring_index(version(), '.', 1) as unsigned) * 100 +
      cast(substring_index(substring_index(version(), '.', 2), '.', -1)
      as unsigned)
    as major_minor
  ) as T
where major_minor <> 506;
```

```
-- List MyISAM and compressed tables. Include the table size.

select concat(TABLE_SCHEMA, '.', TABLE_NAME) as `==> MyISAM or Compressed Tables`,
round(((data_length + index_length) / 1024 / 1024), 2) "Approx size (MB)"
from INFORMATION_SCHEMA.TABLES
where
  ENGINE <> 'InnoDB'
  and
  (
    -- User tables
    TABLE_SCHEMA not in ('mysql', 'performance_schema',
                         'information_schema')
    or
    -- Non-standard system tables
    (
      TABLE_SCHEMA = 'mysql' and TABLE_NAME not in
        (
          'columns_priv', 'db', 'event', 'func', 'general_log',
          'help_category', 'help_keyword', 'help_relation',
          'help_topic', 'host', 'ndb_binlog_index', 'plugin',
          'proc', 'procs_priv', 'proxies_priv', 'servers', 'slow_log',
          'tables_priv', 'time_zone', 'time_zone_leap_second',
          'time_zone_name', 'time_zone_transition',
          'time_zone_transition_type', 'user'
        )
    )
  )
  or
  (
    -- Compressed tables
       ROW_FORMAT = 'Compressed'
  );
```

サポートされていない権限を持つユーザーアカウント

Aurora MySQL でサポートされていない権限を持つユーザーアカウントは、サポートされていない権限なしでもインポートされます。サポートされている権限のリストについては、「ロールベースの特権モデル」を参照してください。

ソースデータベースで次の SQL クエリを実行すると、サポートされていない権限を持つユーザーアカウントを一覧表示できます。

```
SELECT
    user,
    host
FROM
    mysql.user
WHERE
    Shutdown_priv = 'y'
    OR File_priv = 'y'
    OR Super_priv = 'y'
    OR Create_tablespace_priv = 'y';
```

Aurora MySQL バージョン 3 に対する動的権限

動的権限はインポートされません。Aurora MySQL バージョン 3 は、以下の動的権限をサポートしています。

```
'APPLICATION_PASSWORD_ADMIN',
'CONNECTION_ADMIN',
'REPLICATION_APPLIER',
'ROLE_ADMIN',
'SESSION_VARIABLES_ADMIN',
'SET_USER_ID',
'XA_RECOVER_ADMIN'
```

次のサンプルスクリプトは、サポートされている動的権限を Aurora MySQL DB クラスターのユーザーアカウントに付与します。

```
-- This script finds the user accounts that have Aurora MySQL supported dynamic
 privileges
-- and grants them to corresponding user accounts in the Aurora MySQL DB cluster.

/home/ec2-user/opt/mysql/8.0.26/bin/mysql -uusername -pxxxxx -P8026 -h127.0.0.1 -BNe
 "SELECT
  CONCAT('GRANT ', GRANTS, ' ON *.* TO ', GRANTEE ,';') AS grant_statement
  FROM (select GRANTEE, group_concat(privilege_type) AS GRANTS FROM
 information_schema.user_privileges
      WHERE privilege_type IN (
        'APPLICATION_PASSWORD_ADMIN',
        'CONNECTION_ADMIN',
        'REPLICATION_APPLIER',
        'ROLE_ADMIN',
        'SESSION_VARIABLES_ADMIN',
```

```
        'SET_USER_ID',
        'XA_RECOVER_ADMIN')
      AND GRANTEE NOT IN (\"'mysql.session'@'localhost'\",
\"'mysql.infoschema'@'localhost'\",\"'mysql.sys'@'localhost'\") GROUP BY GRANTEE)
      AS PRIVGRANTS; " | /home/ec2-user/opt/mysql/8.0.26/bin/mysql -u master_username -
p master_password -h DB_cluster_endpoint
```

定義者として 'rdsadmin'@'localhost' を使用して保存されたオブジェクト

`'rdsadmin'@'localhost'` が定義子である関数、プロシージャ、ビュー、イベント、トリガーは、インポートされません。

ソース MySQL データベースで以下の SQL スクリプトを使用すると、サポートされていない定義子を持つストアド オブジェクトを一覧表示できます。

```
-- This SQL query lists routines with `rdsadmin`@`localhost` as the definer.

SELECT
    ROUTINE_SCHEMA,
    ROUTINE_NAME
FROM
    information_schema.routines
WHERE
    definer = 'rdsadmin@localhost';

-- This SQL query lists triggers with `rdsadmin`@`localhost` as the definer.

SELECT
    TRIGGER_SCHEMA,
    TRIGGER_NAME,
    DEFINER
FROM
    information_schema.triggers
WHERE
    DEFINER = 'rdsadmin@localhost';

-- This SQL query lists events with `rdsadmin`@`localhost` as the definer.

SELECT
    EVENT_SCHEMA,
    EVENT_NAME
FROM
    information_schema.events
```

```
WHERE
    DEFINER = 'rdsadmin@localhost';

-- This SQL query lists views with `rdsadmin`@`localhost` as the definer.
SELECT
    TABLE_SCHEMA,
    TABLE_NAME
FROM
    information_schema.views
WHERE
    DEFINER = 'rdsadmin@localhost';
```

mysqldump を使用した MySQL から Amazon Aurora MySQL への論理的移行

Amazon Aurora MySQL は MySQL と互換性があるデータベースであるため、mysqldump ユーティリティを使用して MySQL または MariaDB データベースから既存の Aurora MySQL DB クラスターにデータをコピーできます。

巨大な MySQL データベースを移行する方法については、「ダウンタイムを短縮して MySQL または MariaDB DB インスタンスにデータをインポートする」を参照してください データ量の少ない MySQL データベースについては、「MySQL DB または MariaDB DB から MySQL または MariaDB DB インスタンスへのデータのインポート」を参照してください。

RDS for MySQL DB インスタンスから Amazon Aurora MySQL DB クラスターへのデータの移行

RDS for MySQL DB インスタンスから Amazon Aurora MySQL DB クラスターにデータを移行 (コピー) できます。

トピック

> **Note**
>
> Amazon Aurora MySQL は MySQL と互換性があるため、MySQL データベースと Amazon Aurora MySQL DB クラスターの間でレプリケーションをセットアップすることによって、MySQL データベースのデータを移行できます。詳細については、「Amazon Aurora でのレプリケーション」を参照してください。

Aurora への RDS for MySQL スナップショットの移行

RDS for MySQL DB インスタンスの DB スナップショットを移行して、Aurora MySQL DB クラスターを作成することができます。新しい Aurora MySQL DB クラスターには、元の RDS for MySQL DB インスタンスのデータが入力されます。DB スナップショットは、Aurora MySQL と互換性がある MySQL バージョンを実行している Amazon RDS DB インスタンスから作成されたものである必要があります。

手動で作成された DB スナップショットと自動的に作成された DB スナップショットのどちらも移行できます。DB クラスターが作成された後、オプションの Aurora レプリカを作成できます。

> **Note**
>
> ソース RDS for MySQL DB インスタンスの Aurora リードレプリカを作成することにより、RDS for MySQL DB インスタンスを Aurora MySQL DB クラスターに移行することもできます。詳細については、「Aurora リードレプリカを使用した、RDS for MySQL DB インスタンスから Amazon Aurora MySQL DB クラスターへのデータの移行」を参照してください。

8.0.11、8.0.13、8.0.15 などの一部の古い MySQL 8.0 バージョンからは Aurora MySQL バージョン 3.05 以上に移行できません。移行する前に MySQL バージョン 8.0.28 にアップグレードすることをお勧めします。

実行する必要がある一般的なステップは次のとおりです。

1. Aurora MySQL DB クラスターをプロビジョニングするための容量を決定します。詳細については、「必要な容量」を参照してください。
2. コンソールを使用して、Amazon RDS MySQL インスタンスが置かれている AWS リージョン内にスナップショットを作成します。DB スナップショットの作成については、「DB スナップショットの作成」を参照してください。
3. DB スナップショットが DB クラスターと同じ AWS リージョン内にない場合は、Amazon RDS コンソールを使用して DB スナップショットをその AWS リージョンにコピーします。DB スナップショットのコピーについては、「DB スナップショットのコピー」を参照してください。
4. コンソールを使用して DB スナップショットを移行し、元の MySQL DB インスタンスと同じデータベースを持つ Aurora MySQL DB クラスターを作成します。

Warning

Amazon RDS では、各 AWS アカウントによる各 AWS リージョンへのスナップショットのコピーは 1 度に 1 つに制限されています。

必要な容量

MySQL DB インスタンスのスナップショットを Aurora MySQL DB クラスターに移行するとき、Aurora は、スナップショットのデータを移行する前に Amazon Elastic Block Store (Amazon EBS) ボリュームを使用してそのデータの書式を設定します。移行するデータの書式を設定するために追加容量が必要になる場合があります。

MyISAM テーブルではないテーブルおよび圧縮されていないテーブルのサイズは、最大 16 TB が可能です。MyISAM テーブルの場合、Aurora では、Aurora MySQL と互換性のあるテーブルに変換するために、ボリュームに追加のスペースが必要になります。圧縮されたテーブルの場合、Aurora では、圧縮されたテーブルを Aurora クラスターボリュームに保存する前に展開するため、ボリュームに追加のスペースが必要になります。追加のスペースが必要になるため、MySQL DB インスタンス

から移行される MyISAM テーブルおよび圧縮テーブルのサイズが 8 TB を超えていないことを確認する必要があります。

Amazon Aurora MySQL にデータを移行するために必要な容量の削減

Amazon Aurora に移行する前にデータベーススキーマを変更することもできます。このような変更は、次のような場合に便利です。

- 移行プロセスを迅速化したい。
- プロビジョニングするために必要な領域の量がわからない場合。
- データを移行しようとしたが、プロビジョニング済み領域の不足で移行が失敗した場合。

以下の変更を行うことで、データベースを Amazon Aurora に移行するプロセスを改善できます。

> **Important**
>
> これらの更新は、本稼働インスタンスではなく、本稼働データベースのスナップショットから復元された新しい DB インスタンスに対して実行します。その後、新しい DB インスタンスのスナップショットからデータを Aurora DB クラスターに移行することで、本稼働データベースに対するサービスの中断を回避できます。

テーブルタイプ	制限またはガイドライン
MyISAM テーブル	Aurora MySQL は InnoDB テーブルのみをサポートします。データベース内に MyISAM テーブルがある場合は、Aurora MySQL に移行する前にそれらのテーブルを変換する必要があります。移行中の MyISAM から InnoDB への変換プロセスには、追加領域が必要です。 領域不足が発生する可能性を低く抑えて移行プロセスを高速化するには、すべての MyISAM テーブルを移行前に InnoDB テーブルに変換しておきます。処理後の InnoDB テーブルのサイズは、Aurora MySQL がそのテーブルに対して必要とするサイズと同じになります。MyISAM テーブルを InnoDB に変換するには、次のコマンドを実行します。

テーブルタイプ	制限またはガイドライン
	`alter table <schema>.<table_name> engine=innodb, algorithm=copy;`
圧縮テーブル	Aurora MySQL では、圧縮テーブル (`ROW_FORMAT=COMPRESSED` を使用して作成されたテーブル) をサポートしていません。 スペースが不足する可能性を減らしたり、移行処理を高速化するには、`ROW_FORMAT` を `DEFAULT`、`COMPACT`、`DYNAMIC` または `REDUNDANT` に設定して圧縮テーブルを展開します。詳細については、MySQL ドキュメントの「InnoDB 行形式」を参照してください。

既存の MySQL DB インスタンスで以下の SQL スクリプトを使用して、データベースの MyISAM テーブルまたは圧縮テーブルのリストを表示できます。

```
-- This script examines a MySQL database for conditions that block
-- migrating the database into Amazon Aurora.
-- It needs to be run from an account that has read permission for the
-- INFORMATION_SCHEMA database.

-- Verify that this is a supported version of MySQL.

select msg as `==> Checking current version of MySQL.`
from
  (
  select
    'This script should be run on MySQL version 5.6 or higher. ' +
    'Earlier versions are not supported.' as msg,
    cast(substring_index(version(), '.', 1) as unsigned) * 100 +
      cast(substring_index(substring_index(version(), '.', 2), '.', -1)
      as unsigned)
    as major_minor
  ) as T
where major_minor <> 506;


-- List MyISAM and compressed tables. Include the table size.

select concat(TABLE_SCHEMA, '.', TABLE_NAME) as `==> MyISAM or Compressed Tables`,
```

```
round(((data_length + index_length) / 1024 / 1024), 2) "Approx size (MB)"
from INFORMATION_SCHEMA.TABLES
where
  ENGINE <> 'InnoDB'
  and
  (
    -- User tables
    TABLE_SCHEMA not in ('mysql', 'performance_schema',
                         'information_schema')
    or
    -- Non-standard system tables
    (
      TABLE_SCHEMA = 'mysql' and TABLE_NAME not in
        (
          'columns_priv', 'db', 'event', 'func', 'general_log',
          'help_category', 'help_keyword', 'help_relation',
          'help_topic', 'host', 'ndb_binlog_index', 'plugin',
          'proc', 'procs_priv', 'proxies_priv', 'servers', 'slow_log',
          'tables_priv', 'time_zone', 'time_zone_leap_second',
          'time_zone_name', 'time_zone_transition',
          'time_zone_transition_type', 'user'
        )
    )
  )
  or
  (
    -- Compressed tables
       ROW_FORMAT = 'Compressed'
  );
```

スクリプトでは、次の例のような出力が作成されます。この例では、MyISAM から InnoDB に変換する必要のある 2 つのテーブルを示しています。出力には、メガバイト (MB) 単位で示した各テーブルのおおよそのサイズも含まれています。

```
+---------------------------------+------------------+
| ==> MyISAM or Compressed Tables | Approx size (MB) |
+---------------------------------+------------------+
| test.name_table                 |          2102.25 |
| test.my_table                   |            65.25 |
+---------------------------------+------------------+
2 rows in set (0.01 sec)
```

RDS for MySQL DB スナップショットを Aurora MySQL DB クラスターに移行する

RDS for MySQL DB インスタンスの DB スナップショットを移行して、AWS Management Console または AWS CLI を使って、Aurora MySQL DB クラスターを作成することができます。新しい Aurora MySQL DB クラスターには、元の RDS for MySQL DB インスタンスのデータが入力されます。DB スナップショットの作成については、「DB スナップショットの作成」を参照してください。

DB スナップショットがデータを検索する AWS リージョン内にない場合は、DB スナップショットをその AWS リージョンにコピーします。DB スナップショットのコピーについては、「DB スナップショットのコピー」を参照してください。

コンソール

AWS Management Console を使用して DB スナップショットを移行すると、DB クラスターとプライマリインスタンスの両方を作成するために必要なアクションがコンソールによって実行されます。

新しい Aurora MySQL DB クラスターが、AWS KMS key を使用して保管中に暗号化されるよう選択することもできます。

AWS Management Console を使用して MySQL DB スナップショットを移行するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. MySQL DB インスタンスまたはスナップショットから移行をスタートします。

   DB インスタンスから移行をスタートするには:

   1. ナビゲーションペインで、[データベース] を選択し、MySQL DB インスタンスを選択します。
   2. [アクション]、[最新のスナップショットの移行] の順に選択します。

   スナップショットから移行をスタートするには:

   1. [スナップショット] を選択します。
   2. [スナップショット] ページで、Aurora MySQL DB クラスターに移行するスナップショットを選択します。
   3. [スナップショットのアクション]、[スナップショットの移行] の順に選択します。

[データベースの移行] ページが表示されます。

3. [データベースの移行] ページで以下の値を設定します。

   - [DB エンジンへの移行] : aurora を選択します。
   - [DB エンジンのバージョン]: Aurora MySQL DB クラスターの DB エンジンのバージョンを選択します。
   - DB インスタンスクラス: データベースに必要なストレージと容量を持つ DB インスタンスクラス (db.r3.large など) を選択します。Aurora クラスターボリュームは、データベースのデータ量が増えるにつれて自動的に増加します。Aurora クラスターボリュームは、最大 128 tebibytes (TiB) のサイズまで増やすことができます。そのため、現在のストレージ要件を満たしている DB インスタンスクラスを選択する必要があります。詳細については、「Amazon Aurora ストレージの概要」を参照してください。
   - DB インスタンス識別子: DB クラスター名を入力します。選択した AWS リージョン内で、自分のアカウントに対して一意であることが必要です。この識別子は、DB クラスター内のインスタンスのエンドポイントアドレスで使用されます。名前には、選択した AWS リージョンと DB エンジンなどを含めると理解しやすくなります (**aurora-cluster1** など)。

     DB インスタンス識別子には次の制約があります。

     - 1 ~ 63 文字の英数字またはハイフンを使用する必要があります。
     - 1 字目は文字である必要があります。
     - ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません。
     - 1 つの AWS アカウント、1 つの AWS リージョンにつき、すべての DB インスタンスにおいて一意である必要があります。
   - [Virtual Private Cloud (VPC)]: 既存の VPC がある場合は、その VPC 識別子 (vpc-a464d1c1 など) を選択することで、その VPC を Aurora MySQL DB クラスターで使用できます。VPC の作成方法の詳細については、「チュートリアル: DB クラスターで使用する VPC を作成する (IPv4 専用)」を参照してください。

     または、[新しい VPC の作成] を選択し、Aurora で自動的に VPC を作成します。
   - DB サブネットグループ: 既存のサブネットグループがある場合は、そのサブネットグループ識別子 (gs-subnet-group1 など) を選択して、サブネットグループを Aurora MySQL DB クラスターで使用できます。

または、[新しいサブネットグループを作成] を選択し、Aurora で自動的にサブネットグループを作成します。

- Public accessibility: DB クラスターのインスタンスが VPC 内のリソースからのみアクセスできることを指定するには、[いいえ] を選択します。DB クラスターのインスタンスがパブリックネットワーク上のリソースからアクセスできることを指定するには、[はい] を選択します。デフォルトは [はい] です。

  > **Note**
  >
  > 本番稼働用の DB クラスターは、お客様のアプリケーションサーバーのみがアクセスするため、パブリックサブネット内に配置する必要がない場合があります。DB クラスターをパブリックサブネットに配置する必要がない場合は、[パブリックアクセス可能] を [いいえ] に設定します。

- [アベイラビリティーゾーン]: Aurora MySQL DB クラスターのプライマリインスタンスをホストするアベイラビリティーゾーンを選択します。Aurora で自動的にアベイラビリティーゾーンを選択するには、[指定なし] を選択します。
- [データベースのポート]: Aurora MySQL DB クラスターのインスタンスへの接続に使用されるデフォルトのポートを入力します。デフォルトは `3306` です。

  > **Note**
  >
  > 会社のファイアウォールで MySQL のデフォルトポートである 3306 などのデフォルトポートへのアクセスが許可されない場合があります。この場合は、会社のファイアウォールによって許可されるポート値を指定します。そのポート値を覚えておいてください。後で Aurora MySQL DB クラスターに接続するときに使用します。

- [暗号化]: 新しい Aurora MySQL DB クラスターを保管時に暗号化するには、[暗号を有効化] を選択します。[Enable Encryption] (暗号化を有効化) を選択する場合、KMS キーを AWS KMS key 値として選択する必要があります。

  DB スナップショットが暗号化されていない場合は、暗号化キーを指定して保管時の DB クラスターを暗号化します。

  DB スナップショットが暗号化されている場合は、暗号化キーを指定し、その指定された暗号化キーを使用して保管時の DB クラスターを暗号化します。DB スナップショットで使用され

る暗号化キー、または別のキーを指定できます。暗号化された DB スナップショットから 非暗号化の DB クラスターを作成することはできません。

- [マイナーバージョン自動アップグレード]: この設定は Aurora MySQL DB クラスターに適用されません。

  Aurora MySQL のエンジンに関する更新の詳細については、「Amazon Aurora MySQL のデータベースエンジンの更新」を参照してください。

4. [移行] を選択して、DB スナップショットを移行します。
5. [インスタンス] を選択して、矢印アイコンを選択して DB クラスターの詳細を表示し、移行の進行状況をモニタリングします。詳細ページで、DB クラスターのプライマリインスタンスへの接続に使用されているクラスターエンドポイントがわかります。Aurora MySQL DB クラスターとの接続の詳細については、「Amazon Aurora DB クラスターへの接続」を参照してください。

AWS CLI

`restore-db-cluster-from-snapshot` コマンドを次のパラメータで使用することで、RDS for MySQL DB インスタンスの DB スナップショットから Aurora DB クラスターを作成できます。

- `--db-cluster-identifier` - 作成する DB クラスターの名前。
- `--engine aurora-mysql` — MySQL 5.7 互換または 8.0 互換 DB クラスターの場合
- `--kms-key-id` - DB スナップショットが暗号化されるかどうかに応じて、オプションで DB クラスターを暗号化するための AWS KMS key。
  - DB スナップショットが暗号化されていない場合は、暗号化キーを指定して保管時の DB クラスターを暗号化します。これを実行しない場合、DB クラスターは暗号化されません。
  - DB スナップショットが暗号化されている場合は、暗号化キーを指定し、その指定された暗号化キーを使用して保管時の DB クラスターを暗号化します。これを実行しない場合、保管時の DB クラスターは DB スナップショットの暗号化キーを使用して暗号化されます。

  > **Note**
  >
  > 暗号化された DB スナップショットから 非暗号化の DB クラスターを作成することはできません。

- `--snapshot-identifier` – 移行する DB スナップショットの Amazon リソースネーム (ARN)。Amazon RDS ARN の詳細については、「Amazon Relational Database Service (Amazon RDS)」を参照してください。

RestoreDBClusterFromSnapshot コマンドを使用して DB スナップショットを移行すると、DB クラスターとプライマリインスタンスの両方がこのコマンドによって作成されます。

この例では、*mydbcluster* という名前の MySQL 5.7 互換 DB クラスターを ARN が *mydbsnapshotARN* に設定されている DB スナップショットから作成します。

Linux、macOS、Unix の場合:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mydbcluster \
    --snapshot-identifier mydbsnapshotARN \
    --engine aurora-mysql
```

Windows の場合:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mydbcluster ^
    --snapshot-identifier mydbsnapshotARN ^
    --engine aurora-mysql
```

この例では、*mydbcluster* という名前の MySQL 5.7 互換 DB クラスターを ARN が *mydbsnapshotARN* に設定されている DB スナップショットから作成します。

Linux、macOS、Unix の場合:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mydbcluster \
    --snapshot-identifier mydbsnapshotARN \
    --engine aurora-mysql
```

Windows の場合:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mydbcluster ^
    --snapshot-identifier mydbsnapshotARN ^
    --engine aurora-mysql
```

Aurora リードレプリカを使用した、RDS for MySQL DB インスタンスから Amazon Aurora MySQL DB クラスターへのデータの移行

Aurora は、MySQL DB エンジンのバイナリログレプリケーション機能を使用して、ソース RDS for MySQL DB インスタンスの Aurora リードレプリカと呼ばれる特殊なタイプの DB クラスターを作成します。ソース RDS for MySQL DB インスタンスに加えられた更新は、Aurora リードレプリカに非同期的にレプリケートされます。

ソース RDS for MySQL DB インスタンスの Aurora リードレプリカを作成して RDS for MySQL DB インスタンスから Aurora MySQL DB クラスターに移行する場合は、この機能を使用することをお勧めします。RDS for MySQL DB インスタンスと Aurora リードレプリカとの間のレプリカラグが 0 の場合は、クライアントアプリケーションを Aurora リードレプリカに誘導してからレプリケーションを停止することで、Aurora リードレプリカをスタンドアロンの Aurora MySQL DB クラスターにすることができます。移行では、データの 1 テビバイト (TiB) ごとに数時間程度の時間がかかります。

Aurora を使用できるリージョンの一覧は、AWS 全般のリファレンス の「Amazon Aurora」を参照してください。

RDS for MySQL DB インスタンスの Aurora リードレプリカを作成すると、Amazon RDS により、ソース RDS for MySQL DB インスタンスの DB スナップショットが作成されます (このスナップショットは Amazon RDS に対してプライベートで、料金はかかりません)。その後 Amazon RDS は、DB スナップショットから Aurora リードレプリカにデータを移行します。DB スナップショットのデータが新しい Aurora MySQL DB クラスターに移行された後、Amazon RDS は、RDS for MySQL DB インスタンスと Aurora MySQL DB クラスターとの間でレプリケーションをスタートします。RDS for MySQL DB インスタンスに、InnoDB 以外のストレージエンジンを使用するテーブルまたは圧縮行形式を使用するテーブルが含まれている場合は、Aurora リードレプリカを作成する前に InnoDB ストレージエンジンと動的行形式が使用されるようにテーブルを変更することで、Aurora リードレプリカの作成プロセスをスピードアップできます。MySQL DB スナップショットを Aurora MySQL DB クラスターにコピーするプロセスの詳細については、「RDS for MySQL DB インスタンスから Amazon Aurora MySQL DB クラスターへのデータの移行」を参照してください。

1 つの RDS for MySQL DB インスタンスに対して作成できる Aurora リードレプリカは、1 つだけです。

> **Note**
>
> レプリケーションプライマリである RDS for MySQL DB インスタンスの MySQL データベースエンジンバージョンと Aurora MySQL との間に存在する特性の相違が原因で、レプ

リケーションの問題が発生することがあります。エラーが発生した場合のサポートについては、Amazon RDS コミュニティフォーラムを参照するか、AWS Supportまでお問い合わせください。
RDS for MySQL DB インスタンスに既に Aurora リードレプリカのソースがある場合は、Aurora リードレプリカを作成できません。
8.0.11、8.0.13、8.0.15 などの一部の古い RDS for MySQL 8.0 バージョンからは Aurora MySQL バージョン 3.05 以上に移行できません。移行する前に RDS for MySQL バージョン 8.0.28 にアップグレードすることをお勧めします。

MySQL リードレプリカの詳細については、「MariaDB、MySQL、PostgreSQL DB インスタンスのリードレプリカの使用」を参照してください。

Aurora リードレプリカの作成

コンソール、AWS CLI、または RDS API を使用して、RDS for MySQL DB インスタンスの Aurora リードレプリカを作成できます。

コンソール

ソース RDS for MySQL DB インスタンスから Aurora リードレプリカを作成するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. Aurora リードレプリカの出典として使用する MySQL DB インスタンスを選択します。
4. [アクション] で [Aurora リードレプリカの作成] を選択します。
5. 次の表の説明に従って、Aurora リードレプリカに使用する DB クラスターの仕様を選択します。

オプション	説明
DB インスタンスクラス	DB クラスターのプライマリインスタンスに対する処理要件やメモリ要件を定義する DB インスタンスクラスを選択します。DB インスタンスクラスのオプションについては、「Aurora DB インスタンスクラス」を参照してください。

オプション	説明
マルチ AZ 配置	[別のゾーンにレプリカを作成します] を選択し、フェイルオーバーをサポートするため、ターゲット AWS リージョン内の別のアベイラビリティーゾーンに新しい DB クラスターのスタンバイレプリカを作成します。複数のアベイラビリティーゾーンの詳細については、「リージョンとアベイラビリティーゾーン」を参照してください。
DB インスタンス識別子	Aurora リードレプリカ DB クラスターのプライマリインスタンスの名前を入力します。この識別子は、新しい DB クラスターのプライマリインスタンスのエンドポイントアドレスで使用されます。 DB インスタンス識別子には次の制約があります。 • 1 ~ 63 文字の英数字またはハイフンを使用する必要があります。 • 1 字目は文字である必要があります。 • ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません。 • これは AWS リージョンごとの各 AWS アカウントのすべての DB インスタンスに対して一意にする必要があります。 Aurora リードレプリカ DB クラスターは出典 DB インスタンスのスナップショットから作成されるため、Aurora リードレプリカのマスターユーザー名およびマスターパスワードは出典 DB インスタンスのマスターユーザー名およびマスターパスワードと同じになります。

オプション	説明
仮想プライベートクラウド (VPC)	DB クラスターをホストする VPC を選択します。[新しい VPC の作成] を選択し、Aurora で自動的に VPC を作成します。詳細については、「DB クラスターの前提条件」を参照してください。
DB サブネットグループ	DB クラスターで使用する DB サブネットグループを選択します。[新しい DB サブネットグループの作成] を選択し、Aurora で自動的に DB サブネットグループを作成します。詳細については、「DB クラスターの前提条件」を参照してください。
パブリックアクセシビリティ	DB クラスターにパブリック IP アドレスを指定するには [Yes] を選択します。それ以外の場合は [No] を選択します。DB クラスターのインスタンスでは、パブリック DB インスタンスとプライベート DB インスタンスの両方を混在させることができます。パブリックアクセスからインスタンスを隠す方法については、「VPC 内の DB クラスターをインターネットから隠す」を参照してください。
アベイラビリティーゾーン	特定のアベイラビリティーゾーンを指定するかどうかを指定します。利用可能ゾーンについての詳細は、リージョンとアベイラビリティーゾーン を参照してください。
VPC セキュリティグループ (ファイアウォール)	[Create new VPC security group] を選択し、Aurora で自動的に VPC セキュリティグループを作成します。または、[Select existing VPC security groups] を選択し、DB クラスターへのネットワークアクセスの保護用に 1 つ以上の VPC セキュリティグループを指定します。詳細については、「DB クラスターの前提条件」を参照してください。

オプション	説明
データベースポート	データベースのアクセスに使用するために、アプリケーションやユーティリティのポートを指定します。Aurora MySQL DB クラスターのデフォルトの MySQL ポートは 3306 になります。会社のファイアウォールでは、このポートへの接続がブロックされます。会社のファイアウォールがデフォルトのポートをブロックする場合は、新しい DB クラスター用に別のポートを選択します。
DB パラメータグループ	Aurora MySQL DB クラスターの DB パラメータグループを選択します。Aurora にはデフォルトの DB パラメータグループが用意されています。また、独自の DB パラメータグループを作成することもできます。DB パラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。
DB クラスターのパラメータグループ	Aurora MySQL DB クラスター用に、DB クラスターパラメータグループを選択します。Aurora にはデフォルトの DB クラスターパラメータグループが用意されています。また、独自の DB クラスターパラメータグループを作成することもできます。DB クラスターパラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。

オプション	説明
暗号化	新しい Aurora DB クラスターを暗号化しない場合は、[暗号化を無効にする] を選択します。[暗号を有効化] を選択すると、新しい Aurora DB クラスターが保管時に暗号化されます。[暗号を有効化] を選択する場合、KMS キーを AWS KMS key 値として選択する必要があります。 MySQL DB インスタンスが暗号化されていない場合は、暗号化キーを指定して保管時の DB クラスターを暗号化します。 MySQL DB インスタンスが暗号化されている場合は、暗号化キーを指定し、その指定された暗号化キーを使用して保管時の DB クラスターを暗号化します。MySQL DB インスタンスで使用される暗号化キー、または別のキーを指定できます。暗号化された MySQL DB インスタンスから 非暗号化の DB クラスターを作成することはできません。
優先度	DB クラスターのフェイルオーバー優先度を選択します。値を選択しない場合、デフォルト値は tier-1 になります。この優先度により、プライマリインスタンスの障害からの復旧時に、Aurora レプリカを昇格する順序が決まります。詳細については、「Aurora DB クラスターの耐障害性」を参照してください。
バックアップの保存期間	Aurora がデータベースのバックアップコピーを保持する期間 (1〜35 日) を選択します。バックアップコピーは、2 番目のデータベースに対するポイントインタイム復元 (PITR) で使用できます。

オプション	説明
拡張モニタリング	DB クラスターが実行されているオペレーティングシステムに対してリアルタイムでのメトリクスの収集を有効にするには、[拡張モニタリングを有効にする] を選択します。詳細については、「拡張モニタリングを使用した OS メトリクスのモニタリング」を参照してください。
モニタリングロール	[拡張モニタリング] が [拡張モニタリングの有効化] に設定されている場合にのみ使用できます。ユーザーに代わって Amazon CloudWatch Logs と通信することを Aurora に許可するために作成した IAM ロールを選択するか、[デフォルト] を選択して `rds-monitoring-role` という名前のロールを Aurora で自動的に作成します。詳細については、「拡張モニタリングを使用した OS メトリクスのモニタリング」を参照してください。
詳細度	[拡張モニタリング] が [拡張モニタリングの有効化] に設定されている場合にのみ使用できます。DB クラスターのメトリクスを収集する間隔を秒単位で設定します。
マイナーバージョン自動アップグレード	この設定は Aurora MySQL DB クラスターには適用されません。 Aurora MySQL のエンジンに関する更新の詳細については、「Amazon Aurora MySQL のデータベースエンジンの更新」を参照してください。
メンテナンスウィンドウ	[ウィンドウの選択] を選択し、システムメンテナンスが実行される期間を週単位で指定します。または、[指定なし] を選択し、Aurora によって期間をランダムに割り当てます。

6. [Create read replica] を選択します。

AWS CLI

ソース RDS for MySQL DB インスタンスから Aurora リードレプリカを作成するには、AWS CLI コマンドの `create-db-cluster` と `create-db-instance` を使用して、新しい Aurora MySQL DB クラスターを作成します。`create-db-cluster` コマンドを呼び出すときは、`--replication-source-identifier` パラメータを含めて、出典 MySQL DB インスタンスの Amazon リソースネーム (ARN) を指定します。Amazon RDS ARN の詳細については、「Amazon Relational Database Service (Amazon RDS)」を参照してください。

出典 MySQL DB インスタンスと同じマスターユーザーネーム、マスターパスワード、またはデータベース名を Aurora リードレプリカで指定しないでください。

Linux、macOS、Unix の場合:

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine
 aurora \
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 \
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:primary-
mysql-instance
```

Windows の場合:

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine
 aurora ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 ^
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:primary-
mysql-instance
```

コンソールを使用して Aurora リードレプリカを作成すると、DB クラスターの Aurora リードレプリカのプライマリインスタンスが Aurora によって自動的に作成されます。AWS CLI を使用して Aurora リードレプリカを作成する場合、使用する DB クラスターのプライマリインスタンスを明示的に作成する必要があります。プライマリ インスタンスは、DB クラスターで作成される初期の DB インスタンスです。

以下のパラメータを指定して `create-db-instance` AWS CLI コマンドを使用することで、DB クラスターにプライマリインスタンスを作成できます。

- `--db-cluster-identifier`

  DB クラスターの名前。

- `--db-instance-class`

  プライマリインスタンスに使用するための DB インスタンス名。

- `--db-instance-identifier`

  プライマリインスタンスの名前。

- `--engine aurora`

この例では、*myinstanceclass* で指定される DB インスタンスクラスを使用して、*myreadreplicacluster* という名前の DB クラスターに *myreadreplicainstance* という名前のプライマリインスタンスを作成します。

Example

Linux、macOS、Unix の場合:

```
aws rds create-db-instance \
    --db-cluster-identifier myreadreplicacluster \
    --db-instance-class myinstanceclass \
    --db-instance-identifier myreadreplicainstance \
    --engine aurora
```

Windows の場合:

```
aws rds create-db-instance ^
    --db-cluster-identifier myreadreplicacluster ^
    --db-instance-class myinstanceclass ^
    --db-instance-identifier myreadreplicainstance ^
    --engine aurora
```

RDS API

ソース RDS for MySQL DB インスタンスから Aurora リードレプリカを作成するには、CreateDBCluster と CreateDBInstance Amazon RDS API コマンドを使用して新しい Aurora DB クラスターとプライマリインスタンスを作成します。ソース RDS for MySQL DB インスタンスと同じマスターユーザーネーム、マスターパスワード、またはデータベース名と同じユーザーネーム、マスターパスワード、またはデータベース名を Aurora リードレプリカに指定しないでください。

以下のパラメータを指定して CreateDBCluster Amazon RDS API コマンドを使用することで、RDS for MySQL DB インスタンスから Aurora リードレプリカに新しい Aurora DB クラスターを作成できます。

- DBClusterIdentifier

  作成する DB クラスターの名前。

- DBSubnetGroupName

  この DB クラスターに関連付ける DB サブネットグループの名前。

- Engine=aurora
- KmsKeyId

  MySQL DB インスタンスが暗号化されているかどうかによって、オプションで DB クラスターを暗号化する AWS KMS key。

  - MySQL DB インスタンスが暗号化されていない場合は、暗号化キーを指定して保管時の DB クラスターを暗号化します。これを実行しない場合、保管時の DB クラスターはデフォルトでアカウントの暗号化キーを使用して暗号化されます。
  - MySQL DB インスタンスが暗号化されている場合は、暗号化キーを指定し、その指定された暗号化キーを使用して保管時の DB クラスターを暗号化します。これを実行しない場合、保管時の DB クラスターは MySQL DB インスタンスの暗号化キーを使用して暗号化されます。

  > **Note**
  >
  > 暗号化された MySQL DB インスタンスから 非暗号化の DB クラスターを作成することはできません。

- ReplicationSourceIdentifier

  送信元の MySQL DB インスタンスの Amazon リソースネーム (ARN)。Amazon RDS ARN の詳細については、「Amazon Relational Database Service (Amazon RDS)」を参照してください。

- VpcSecurityGroupIds

  この DB クラスターに関連付ける EC2 VPC セキュリティグループのリスト。

この例では、ARN が *mysqlprimaryARN* に設定された元の MySQL DB インスタンスから、*mysubnetgroup* という名前の DB サブネットグループと *mysecuritygroup* という名前の

VPC セキュリティグループに関連付けられる *myreadreplicacluster* という名前の DB クラスターを作成します。

Example

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CreateDBCluster
    &DBClusterIdentifier=myreadreplicacluster
    &DBSubnetGroupName=mysubnetgroup
    &Engine=aurora
    &ReplicationSourceIdentifier=mysqlprimaryARN
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &Version=2014-10-31
    &VpcSecurityGroupIds=mysecuritygroup
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20150927/us-east-1/rds/aws4_request
    &X-Amz-Date=20150927T164851Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=6a8f4bd6a98f649c75ea04a6b3929ecc75ac09739588391cd7250f5280e716db
```

コンソールを使用して Aurora リードレプリカを作成すると、DB クラスターの Aurora リードレプリカのプライマリインスタンスが Aurora によって自動的に作成されます。AWS CLI を使用して Aurora リードレプリカを作成する場合、使用する DB クラスターのプライマリインスタンスを明示的に作成する必要があります。プライマリ インスタンスは、DB クラスターで作成される初期の DB インスタンスです。

以下のパラメータを指定して `CreateDBInstance` Amazon RDS API コマンドを使用することで、DB クラスターにプライマリインスタンスを作成できます。

- `DBClusterIdentifier`

  DB クラスターの名前。

- `DBInstanceClass`

  プライマリインスタンスに使用するための DB インスタンス名。

- `DBInstanceIdentifier`

  プライマリインスタンスの名前。

- `Engine=aurora`

この例では、*myinstanceclass* で指定される DB インスタンスクラスを使用して、*myreadreplicacluster* という名前の DB クラスターに *myreadreplicainstance* という名前のプライマリインスタンスを作成します。

Example

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CreateDBInstance
    &DBClusterIdentifier=myreadreplicacluster
    &DBInstanceClass=myinstanceclass
    &DBInstanceIdentifier=myreadreplicainstance
    &Engine=aurora
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &Version=2014-09-01
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20140424/us-east-1/rds/aws4_request
    &X-Amz-Date=20140424T194844Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=bee4aabc750bf7dad0cd9e22b952bd6089d91e2a16592c2293e532eeaab8bc77
```

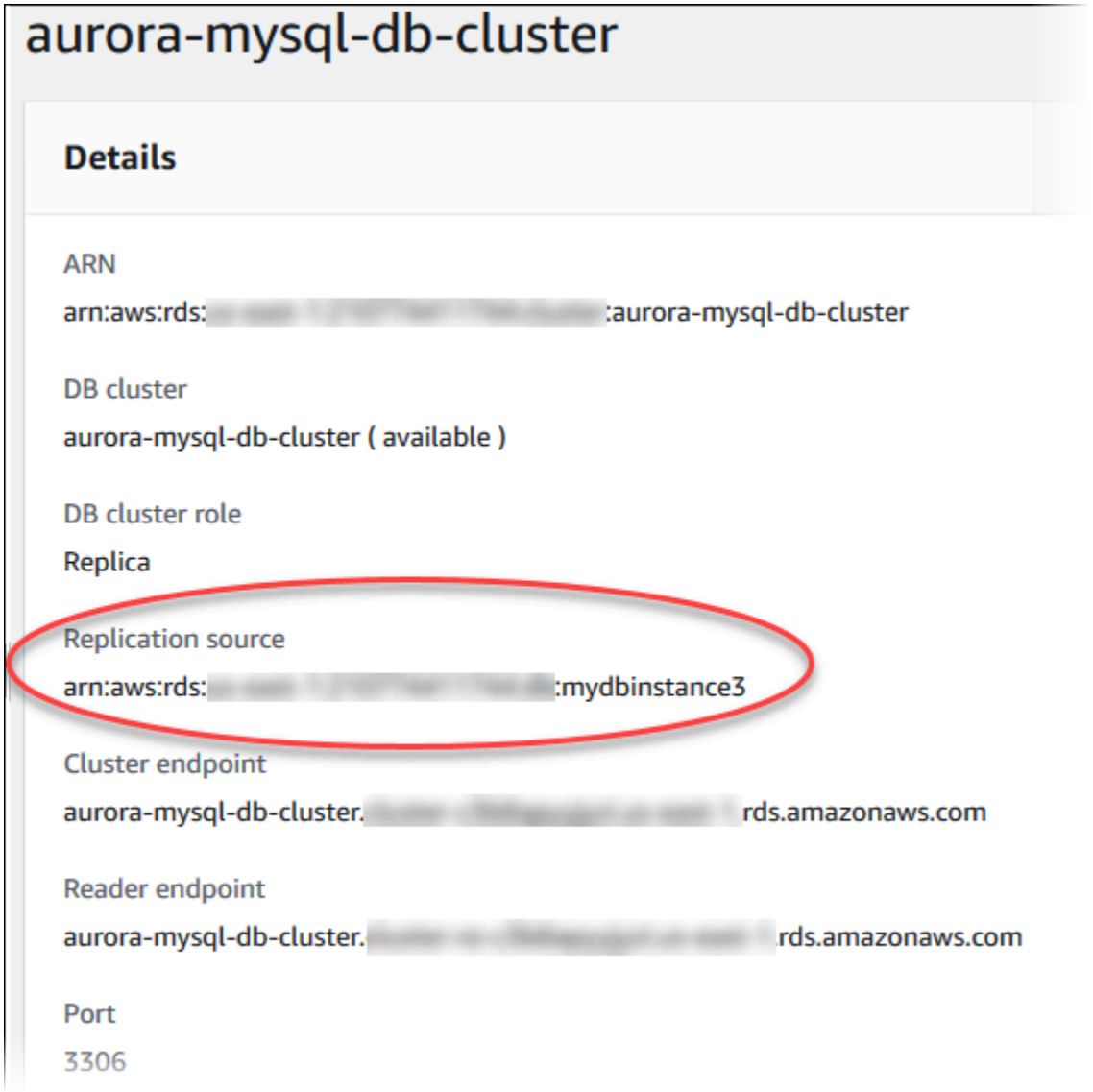
Aurora リードレプリカの表示

AWS Management Console または AWS CLI を使用すると、Aurora MySQL DB クラスターでの、MySQL から Aurora MySQL へのレプリケーション関係を確認できます。

コンソール

Aurora リードレプリカのプライマリ MySQL DB インスタンスを表示するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. Aurora リードレプリカの DB クラスターを選択して、その詳細を表示します。プライマリの MySQL DB インスタンスの情報は、[レプリケーション出典] フィールドに表示されます。



AWS CLI

AWS CLI により、Aurora MySQL DB クラスターでの Aurora MySQL レプリケーションと MySQL の関係性を確認するには、`describe-db-clusters` および `describe-db-instances` コマンドを使用します。

どの MySQL DB インスタンスがプライマリかを判別するには、`describe-db-clusters` を使用して、Aurora リードレプリカのクラスター識別子を `--db-cluster-identifier` オプションに指定します。レプリケーションのプライマリである DB インスタンスの ARN については、出力の `ReplicationSourceIdentifier` 要素を参照してください。

どの DB クラスターが Aurora リードレプリカであるかを判別するには、describe-db-instances を使用して、MySQL DB インスタンスのインスタンス識別子を --db-instance-identifier オプションに指定します。Aurora リードレプリカの DB クラスター識別子については、出力の ReadReplicaDBClusterIdentifiers 要素を参照してください。

Example

Linux、macOS、Unix の場合:

```
aws rds describe-db-clusters \
    --db-cluster-identifier myreadreplicacluster
```

```
aws rds describe-db-instances \
    --db-instance-identifier mysqlprimary
```

Windows の場合:

```
aws rds describe-db-clusters ^
    --db-cluster-identifier myreadreplicacluster
```

```
aws rds describe-db-instances ^
    --db-instance-identifier mysqlprimary
```

Aurora リードレプリカの昇格

移行が完了したら、AWS Management Console または AWS CLI を使って、Aurora リードレプリカをスタンドアロンの DB クラスターに昇格することができます。

次に、Aurora リードレプリカのエンドポイントにクライアントアプリケーションを誘導できます。Aurora エンドポイントの詳細については、「Amazon Aurora 接続管理」を参照してください。昇格はすばやく完了し、昇格中も Aurora リードレプリカに対する読み取り/書き込みを行うことができます。ただし昇格中に、プライマリ MySQL DB インスタンスを削除したり、DB インスタンスと Aurora リードレプリカのリンクを解除する操作は行うことができません。

Aurora リードレプリカを昇格する前に、出典 MySQL DB インスタンスに対するトランザクションの書き込みをすべて停止し、Aurora リードレプリカのレプリカラグが 0 になるまで待ちます。Aurora リードレプリカのレプリカラグを確認するには、SHOW SLAVE STATUS (Aurora MySQL バージョン 2) または SHOW REPLICA STATUS (Aurora MySQL バージョン 3) コマンドを Aurora リードレプリカに対して呼び出します。[マスターから数秒遅れ] 値をチェックします。

プライマリへの書き込みトランザクションが停止し、レプリカラグが 0 になったら、Aurora リードレプリカへの書き込みがスタートできるようになります。それより前に Aurora リードレプリカへの書き込みを行い、MySQL プライマリでも変更されているテーブルを変更した場合、Aurora へのレプリケーションが失われるおそれがあります。その場合は、Aurora リードレプリカを削除して再作成する必要があります。

コンソール

Aurora リードレプリカを Aurora DB クラスターに昇格させるには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. Aurora リードレプリカの DB クラスターを選択します。
4. [アクション] で、[Promote (昇格)] を選択します。
5. [リードレプリカの昇格] を選択します。

昇格したら、以下の手順を実行して昇格が完了したことを確認します。

Aurora リードレプリカが昇格したことを確認するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインの [Events] (イベント) を選択します。
3. [イベント] ページで、昇格したクラスターの `Promoted Read Replica cluster to a stand-alone database cluster` イベントがあることを確認します。

昇格が完了したら、プライマリ MySQL DB インスタンスと Aurora リードレプリカのリンクは解除され、DB インスタンスは必要に応じて安全に削除できるようになります。

AWS CLI

Aurora リードレプリカをスタンドアロン DB クラスターに昇格させるには、AWS CLI の `promote-read-replica-db-cluster` コマンドを使用します。

Example

Linux、macOS、Unix の場合:

```
aws rds promote-read-replica-db-cluster \
    --db-cluster-identifier myreadreplicacluster
```

Windows の場合:

```
aws rds promote-read-replica-db-cluster ^
    --db-cluster-identifier myreadreplicacluster
```

Amazon Aurora MySQL の管理

以下のセクションでは、Amazon Aurora MySQL DB クラスターの管理について説明します。

トピック

Amazon Aurora MySQL のパフォーマンスとスケーリングの管理

Aurora MySQL DB インスタンスのスケーリング

Aurora MySQL DB インスタンスは、インスタンススケーリングと読み取りスケーリングの 2 つの方法でスケールできます。読み取りスケーリングの詳細については、「読み取りのスケーリング」を参照してください。

DB クラスター内の各 DB インスタンスの DB インスタンスクラスを変更することで、Aurora MySQL DB クラスターをスケーリングできます。Aurora MySQL は、Aurora 用に最適化された複数の DB インスタンスクラスをサポートしています サイズが 40 TB より大きい Aurora クラスターには、db.t2 または db.t3 インスタンスクラスを使用しないでください。Aurora MySQL でサポートされている DB インスタンスクラスの詳細な仕様については、「Aurora DB インスタンスクラス」を参照してください。

> **Note**
>
> T DB インスタンスクラスは、開発サーバーおよびテストサーバー、またはその他の本稼働以外のサーバーにのみ使用することをお勧めします。T インスタンスクラスの詳細については、「開発やテストのための T インスタンスクラスの使用」を参照してください。

Aurora MySQL DB インスタンスへの最大接続数

Aurora MySQL DB インスタンスへの許可されている接続の最大数は、DB インスタンスのインスタンスレベルパラメータグループの `max_connections` パラメータによって決まります。

次の表は、Aurora MySQL で使用できる DB インスタンスクラスごとの max_connections のデフォルト値です。Aurora MySQL DB インスタンスへの接続の最大数を増やすには、このインスタンスをメモリ量のより多い DB インスタンスクラスにスケールするか、インスタンスの DB パラメータグループの max_connections パラメータの値を最大 16,000 に設定できます。

> **Tip**
>
> アプリケーションが頻繁に接続を開いたり閉じたりする場合や、長時間の接続を多数開いたままにする場合は、Amazon RDS Proxy の使用を推奨します。RDS Proxy は、接続プーリングを使用してデータベース接続を安全かつ効率的に共有する、フルマネージドの高可用性データベースプロキシです。RDS Proxy の詳細については、Amazon RDS Proxy for Aurora の使用 を参照してください。

Aurora Serverless v2 インスタンスによるこのパラメータの処理方法については、「Aurora Serverless v2 の最大接続数」を参照してください。

インスタンスクラス	max_connections のデフォルト値
db.t2.small	45
db.t2.medium	90
db.t3.small	45
db.t3.medium	90
db.t3.large	135
db.t4g.medium	90
db.t4g.large	135
db.r3.large	1,000
db.r3.xlarge	2000

インスタンスクラス	max_connections のデフォルト値		
db.r3.2xlarge	3000		
db.r3.4xlarge	4000		
db.r3.8xlarge	5000		
db.r4.large	1,000		
db.r4.xlarge	2000		
db.r4.2xlarge	3000		
db.r4.4xlarge	4000		
db.r4.8xlarge	5000		
db.r4.16xlarge	6000		
db.r5.large	1,000		
db.r5.xlarge	2000		
db.r5.2xlarge	3000		
db.r5.4xlarge	4000		
db.r5.8xlarge	5000		
db.r5.12xlarge	6000		
db.r5.16xlarge	6000		
db.r5.24xlarge	7000		
db.r6g.large	1000		
db.r6g.xlarge	2000		

インスタンスクラス	max_connections のデフォルト値		
db.r6g.2xlarge	3000		
db.r6g.4xlarge	4000		
db.r6g.8xlarge	5000		
db.r6g.12xlarge	6000		
db.r6g.16xlarge	6000		
db.r6i.large	1000		
db.r6i.xlarge	2000		
db.r6i.2xlarge	3000		
db.r6i.4xlarge	4000		
db.r6i.8xlarge	5000		
db.r6i.12xlarge	6000		
db.r6i.16xlarge	6000		
db.r6i.24xlarge	7000		
db.r6i.32xlarge	7000		
db.r7g.large	1000		
db.r7g.xlarge	2000		
db.r7g.2xlarge	3000		
db.r7g.4xlarge	4000		
db.r7g.8xlarge	5000		

インスタンスクラス	max_connections のデフォルト値		
db.r7g.12xlarge	6000		
db.r7g.16xlarge	6000		
db.x2g.large	2000		
db.x2g.xlarge	3000		
db.x2g.2xlarge	4000		
db.x2g.4xlarge	5000		
db.x2g.8xlarge	6000		
db.x2g.12xlarge	7000		
db.x2g.16xlarge	7000		

接続制限のデフォルトをカスタマイズする新しいパラメータグループを作成すると、DBInstanceClassMemory 値に基づく式を使用してデフォルトの接続制限が取得されます。前の表で示されているように、式は、メモリが段階的により大きな R3、R4、R5 インスタンスへと倍増すると 1000 ごとに増える接続最大数を、また T2 インスタンス、および T3 インスタンスの異なるメモリサイズでは 45 ごとに増える接続最大数を生成します。

DBInstanceClassMemory を計算する方法の詳細については、「DB パラメータの指定」を参照してください。

Aurora MySQL と RDS for MySQL DB インスタンスには、異なる量のメモリオーバーヘッドがあります。したがって、同じインスタンスクラスを使用する RDS for MySQL DB インスタンスと Aurora MySQL インスタンスでは、max_connections 値が異なる場合があります。テーブルの値は Aurora MySQL DB インスタンスにのみ適用されます。

> **Note**
>
> T2 インスタンス、および T3 インスタンスの接続制限がかなり低いのは、Aurora では、これらのインスタンスが本番稼働のワークロードのためではなく、開発やテストシナリオのみを目的としているためです。

デフォルトの接続制限は、バッファプールやクエリのキャッシュといった多くのメモリを消費する他の処理のデフォルト値を使用するシステムに合わせて調整されています。クラスターのこれらの他の設定を変更する場合は、DB インスタンスで使用可能なメモリの増減に応じて接続制限を調整することを検討してください。

Aurora MySQL 用の一時ストレージの制限

Aurora MySQL は、Aurora ストレージサブシステムにテーブルとインデックスを格納します。Aurora MySQL は、非永続的な一時ファイルや非 InnoDB 一時テーブル用に、別個の一時ストレージまたはローカルストレージを使用します。ローカルストレージには、クエリ処理中の大きなデータセットのソートや、インデックスの作成オペレーションなどの目的に使用するファイルも含まれます。InnoDB 一時テーブルは含まれません。

Aurora MySQL バージョン 3 の一時テーブルの詳細については、「Aurora MySQL バージョン 3 での新しい一時テーブルの動作」を参照してください。バージョン 2 の一時テーブルの詳細については、「Aurora MySQL バージョン 2 での一時テーブルスペースの動作」を参照してください。

これらのボリュームのデータと一時ファイルは、DB インスタンスの起動時と停止時、およびホストの交換時に失われます。

これらのローカルストレージボリュームは、Amazon Elastic Block Store (EBS) によってバックアップされ、より大きな DB インスタンスクラスを使用することで拡張できます。ストレージの詳細については、「Amazon Aurora ストレージと信頼性」を参照してください。

ローカルストレージは、`LOAD DATA FROM S3` や `LOAD XML FROM S3` を使用して Amazon S3 からデータをインポートする場合や、SELECT INTO OUTFILE S3 を使用して S3 にデータをエクスポートする場合にも使用します。S3 からのインポートと S3 へのエクスポートの詳細については、以下を参照してください。

- Amazon S3 バケットのテキストファイルから Amazon Aurora MySQL DB クラスターへのデータのロード

- Amazon Aurora MySQL DB クラスターから Amazon S3 バケット内のテキストファイルへのデータの保存

Aurora MySQL は、ほとんどの Aurora MySQL DB インスタンスクラス (db.t2、db.t3、db.t4g などのバーストパフォーマンスインスタンスクラスタイプは除く) のエラーログ、一般ログ、スロークエリログ、監査ログに別個の永続ストレージを使用します。このボリュームのデータは、DB インスタンスの起動時や停止時、ホストの交換時に保持されます。

また、この永続的ストレージボリュームは Amazon EBS-backed であり、DB インスタンスクラスに応じた固定サイズを持ちます。より大きな DB インスタンスクラスを使用して拡張することはできません。

次の表は、Aurora MySQL DB インスタンスクラス別に使用可能な一時ストレージと永続的ストレージの最大量を示しています。Aurora の DB インスタンスクラスサポートの詳細については、「Aurora DB インスタンスクラス」を参照してください。

DB インスタンスクラス	使用可能な一時ストレージ/ローカルストレージの最大量 (GiB)	ログファイルに使用可能な追加のストレージの最大量 (GiB)
db.x2g.16xlarge	1280	500
db.x2g.12xlarge	960	500
db.x2g.8xlarge	640	500
db.x2g.4xlarge	320	500
db.x2g.2xlarge	160	60
db.x2g.xlarge	80	60
db.x2g.large	40	60
db.r7g.16xlarge	1280	500
db.r7g.12xlarge	960	500
db.r7g.8xlarge	640	500

DB インスタンスクラス	使用可能な一時ストレージ/ローカルストレージの最大量 (GiB)	ログファイルに使用可能な追加のストレージの最大量 (GiB)
db.r7g.4xlarge	320	500
db.r7g.2xlarge	160	60
db.r7g.xlarge	80	60
db.r7g.large	32	60
db.r6i.32xlarge	2560	500
db.r6i.24xlarge	1920	500
db.r6i.16xlarge	1280	500
db.r6i.12xlarge	960	500
db.r6i.8xlarge	640	500
db.r6i.4xlarge	320	500
db.r6i.2xlarge	160	60
db.r6i.xlarge	80	60
db.r6i.large	32	60
db.r6g.16xlarge	1280	500
db.r6g.12xlarge	960	500
db.r6g.8xlarge	640	500
db.r6g.4xlarge	320	500
db.r6g.2xlarge	160	60
db.r6g.xlarge	80	60

DB インスタンスクラス	使用可能な一時ストレージ/ローカルストレージの最大量 (GiB)	ログファイルに使用可能な追加のストレージの最大量 (GiB)
db.r6g.large	32	60
db.r5.24xlarge	1920	500
db.r5.16xlarge	1280	500
db.r5.12xlarge	960	500
db.r5.8xlarge	640	500
db.r5.4xlarge	320	500
db.r5.2xlarge	160	60
db.r5.xlarge	80	60
db.r5.large	32	60
db.r4.16xlarge	1280	500
db.r4.8xlarge	640	500
db.r4.4xlarge	320	500
db.r4.2xlarge	160	60
db.r4.xlarge	80	60
db.r4.large	32	60
db.t4g.large	32	–
db.t4g.medium	32	–
db.t3.large	32	–
db.t3.medium	32	–

DB インスタンスクラス	使用可能な一時ストレージ/ローカルストレージの最大量 (GiB)	ログファイルに使用可能な追加のストレージの最大量 (GiB)
db.t3.small	32	–
db.t2.medium	32	–
db.t2.small	32	–

Important

これらの値は、各 DB インスタンスの理論上の最大空きストレージ量を表します。実際に使用可能なローカルストレージは、これより小さい場合があります。Aurora は、管理プロセスにローカルストレージを使用します。DB インスタンスでは、データをロードする前でもローカルストレージを使用します。特定の DB インスタンスで使用できる一時ストレージをモニタリングするには、 `FreeLocalStorage` CloudWatch メトリクスを使用できます。詳細については、Amazon Aurora の Amazon CloudWatch メトリクス を参照してください。現時点での空きストレージの量を確認できます。空きストレージの量を時間の経過に合わせてグラフ化することもできます。時間の経過に合わせて空きストレージをモニタリングすると、値が増加または減少しているかどうかを判断したり、最小値、最大値、または平均値を確認したりするのに役立ちます。
(これは Aurora Serverless v2 には適用されません。)

Aurora DB クラスターのバックトラック

Amazon Aurora MySQL 互換エディション では、バックアップからデータを復元しないで、DB クラスターを特定の時刻までバックトラックできます。

目次

- バックトラック対応クラスターのアップグレードに関する考慮事項
- バックトラックの設定
- バックトラックの実行
- バックトラックのモニタリング
- コンソールを使用したバックトラックイベントへのサブスクライブ
- 既存のバックトラックの取得
- DB クラスターのバックトラックの無効化

バックトラックの概要

バックトラックは、指定した時間まで DB クラスターを「巻き戻し」ます。バックトラックは、DB クラスターをバックアップして特定の時点の状態に復元する操作に代わるものではありません。ただし、バックトラックは、従来のバックアップと復元に比べて、以下の利点があります。

- 簡単にエラーを取り消すことができます。WHERE 句なしの DELETE などの破壊的なアクションを間違えて実行した場合、サービスの中断を最小限に抑えながら、破壊的なアクション以前の時点まで DB クラスターをバックトラックできます。
- DB クラスターのバックトラックは迅速に実行できます。DB クラスターを特定の時点の状態に復元するには、新しい DB クラスターを起動し、これに対してバックアップデータや DB クラスターのスナップショットから復元する必要があり、時間がかかります。DB クラスターのバックトラックでは、新しい DB クラスターを作成せずに、DB クラスターを数分で巻き戻します。
- 以前のデータの変更を調べることができます。DB クラスターを前後に繰り返しバックトラックして、特定のデータの変更がどの時点で発生したかを確認できます。例えば、DB クラスターを 3 時間前までバックトラックし、そこから 1 時間後まで戻すことができます。この場合、バックトラック時間は元の時間の 2 時間前となります。

> **Note**
>
> DB クラスターを特定の時点の状態に復元する方法については、「Aurora DB クラスターのバックアップと復元の概要」を参照してください。

バックトラックウィンドウ

バックトラックには、ターゲットバックトラックウィンドウと実際のバックトラックウィンドウがあります。

- ターゲットバックトラックウィンドウは、DB クラスターをバックトラックする所望時間です。ターゲットバックトラックウィンドウは、バックトラックを有効化するときに指定します。例えば、DB クラスターを 1 日バックトラックする場合は、ターゲットバックトラックウィンドウとして 24 時間を指定します。
- 実際のバックトラックウィンドウは、DB クラスターを実際にバックトラックできる時間であり、ターゲットバックトラックウィンドウより小さい値になることがあります。実際のバックトラックウィンドウは、ワークロードとストレージ (変更レコードと呼ばれるデータベースの変更に関する情報を保存) に基づきます。

バックトラックが有効になっている Aurora DB クラスターを更新すると、変更レコードが生成されます。Aurora は、ターゲットのバックトラックウィンドウの変更レコードを保持します。変更レコードの保存には利用料金 (時間単位) が発生します。DB クラスターのターゲットバックトラックウィンドウとワークロードの両方により、保存する変更レコード数が決まります。ワークロードは、特定の時間内に DB クラスターを変更する回数です。ワークロードが重い場合は、ワークロードが軽い場合と比べて、バックトラックウィンドウに保存される変更レコードが多くなります。

ターゲットバックトラックウィンドウは、DB クラスターをバックトラックできる最大の目標時間と考えることができます。通常、指定した最大時間までバックトラックできます。ただし、状況によっては、DB クラスターに保存できる変更レコードの数が少なくて、最大時間までバックトラックできない場合があります。この場合、実際のバックトラックウィンドウはターゲットより小さくなります。通常、DB クラスターのワークロードが非常に重い場合、実際のバックトラックウィンドウはターゲットより小さくなります。実際のバックトラックウィンドウがターゲットより小さい場合は、通知が送信されます。

DB クラスターのバックトラックが有効になっている場合、DB クラスターに保存されているテーブルを削除すると、そのテーブルは Aurora のバックトラック変更レコード内に保持されます。これにより、テーブルを削除する前の時点まで戻すことができます。バックトラックウィンドウ内のスペース不足でテーブルを保存できないと、テーブルは最終的にバックトラック変更レコードから削除される場合があります。

バックトラック時間

Aurora は、常に DB クラスターに即した時間までバックトラックします。これにより、トランザクションがコミットされないままバックトラックが完了するという事態が避けられます。バックトラッ

ク時間を指定すると、Aurora はそれに即した最も近い時間を自動的に選択します。このアプローチでは、最終的なバックトラック時間が、指定した時間と正確に一致しない場合があります。正確なバックトラック時間は、AWS CLI の describe-db-cluster-backtracks コマンドを使用して確認できます。詳細については、「既存のバックトラックの取得」を参照してください。

バックトラックの制限事項

バックトラックには以下の制限が適用されます。

- バックトラックは、バックトラック機能を有効にして作成した DB クラスターでのみ使用できます。DB クラスターを変更せずにバックトラック機能を有効にすることはできません。バックトラック機能は、新しい DB クラスターの作成時または DB クラスターのスナップショットの復元時に有効化できます。
- バックトトラックウィンドウの上限は 72 時間です。
- バックトラックは、DB クラスター全体に影響します。例えば、1 つのテーブルや 1 つのデータ更新に限定してバックトラックすることはできません。
- バックトラック対応クラスターからクロスリージョンリードレプリカを作成することはできませんが、クラスターでバイナリログ (binlog) レプリケーションを有効にすることはできます。バイナリログ記録が有効になっている DB クラスターをバックトラックしようとすると、バックトラックの強制を選択していない限り、通常はエラーが発生します。バックトラックを強制しようとすると、ダウンストリームのリードレプリカが壊れ、ブルー/グリーンデプロイなどの他のオペレーションが妨げられます。
- データベースのクローンを、その作成時より前の時点までバックトラックすることはできません。ただし、元のデータベースを使用すると、クローン作成時より前の時点までバックトラックできます。データベースのクローンの詳細については、「Amazon Aurora DB クラスターのボリュームのクローン作成」を参照してください。
- バックトラックに伴って DB インスタンスがわずかに中断します。バックトラックオペレーションをスタートする前にアプリケーションを停止または一時停止し、新しい読み取り/書き込みリクエストを阻止する必要があります。Aurora は、バックトラックオペレーション時に、データベースを一時停止し、開いている接続をすべて閉じて、コミットされていない読み取りや書き込みをすべて削除します。その後、バックトラックオペレーションが完了するまで待ちます。
- バックトラックをサポートしていない AWS リージョンでは、バックトラック対応クラスターのクロスリージョンスナップショットを復元できません。
- バックトラック対応クラスターで Aurora MySQL バージョン 2 からバージョン 3 へのインプレースアップグレードを実行する場合、アップグレードを実行する前の時点にバックトラックすることはできません。

リージョンとバージョンの可用性

バックトラックは Aurora PostgreSQL では使用できません。

Aurora MySQL によるバックトラックでサポートされているエンジンとリージョンは以下のとおりです。

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
米国東部 (オハイオ)	すべてのバージョン	すべてのバージョン
米国東部 (バージニア北部)	すべてのバージョン	すべてのバージョン
米国西部 (北カリフォルニア)	すべてのバージョン	すべてのバージョン
米国西部 (オレゴン)	すべてのバージョン	すべてのバージョン
アフリカ (ケープタウン)	–	–
アジアパシフィック (香港)	–	–
アジアパシフィック (ジャカルタ)	–	–
アジアパシフィック (メルボルン)	–	–
アジアパシフィック (ムンバイ)	すべてのバージョン	すべてのバージョン

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
アジアパシフィック (大阪)	すべてのバージョン	バージョン 2.07.3 以降
アジアパシフィック (ソウル)	すべてのバージョン	すべてのバージョン
アジアパシフィック (シンガポール)	すべてのバージョン	すべてのバージョン
アジアパシフィック (シドニー)	すべてのバージョン	すべてのバージョン
アジアパシフィック (東京)	すべてのバージョン	すべてのバージョン
カナダ (中部)	すべてのバージョン	すべてのバージョン
カナダ西部 (カルガリー)	–	–
中国 (北京)	–	–
中国 (寧夏)	–	–
欧州 (フランクフルト)	すべてのバージョン	すべてのバージョン
欧州 (アイルランド)	すべてのバージョン	すべてのバージョン
欧州 (ロンドン)	すべてのバージョン	すべてのバージョン
欧州 (ミラノ)	–	–

リージョン	Aurora MySQL バージョン 3	Aurora MySQL バージョン 2
欧州 (パリ)	すべてのバージョン	すべてのバージョン
欧州 (スペイン)	–	–
欧州 (ストックホルム)	–	–
欧州 (チューリッヒ)	–	–
イスラエル (テルアビブ)	–	–
中東 (バーレーン)	–	–
中東 (アラブ首長国連邦)	–	–
南米 (サンパウロ)	–	–
AWS GovCloud (米国東部)	–	–
AWS GovCloud (米国西部)	–	–

バックトラック対応クラスターのアップグレードに関する考慮事項

Aurora MySQL バージョン 3 のすべてのマイナーバージョンがバックトラックについてサポートされているため、バックトラック対応 DB クラスターを Aurora MySQL バージョン 2 からバージョン 3 にアップグレードできます。

バックトラックの設定

バックトラック機能を使用するには、バックトラックを有効にしてターゲットバックトラックウィンドウを指定する必要があります。それ以外の場合、バックトラックは無効です。

ターゲットのバックトラックウィンドウでは、バックトラックを使用してデータベースを巻き戻したい時間の長さを指定します。Aurora は、その時間枠をサポートするのに十分な変更レコードを保持しようと試みます。

コンソール

コンソールを使用して、新しい DB クラスターの作成時にバックトラックを設定できます。DB クラスターを変更して、バックトラック対応クラスターのバックトラックウィンドウを変更することもできます。バックトラックウィンドウを 0 に設定してクラスターのバックトラックを完全に無効にすると、そのクラスターでバックトラックを再度有効にすることはできません。

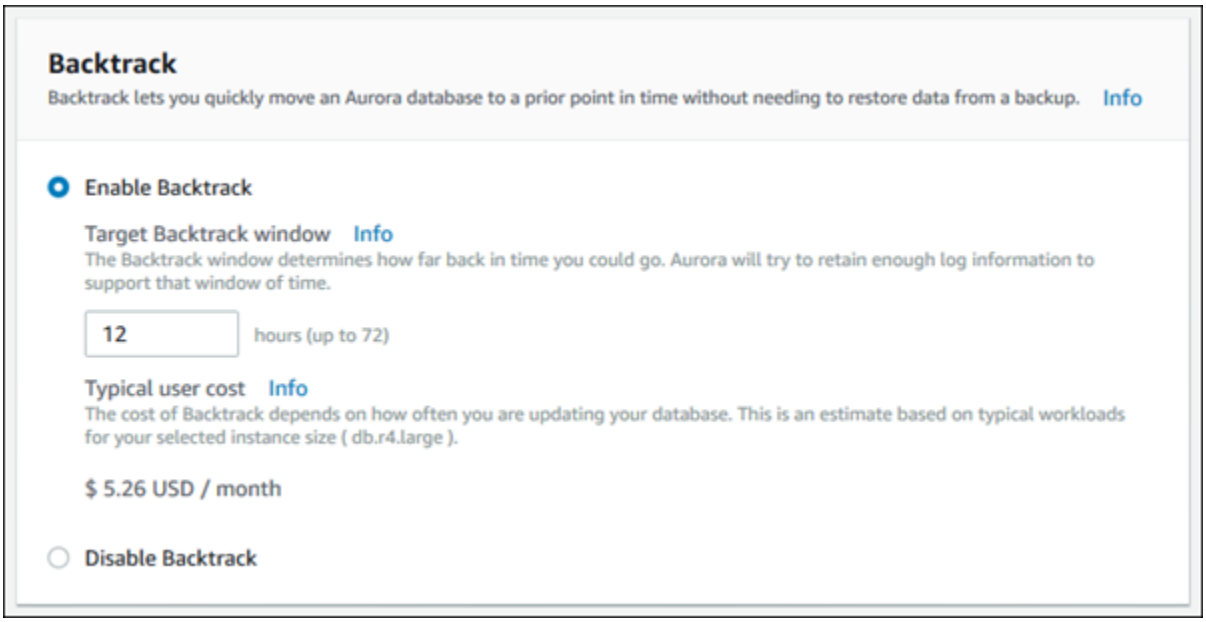
トピック

- DB クラスターの作成時にコンソールでバックトラックを設定する
- DB クラスターの変更時にコンソールでバックトラックを設定する

DB クラスターの作成時にコンソールでバックトラックを設定する

新しい Aurora MySQL DB クラスターの作成時にバックトラックを設定するには、[Enable Backtrack (バックトラックの有効化)] を選択し、[Target Backtrack window (ターゲットバックトラックウィンドウ)] 値としてゼロより大きい値を [Backtrack (バックトラック)] セクションに指定します。

DB クラスターを作成するには、「Amazon Aurora DB クラスターの作成」の手順に従います。次のイメージは [Backtrack (バックトラック)] セクションを示しています。



新しい DB クラスターを作成した時点では、Aurora には DB クラスターのワークロード用のデータがありません。したがって、新しい DB クラスターのコストを具体的に見積もることはできません。コンソールでは、代わりに一般的なワークロードに基づいて、指定したターゲットバックトラックウィンドウの一般的なユーザーコストを提示します。一般的なコストは、バックトラック機能のコストを見積もるための一般的な参考として提供されます。

> **Important**
>
> 実際のコストは、DB クラスターのワークロードに基づくため、一般的なコストとは一致しない場合があります。

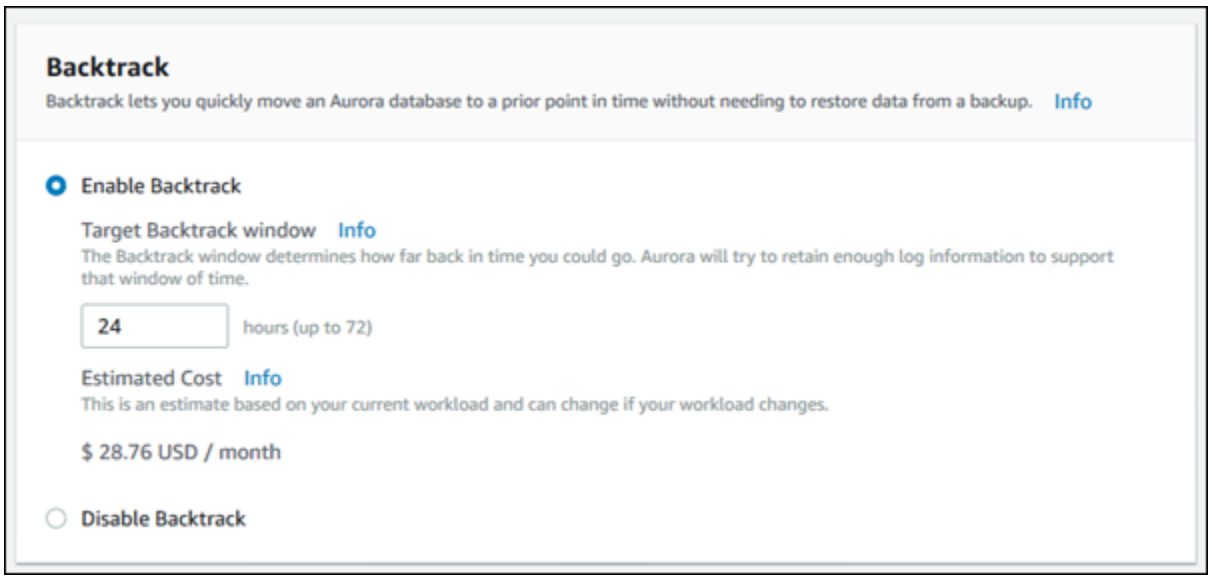
DB クラスターの変更時にコンソールでバックトラックを設定する

コンソールを使用して DB クラスターのバックトラックを変更できます。

> **Note**
>
> 現時点では、バックトラック機能が有効になっている DB クラスターに限り、バックトラックを変更できます。バックトラック機能を無効にして作成した DB クラスターや後でバックトラック機能を無効にした DB クラスターでは、[バックトラック] 機能が表示されません。

コンソールを使用して DB クラスターのバックトラックを変更するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. [データベース] をクリックします。
3. 変更するクラスターを選択し、[変更] を選択します。
4. [Target Backtrack window (ターゲットバックトラックウィンドウ)] で、バックトラックする所望時間を変更します。上限は 72 時間です。



コンソールは、DB クラスターの過去のワークロードに基づいて、指定した時間のコストを見積もります。

- DB クラスターでバックトラックが無効になっている場合は、Amazon CloudWatch の DB クラスターの `VolumeWriteIOPS` メトリクスに基づいてコストを見積もります。
- DB クラスターでバックトラックが以前に有効化されている場合は、Amazon CloudWatch の DB クラスターの `BacktrackChangeRecordsCreationRate` メトリクスに基づいてコストを見積もります。

5. Continue (続行) をクリックします。
6. [Scheduling of Modifications (変更の予定)] で、以下のいずれかを選択します。
   - Apply during the next scheduled maintenance window (次回予定のメンテナンスウィンドウで適用する) - 次回のメンテナンスウィンドウまで待ってから [Target Backtrack window (ターゲットバックトラックウィンドウ)] の変更を適用します。
   - Apply immediately (即時に適用する) - [Target Backtrack window (ターゲットバックトラックウィンドウ)] の変更をできるだけ早急に適用します。
7. [Modify cluster] (クラスターの変更) を選択します。

AWS CLI

AWS CLI の create-db-cluster コマンドを使用して新しい Aurora MySQL DB クラスターを作成する場合、バックトラックを設定するには、ゼロより大きい `--backtrack-window` 値を指定しま

す。--backtrack-window 値は、ターゲットバックトラックウィンドウを指定します。詳細については、「Amazon Aurora DB クラスターの作成」を参照してください。

以下の --backtrack-window CLI コマンドを使用して AWS 値を指定することもできます。

- modify-db-cluster
- restore-db-cluster-from-s3
- restore-db-cluster-from-snapshot
- restore-db-cluster-to-point-in-time

次の手順では、AWS CLI を使用して DB クラスターのターゲットバックトラックウィンドウを変更する方法を示します。

AWS CLI を使用して DB クラスターのターゲットバックトラックウィンドウを変更するには

- AWS CLI の modify-db-cluster コマンドを呼び出して以下の値を指定します。
  - --db-cluster-identifier - DB クラスターの名前。
  - --backtrack-window - DB クラスターをバックトラックする所望の最大秒数。

  次の例では、sample-cluster のターゲットバックトラックウィンドウを 1 日 (86,400 秒) に設定します。

  Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --backtrack-window 86400
```

  Windows の場合:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --backtrack-window 86400
```

> **Note**
>
> 現在、バックトラックを有効化できるのは、バックトラック機能を有効にして作成した DB クラスターに限ります。

RDS API

CreateDBCluster Amazon RDS API オペレーションを使用して新しい Aurora MySQL DB クラスターを作成する場合、バックトラックを設定するには、ゼロより大きい `BacktrackWindow` 値を指定します。`BacktrackWindow` 値は、`DBClusterIdentifier` 値で指定した DB クラスターのターゲットバックトラックウィンドウを指定します。詳細については、「Amazon Aurora DB クラスターの作成」を参照してください。

以下の API オペレーションを使用して `BacktrackWindow` 値を指定することもできます。

- ModifyDBCluster
- RestoreDBClusterFromS3
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime

> **Note**
>
> 現在、バックトラックを有効化できるのは、バックトラック機能を有効にして作成した DB クラスターに限ります。

バックトラックの実行

DB クラスターは、指定したバックトラックタイムスタンプまでバックトラックできます。バックトラックタイムスタンプが、最も早いバックトラック時間から現時点までの範囲内にあれば、DB クラスターはそのタイムスタンプまでバックトラックされます。

それ以外の場合は、通常、エラーが発生します。また、バイナリログが有効になっている DB クラスターをバックトラックしようとすると、通常、エラーが発生します。ただし、バックトラックの強制実行を選択した場合を除きます。バックトラックの強制実行は、バイナリログを使用する他のオペレーションに干渉する場合があります。

> **Important**
>
> バックトラックでは、それに伴う変更の binlog エントリが生成されません。DB クラスターでバイナリログを有効にしている場合、バックトラックは binlog 実装と互換性がない可能性があります。

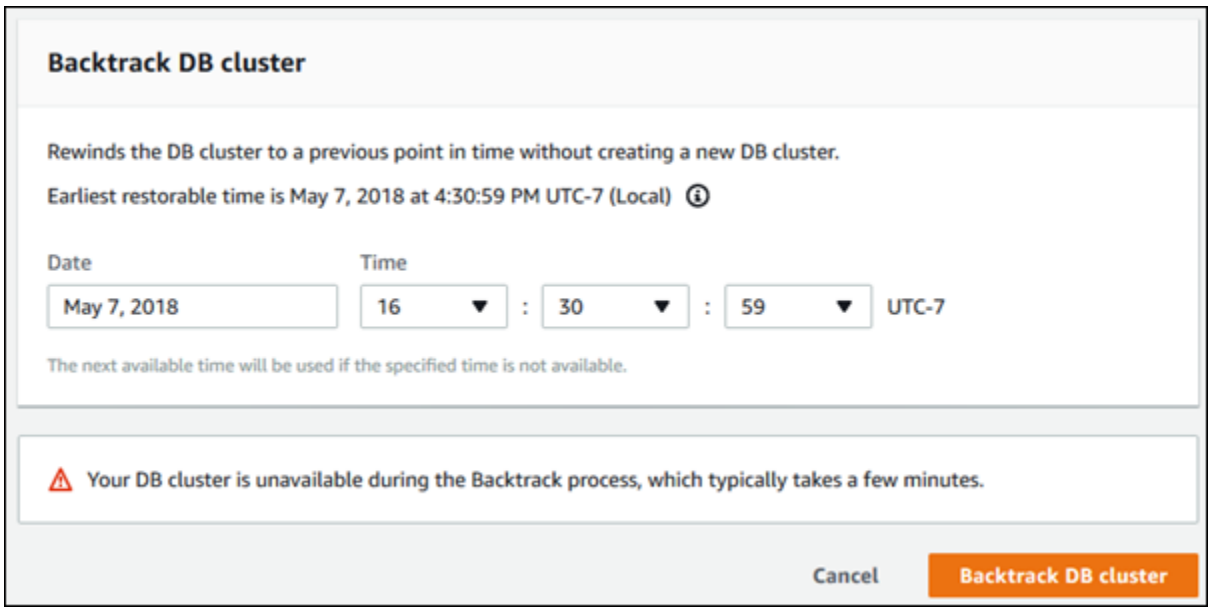
> **Note**
>
> データベースのクローンでは、クローンの作成日時より前の時点まで DB クラスターをバックトラックすることはできません。データベースのクローンの詳細については、「Amazon Aurora DB クラスターのボリュームのクローン作成」を参照してください。

コンソール

次の手順では、コンソールを使用して DB クラスターのバックトラックオペレーションを実行する方法を示します。

コンソールを使用してバックトラックオペレーションを実行するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[インスタンス] を選択します。
3. バックトラックする DB クラスターのプライマリインスタンスを選択します。
4. [アクション] で、[DB クラスターのバックトラック] を選択します。
5. [DB クラスターのバックトラック] ページで、DB クラスターのバックトラック先のバックトラックタイムスタンプを入力します。



6. [Backtrack DB cluster (DB クラスターのバックトラック)] を選択します。

AWS CLI

次の手順では、AWS CLI を使用して DB クラスターをバックトラックする方法を示します。

AWS CLI を使用して DB クラスターをバックトラックするには

- AWS CLI の backtrack-db-cluster コマンドを呼び出して以下の値を渡します。
  - `--db-cluster-identifier` - DB クラスターの名前。
  - `--backtrack-to` - DB クラスターをバックトラックする先のバックトラックタイムスタンプ (ISO 8601 形式で指定)。

  次の例では、DB クラスター `sample-cluster` を 2018 年 3 月 19 日午前 10 時までバックトラックします。

  Linux、macOS、Unix の場合:

```
aws rds backtrack-db-cluster \
    --db-cluster-identifier sample-cluster \
    --backtrack-to 2018-03-19T10:00:00+00:00
```

  Windows の場合:

```
aws rds backtrack-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --backtrack-to 2018-03-19T10:00:00+00:00
```

RDS API

Amazon RDS API を使用して DB クラスターをバックトラックするには、BacktrackDBCluster オペレーションを使用します。このオペレーションでは、`DBClusterIdentifier` 値で指定した DB クラスターを、指定した時間までバックトラックします。

バックトラックのモニタリング

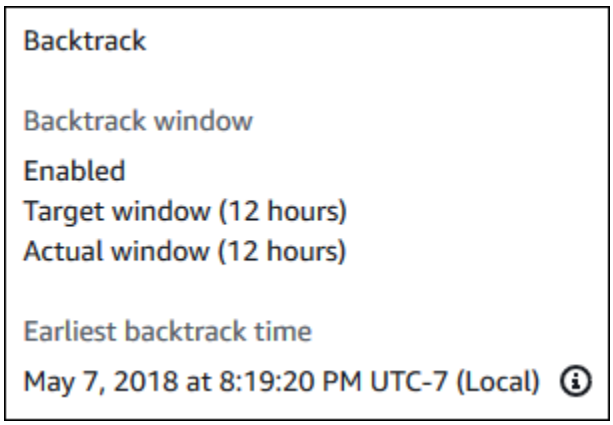
DB クラスターのバックトラック情報を表示し、バックトラックのメトリクスをモニタリングできます。

コンソール

コンソールを使用してバックトラック情報を表示し、バックトラックのメトリクスをモニタリングするには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. [データベース] をクリックします。
3. 情報を表示する DB クラスターの名前を選択します。

   バックトラック情報は [Backtrack (バックトラック)] セクションにあります。



   バックトラックが有効になっていると、以下の情報が表示されます。

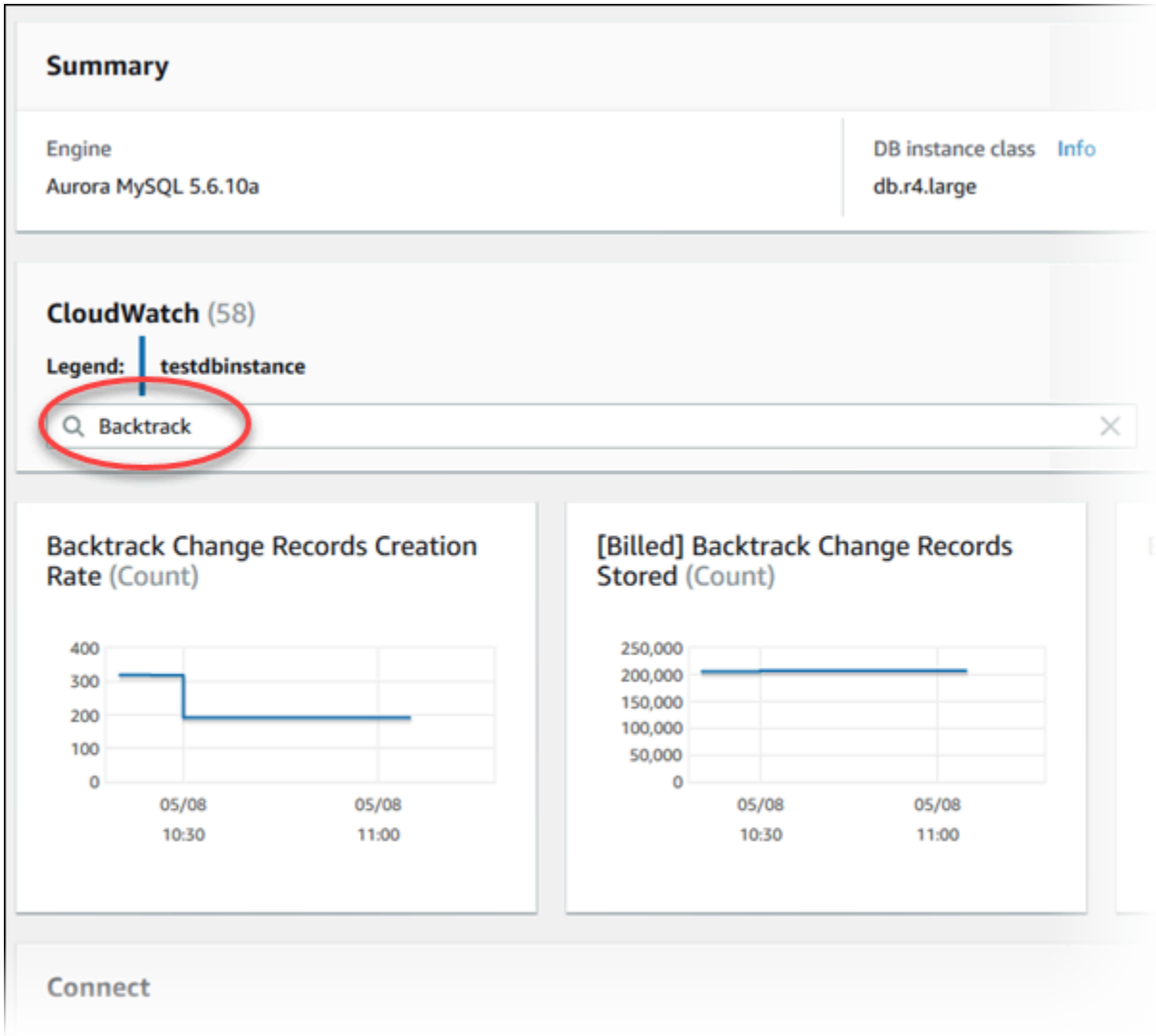
- Target window (ターゲットウィンドウ) - ターゲットバックトラックウィンドウとして現在指定されている時間。十分なストレージがある場合、ターゲットはバックトラックできる最大時間です。
- Actual window (実際のウィンドウ) - 実際にバックトラックできる時間であり、ターゲットバックトラックウィンドウより小さい値になることがあります。実際のバックトラックウィンドウは、ワークロードと、バックトラック変更レコードを保持できるストレージに基づきます。
- Earliest backtrack time (最も早いバックトラック時間) - DB クラスターの最も早い (可能な限り) バックトラック時間。表示された時間より前の時点まで DB クラスターをバックトラックすることはできません。

4. DB クラスターのバックトラックのメトリクスを表示するには、以下の操作を行います。

   a. ナビゲーションペインで、[インスタンス] を選択します。

   b. 詳細を表示する DB クラスターのプライマリインスタンス名を選択します。

   c. [CloudWatch] セクションで、[CloudWatch] ボックスに **`Backtrack`** と入力し、バックトラックのメトリクスのみを表示します。



以下のメトリクスが表示されます。

- バックトラック変更レコードの作成率 (カウント) - このメトリクスは、DB クラスターで 5 分間に作成されたバックトラック変更レコードの数を示します。このメトリクスを使用して、ターゲットバックトラックウィンドウのバックトラックコストを見積もることができます。
- [課金済み] 保存されたバックトラック変更レコード数 (カウント) - このメトリクスは、DB クラスターで実際に使用されたバックトラック変更レコードの数を示します。
- 実際のバックトラックウィンドウ (分数) - このメトリクスは、ターゲットバックトラックウィンドウと実際のバックトラックウィンドウの間に差異があるかどうかを示します。例えば、ターゲットバックトラックウィンドウが 2 時間 (120 分) で、このメトリクスで示された実際のバックトラックウィンドウが 100 分の場合、実際のバックトラックウィンドウはターゲットよりも小さくなります。

- バックトラックウィンドウのアラート (カウント) - このメトリクスは、特定の時間内に実際のバックトラックウィンドウがターゲットバックトラックウィンドウより小さくなった回数を示します。

> **Note**
>
> 以下のメトリクスは、現在の時刻より遅れる場合があります。
>
> - バックトラック変更レコードの作成率 (カウント)
> - [課金済み] 保存されたバックトラック変更レコード (カウント)

AWS CLI

次の手順では、AWS CLI を使用して DB クラスターのバックトラック情報を表示する方法を示します。

AWS CLI を使用して DB クラスターのバックトラック情報を表示するには

- AWS CLI の describe-db-clusters コマンドを呼び出して以下の値を渡します。
  - `--db-cluster-identifier` - DB クラスターの名前。

  次の例では、`sample-cluster` のバックトラック情報を一覧表示します。

  Linux、macOS、Unix の場合:

```
aws rds describe-db-clusters \
    --db-cluster-identifier sample-cluster
```

  Windows の場合:

```
aws rds describe-db-clusters ^
    --db-cluster-identifier sample-cluster
```

RDS API

Amazon RDS API を使用して DB クラスターのバックトラック情報を表示するには、DescribeDBClusters オペレーションを使用します。このオペレーションでは、DBClusterIdentifier 値で指定した DB クラスターのバックトラック情報を返します。

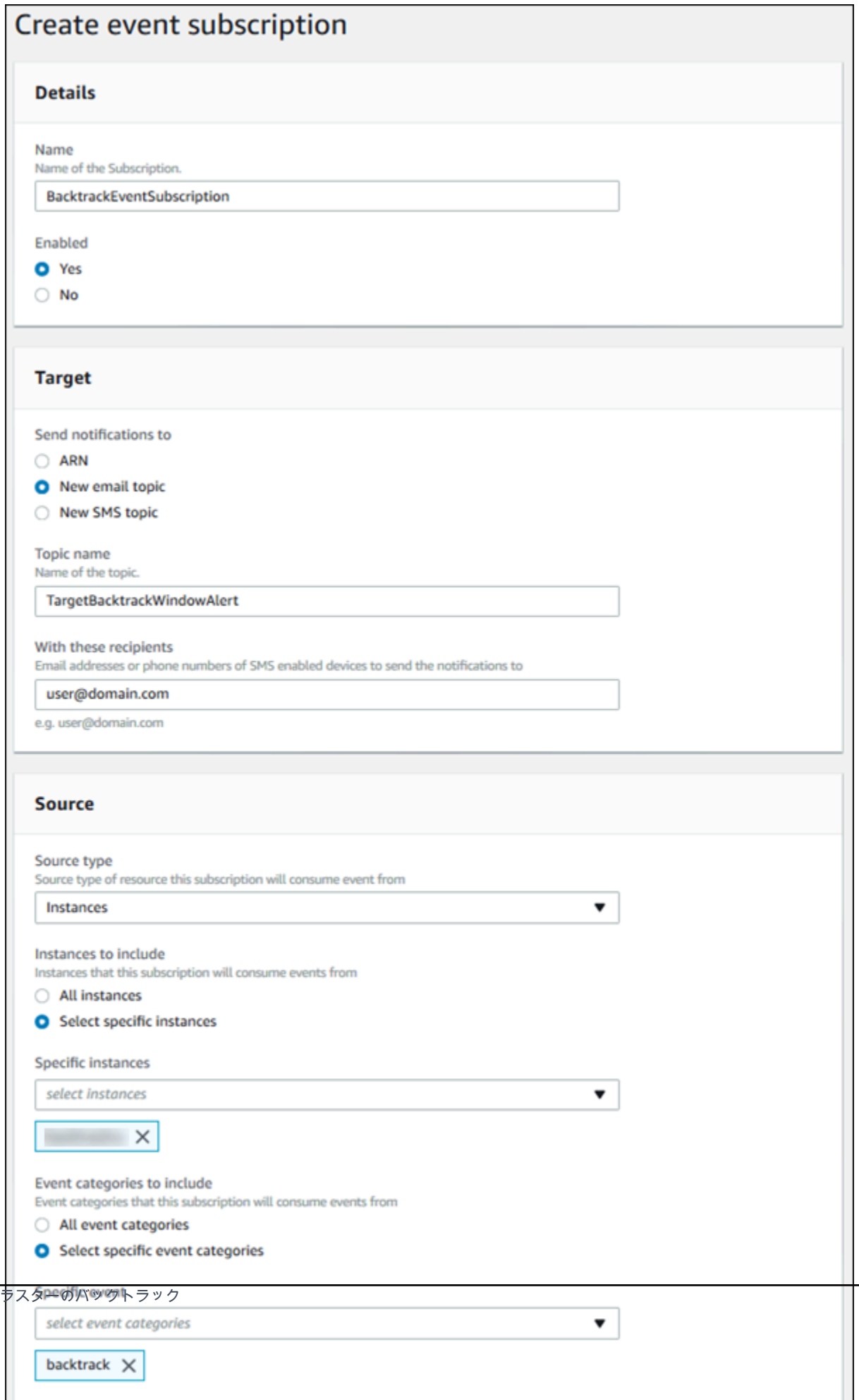
コンソールを使用したバックトラックイベントへのサブスクライブ

次の手順では、コンソールを使用してバックトラックイベントにサブスクライブする方法を示します。実際のバックトラックウィンドウがターゲットバックトラックウィンドウより小さい場合、このイベントから E メール通知またはテキスト通知が送信されます。

コンソールを使用してバックトラック情報を表示するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. [イベントサブスクリプション] をクリックします。
3. [イベントサブスクリプションを作成] をクリックします。
4. [名前] ボックスにイベントサブスクリプションの名前を入力し、[有効] で [はい] を必ず選択します。
5. [ターゲット] セクションで、[新しい E メールトピック] を選択します。
6. [トピック名] にトピックの名前を入力し、[これらの受取人を含みます] に通知を受け取る E メールアドレスまたは電話番号を入力します。
7. [出典] セクションで、[出典タイプ] として [インスタンス] を選択します。
8. [含めるインスタンス] で、[特定のインスタンスを選択] をクリックし、DB インスタンスを選択します。
9. [含めるイベントカテゴリ] で、[特定のイベントカテゴリを選択] をクリックし、[バックトラック] を選択します。

   ページは次のように表示されます。


Create event subscription
Details
Name
Name of the Subscription.
BacktrackEventSubscription
Enabled
Yes
No
Target
Send notifications to
ARN
New email topic
New SMS topic
Topic name
Name of the topic.
TargetBacktrackWindowAlert
With these recipients
Email addresses or phone numbers of SMS enabled devices to send the notifications to
user@domain.com
e.g. user@domain.com
Source
Source type
Source type of resource this subscription will consume event from
Instances
Instances to include
Instances that this subscription will consume events from
All instances
Select specific instances
Specific instances
select instances
Event categories to include
Event categories that this subscription will consume events from
All event categories
Select specific event categories
select event categories
backtrack

10. [作成] を選択します。

既存のバックトラックの取得

DB クラスターの既存のバックトラックに関する情報を取得できます。この情報には、バックトラック固有の識別子、出典から/ターゲットへのバックトラック日時、バックトラックのリクエスト日時、バックトラックの現在のステータスが含まれます。

> **Note**
>
> 現時点では、コンソールを使用して既存のバックトラックを取得することはできません。

AWS CLI

次の手順では、AWS CLI を使用して DB クラスターの既存のバックトラックを取得する方法を示します。

AWS CLI を使用して既存のバックトラックを取得するには

- AWS CLI の describe-db-cluster-backtracks コマンドを呼び出して以下の値を渡します。
  - `--db-cluster-identifier` - DB クラスターの名前。

  次の例では、`sample-cluster` の既存のバックトラックを取得します。

  Linux、macOS、Unix の場合:

```
aws rds describe-db-cluster-backtracks \
    --db-cluster-identifier sample-cluster
```

  Windows の場合:

```
aws rds describe-db-cluster-backtracks ^
    --db-cluster-identifier sample-cluster
```

RDS API

Amazon RDS API を使用して DB クラスターのバックトラックに関する情報を取得するには、DescribeDBClusterBacktracks オペレーションを使用します。このオペレーションでは、`DBClusterIdentifier` 値で指定した DB クラスターのバックトラックに関する情報を返します。

DB クラスターのバックトラックの無効化

DB クラスターのバックトラック機能を無効にすることができます。

コンソール

コンソールを使用して DB クラスターのバックトラックを無効にすることができます。クラスターのバックトラックを完全に無効にした後に、そのクラスターに対して再度有効にすることはできません。

コンソールを使用して DB クラスターのバックトラック機能を無効にするには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. [データベース] をクリックします。
3. 変更するクラスターを選択し、[変更] をクリックします。
4. [バックトラック] セクションで、[バックトラックを無効にする] をクリックします。
5. Continue (続行) をクリックします。
6. [Scheduling of Modifications (変更の予定)] で、以下のいずれかを選択します。
   - 次回の定期メンテナンス期間中に適用 - 次回のメンテナンスウィンドウまで待ってから変更を適用します。
   - すぐに適用 - 変更をできるだけ早急に適用します。
7. [クラスターの変更] をクリックします。

AWS CLI

AWS CLI を使用して DB クラスターのバックトラック機能を無効にするには、ターゲットバックトラックウィンドウを `0` (ゼロ) に設定します。クラスターのバックトラックを完全に無効にした後に、そのクラスターに対して再度有効にすることはできません。

AWS CLI を使用して DB クラスターのターゲットバックトラックウィンドウを変更するには

- AWS CLI の modify-db-cluster コマンドを呼び出して以下の値を指定します。
  - `--db-cluster-identifier` - DB クラスターの名前。
  - `--backtrack-window` - バックトラックをオフにするには、`0` を指定します。

  次の例では、`sample-cluster` のバックトラック機能を無効化するために `--backtrack-window` を `0` に設定します。

  Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --backtrack-window 0
```

  Windows の場合:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --backtrack-window 0
```

RDS API

Amazon RDS API を使用して DB クラスターのバックトラック機能を無効にするには、ModifyDBCluster オペレーションを使用します。`BacktrackWindow` 値を `0` (ゼロ) に設定し、`DBClusterIdentifier` 値に DB クラスターを指定します。クラスターのバックトラックを完全に無効にした後に、そのクラスターに対して再度有効にすることはできません。

障害挿入クエリを使用した Amazon Aurora MySQL のテスト

障害挿入クエリを使用して、Aurora MySQL DB クラスターの耐障害性をテストできます。フォールト挿入クエリは、SQL コマンドとして Amazon Aurora インスタンスに発行されます。次のいずれかのイベント発生をシミュレートしてスケジュールすることができます。

- 書き込みまたは読み取り DB インスタンスの障害

- Aurora レプリカの障害
- ディスクの障害
- ディスクの輻輳

障害挿入クエリでクラッシュを指定すると、Aurora MySQL DB インスタンスのクラッシュが強制的に実行されます。その他の障害挿入クエリでは、障害イベントのシミュレーションが実行されますが、そのイベントは発生しません。障害挿入クエリを送信する場合、障害イベントのシミュレーションが発生する時間も指定します。

Aurora レプリカのエンドポイントに接続することによって、Aurora レプリカインスタンスの 1 つに障害挿入クエリを送信できます。詳細については、「Amazon Aurora 接続管理」を参照してください。

障害挿入クエリの実行には、すべてのマスターユーザー権限が必要です。詳細については、「マスターユーザーアカウント権限」を参照してください。

インスタンスのクラッシュのテスト

`ALTER SYSTEM CRASH` 障害挿入クエリを使用して、Amazon Aurora インスタンスのクラッシュを強制的に発生させることができます。

この障害挿入クエリでは、フェイルオーバーが発生しません。フェイルオーバーをテストする場合、RDS コンソールで DB クラスターの [Failover] (フェイルオーバー) インスタンスアクションを選択するか、AWS CLI コマンドの failover-db-cluster、または RDS API の FailoverDBCluster オペレーションを使用します。

構文

```
ALTER SYSTEM CRASH [ INSTANCE | DISPATCHER | NODE ];
```

オプション

この障害挿入クエリでは、次のクラッシュタイプのいずれかを指定できます。

- **`INSTANCE`** — Amazon Aurora インスタンスの MySQL 互換データベースのクラッシュがシミュレートされます。
- **`DISPATCHER`** — Aurora DB クラスターのライターインスタンスにあるディスパッチャーのクラッシュがシミュレートされます。ディスパッチャー は Amazon Aurora DB クラスターのクラスターボリュームに対して更新を書き込みます。

- **NODE** — MySQL 互換データベースと Amazon Aurora インスタンスのディスパッチャーの両方のクラッシュがシミュレートされます。この障害挿入のシミュレーションでは、キャッシュも削除されます。

デフォルトのクラッシュタイプは INSTANCE です。

Aurora レプリカの障害のテスト

ALTER SYSTEM SIMULATE READ REPLICA FAILURE 障害挿入クエリを使用して、Aurora レプリカの障害をシミュレートできます。

Aurora レプリカの障害は、指定した期間で、ライターインスタンスから Aurora レプリカまたは DB クラスター内のすべての Aurora レプリカに対するすべてのリクエストをブロックします。指定した期間が終了すると、影響を受けた Aurora レプリカは自動的にマスターインスタンスと同期されます。

構文

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT READ REPLICA FAILURE
    [ TO ALL | TO "replica name" ]
    FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE |
 SECOND };
```

オプション

この障害挿入クエリでは、以下のパラメータを使用します。

- **percentage_of_failure** — 障害イベントの発生時にブロックするリクエストの割合です。この値は 0 ~ 100 の倍精度にすることができます。0 を指定すると、リクエストはブロックされません。100 を指定すると、すべてのリクエストがブロックされます。
- 障害のタイプ — シミュレートする障害のタイプ。DB クラスター内のすべての Aurora レプリカの障害をシミュレートするには、TO ALL を指定します。1 つの Aurora レプリカの障害をシミュレートするには、TO と Aurora レプリカの名前を指定します。デフォルトの障害タイプは TO ALL です。
- **quantity** — Aurora レプリカの障害のシミュレートにかかる時間。この値は時間の量の後に時間単位を続けて指定します。シミュレーションは、指定した単位の期間だけ発生します。例えば、20 MINUTE と指定すると、シミュレーションは 20 分間実行されます。

> **Note**
>
> Aurora レプリカの障害イベントの時間を指定するときには注意が必要です。指定する時間が長すぎ、障害イベントの発生時にライターインスタンスが大量のデータを書き込んだ場合、Aurora DB クラスターは Aurora レプリカがクラッシュしたものと見なし、レプリカを置き換える可能性があります。

ディスクの障害のテスト

`ALTER SYSTEM SIMULATE DISK FAILURE` 障害挿入クエリを使用して、Aurora DB クラスターのディスクの障害をシミュレートできます。

ディスク障害のシミュレーションでは、Aurora DB クラスターがランダムにディスクセグメントをエラーとしてマークします。シミュレーションの実行中、これらのセグメントに対するリクエストはブロックされます。

構文

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT DISK FAILURE
    [ IN DISK index | NODE index ]
    FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE |
 SECOND };
```

オプション

この障害挿入クエリでは、以下のパラメータを使用します。

- **`percentage_of_failure`** — 障害イベントの発生時にエラーとしてマークするディスクの割合。この値は 0 ~ 100 の倍精度にすることができます。0 を指定すると、ディスクのどの部分もエラーとしてマークされません。100 を指定すると、ディスク全体がエラーとしてマークされます。
- **`DISK index`** — 障害イベントをシミュレートする特定のデータの論理的なブロック。利用可能な論理的なデータブロックの範囲を超えた場合は、指定できる最大インデックス値を示すエラーが表示されます。詳細については、「Aurora MySQL DB クラスターのボリュームステータスの表示」を参照してください。
- **`NODE index`** — 障害イベントをシミュレートする特定のストレージノード。利用可能なストレージノードの範囲を超えた場合は、指定できる最大インデックス値を示すエラーが表示されます。詳

細については、「Aurora MySQL DB クラスターのボリュームステータスの表示」を参照してください。

- **quantity** — ディスクの障害をシミュレートする時間。この値は時間の量の後に時間単位を続けて指定します。シミュレーションは、指定した単位の期間だけ発生します。例えば、20 MINUTE と指定すると、シミュレーションは 20 分間実行されます。

ディスクの輻輳のテスト

ALTER SYSTEM SIMULATE DISK CONGESTION 障害挿入クエリを使用して、Aurora DB クラスターのディスクの障害をシミュレートできます。

ディスク輻輳のシミュレーションでは、Aurora DB クラスターがランダムにディスクセグメントを輻輳としてマークします。これらのセグメントに対するリクエストは、シミュレーションの実行中、指定した最小遅延値と最大遅延値の間で遅延します。

構文

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT DISK CONGESTION
    BETWEEN minimum AND maximum MILLISECONDS
    [ IN DISK index | NODE index ]
    FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE |
 SECOND };
```

オプション

この障害挿入クエリでは、以下のパラメータを使用します。

- **percentage_of_failure**—障害イベントの発生時に輻輳としてマークするディスクの割合です。この値は 0 ~ 100 の倍精度にすることができます。0 を指定すると、ディスクのどの部分も輻輳としてマークされません。100 を指定すると、ディスク全体が輻輳としてマークされます。
- **DISK index** または **NODE index** — 障害イベントをシミュレートする特定のディスクまたはノード。ディスクまたはノードのインデックスの範囲を超えた場合は、指定できる最大インデックス値を示すエラーが表示されます。
- **minimum** および **maximum** — 輻輳による遅延の最小値と最大値をミリ秒単位で指定します。シミュレーションの実行中、輻輳としてマークしたディスクセグメントでは、ミリ秒単位の最小値と最大値の間の範囲でランダムな遅延が発生します。

- **quantity** — ディスクの輻輳のシミュレートにかかる時間。この値は時間の量の後に時間単位を続けて指定します。シミュレーションは、指定した時間単位の期間だけ発生します。例えば、20 MINUTE と指定すると、シミュレーションは 20 分間実行されます。

高速 DDL を使用して Amazon Aurora のテーブルを変更する

Amazon Aurora には、ほぼ瞬時に所定の位置で ALTER TABLE オペレーションを実行するための最適化が含まれています。このオペレーションを実行するために、テーブルをコピーする必要はありません。また、他の DML ステートメントに実質的な影響を及ぼすことなく実行できます。このオペレーションは、テーブルのコピーにテンポラリストレージを使用しないため、スモールインスタンスクラスの大きなテーブルに対しても、DDL ステートメントを使用できます。

Aurora MySQL バージョン 3 は、インスタント DDL と呼ばれる MySQL 8.0 の特徴と互換性があります。Aurora MySQL バージョン 2 では、高速 DDL と呼ばれる異なる実装が使用されています。

トピック

インスタント DDL (Aurora MySQL バージョン 3)

DDL オペレーションの効率性を向上するため Aurora MySQL バージョン 3 によって実行される最適化を、インスタント DDL と呼びます。

Aurora MySQL バージョン 3 はコミュニティ MySQL 8.0 のインスタント DDL と互換性があります。インスタント DDL オペレーションを実行するには、ALTER TABLE ステートメントで ALGORITHM=INSTANT 句を使用します。インスタント DDL の構文と使用方法の詳細については、MySQL ドキュメントの「ALTER TABLE」ならびに「Online DDL Operations」(オンライン DDL オペレーション) を参照してください。

以下の例は、インスタンス DDL の特徴を説明しています。ALTER TABLE ステートメントは、列の追加、および列でのデフォルト値の変更を行います。例には、スタンダードカラムと仮想カラムの両方、およびスタンダードテーブルとパーティションテーブルの両方が含まれます。各ステップで、SHOW CREATE TABLE と DESCRIBE ステートメントを発行すると結果が確認できます。

```
mysql> CREATE TABLE t1 (a INT, b INT, KEY(b)) PARTITION BY KEY(b) PARTITIONS 6;
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> ALTER TABLE t1 RENAME TO t2, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ALTER COLUMN b SET DEFAULT 100, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.00 sec)

mysql> ALTER TABLE t2 ALTER COLUMN b DROP DEFAULT, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ADD COLUMN c ENUM('a', 'b', 'c'), ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 MODIFY COLUMN c ENUM('a', 'b', 'c', 'd', 'e'), ALGORITHM =
 INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ADD COLUMN (d INT GENERATED ALWAYS AS (a + 1) VIRTUAL), ALGORITHM
 = INSTANT;
Query OK, 0 rows affected (0.02 sec)

mysql> ALTER TABLE t2 ALTER COLUMN a SET DEFAULT 20,
    ->   ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t3 (a INT, b INT) PARTITION BY LIST(a)(
    ->   PARTITION mypart1 VALUES IN (1,3,5),
    ->   PARTITION MyPart2 VALUES IN (2,4,6)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> ALTER TABLE t3 ALTER COLUMN a SET DEFAULT 20, ALTER COLUMN b SET DEFAULT 200,
 ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t4 (a INT, b INT) PARTITION BY RANGE(a)
    ->   (PARTITION p0 VALUES LESS THAN(100), PARTITION p1 VALUES LESS THAN(1000),
    ->   PARTITION p2 VALUES LESS THAN MAXVALUE);
Query OK, 0 rows affected (0.05 sec)

mysql> ALTER TABLE t4 ALTER COLUMN a SET DEFAULT 20,
    ->   ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

/* Sub-partitioning example */
```

```
mysql> CREATE TABLE ts (id INT, purchased DATE, a INT, b INT)
    ->   PARTITION BY RANGE( YEAR(purchased) )
    ->     SUBPARTITION BY HASH( TO_DAYS(purchased) )
    ->     SUBPARTITIONS 2 (
    ->       PARTITION p0 VALUES LESS THAN (1990),
    ->       PARTITION p1 VALUES LESS THAN (2000),
    ->       PARTITION p2 VALUES LESS THAN MAXVALUE
    ->    );
Query OK, 0 rows affected (0.10 sec)

mysql> ALTER TABLE ts ALTER COLUMN a SET DEFAULT 20,
    ->   ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)
```

高速 DDL (Aurora MySQL バージョン 2)

MySQL において、何度もデータ操作言語 (DDL) オペレーションを行うと、パフォーマンスに大きな影響が出ることがあります。

例えば、ALTER TABLE オペレーションを使用して列をテーブルに追加するとします。オペレーションに指定するアルゴリズムによっては、このオペレーションに以下の操作が伴う場合があります。

- テーブル全体のコピーの作成
- 同時データ操作言語 (DML) オペレーションを処理するためのテンポラリテーブルの作成
- テーブルのすべてのインデックスの再構築
- 同時 DML 変更の適用時におけるテーブルロックの適用
- 同時 DML スループットの低下

DDL オペレーションの効率性向上のために Aurora MySQL バージョン 2 によって実行される最適化を、高速 DDL と呼びます。

Aurora MySQL バージョン 3 では、Aurora はインスタント DDL と呼ばれる MySQL 8.0 特徴を使用します。Aurora MySQL バージョン 2 では、高速 DDL と呼ばれる異なる実装が使用されています。

> **Important**
>
> 現在、Aurora MySQL で高速 DDL を使用するには、Aurora ラボモードを有効にする必要があります。本番 DB クラスターに高速 DDL を使用することはお勧めしません。Aurora ラボ

モードを有効にする方法については、「Amazon Aurora MySQL ラボモード」を参照してください。

高速 DDL の制限事項

現在、高速 DDL には以下の制限があります。

- 高速 DDL は、NULL を許容する列 (デフォルト値を持たない) を、既存テーブルの末尾に追加する場合にのみ使用できます。
- 高速DDLは、パーティション化されたテーブルでは機能しません。
- 高速 DDL は、REDUNDANT 行形式を使用する InnoDB テーブルをサポートしていません。
- Fast DDL は、フルテキスト検索インデックスを持つテーブルでは機能しません。
- DDL オペレーションの最大可能レコードサイズが大きすぎる場合、高速 DDL は使用されません。ページサイズの半分を超えるレコードサイズは大きすぎます。レコードの最大サイズは、すべての列の最大サイズを追加して計算されます。サイズを変更可能な列の場合は、InnoDB スタンダードに基づき、extern byte は計算に含まれません。

高速 DDL の構文

```
ALTER TABLE tbl_name ADD COLUMN col_name column_definition
```

このステートメントには、以下のオプションがあります。

- **tbl_name** — 変更するテーブルの名前。
- **col_name** — 追加する列の名前。
- **col_definition** — 追加する列の定義。

> **Note**
>
> NULL を許容する列の定義は、デフォルト値を使用せずに指定する必要があります。そうでない場合、高速 DDL は使用されません。

高速 DDL の例

次の例は、高速 DDL オペレーションによる高速化を示しています。最初の SQL の例では、高速 DDL を使用せずに、大きなテーブルに対して ALTER TABLE ステートメントを実行しています。この操作にはかなりの時間がかかります。CLI の例は、クラスターで高速 DDL を有効化する方法を示しています。次に、別の SQL の例では、同じテーブルで同じ ALTER TABLE ステートメントを実行します。高速 DDL を有効にすると、オペレーションが非常に高速になります。

この例では、1 億 5000 万行を含む、TPC-H ベンチマークの ORDERS テーブルを使用しています。このクラスターでは、比較的小さなインスタンスクラスを意図的に使用して、高速 DDL を使用できない場合の ALTER TABLE ステートメントの所要時間を示しています。この例では、同じデータを含む元のテーブルのクローンを作成します。aurora_lab_mode 設定を確認すると、ラボモードが有効になっていないため、クラスターで高速 DDL を使用できないことが分かります。その場合、ALTER TABLE ADD COLUMN ステートメントでテーブルの最後に新しい列を追加するには、かなりの時間がかかります。

```
mysql> create table orders_regular_ddl like orders;
Query OK, 0 rows affected (0.06 sec)

mysql> insert into orders_regular_ddl select * from orders;
Query OK, 150000000 rows affected (1 hour 1 min 25.46 sec)

mysql> select @@aurora_lab_mode;
+-------------------+
| @@aurora_lab_mode |
+-------------------+
|                 0 |
+-------------------+

mysql> ALTER TABLE orders_regular_ddl ADD COLUMN o_refunded boolean;
Query OK, 0 rows affected (40 min 31.41 sec)

mysql> ALTER TABLE orders_regular_ddl ADD COLUMN o_coverletter varchar(512);
Query OK, 0 rows affected (40 min 44.45 sec)
```

この例では、前の例と同様に大きなテーブルを準備します。ただし、Interactive SQL セッション内で単にラボモードを有効にすることはできません。この設定は、カスタムパラメータグループで有効にする必要があります。そのためには、mysql セッションを終了して AWS CLI コマンドをいくつか実行するか、AWS Management Console を使用する必要があります。

```
mysql> create table orders_fast_ddl like orders;
```

```
Query OK, 0 rows affected (0.02 sec)

mysql> insert into orders_fast_ddl select * from orders;
Query OK, 150000000 rows affected (58 min 3.25 sec)

mysql> set aurora_lab_mode=1;
ERROR 1238 (HY000): Variable 'aurora_lab_mode' is a read only variable
```

クラスターのラボモードを有効にするには、パラメータグループをいくつか使用する必要があります。この AWS CLI の例では、クラスターのパラメータグループを使用して、クラスター内のすべての DB インスタンスのラボモード設定で同じ値を使用するようにします。

```
$ aws rds create-db-cluster-parameter-group \
  --db-parameter-group-family aurora5.7 \
    --db-cluster-parameter-group-name lab-mode-enabled-57 --description 'TBD'
$ aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name lab-mode-enabled-57 \
    --query '*[*].[ParameterName,ParameterValue]' \
      --output text | grep aurora_lab_mode
aurora_lab_mode 0
$ aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name lab-mode-enabled-57 \
    --parameters ParameterName=aurora_lab_mode,ParameterValue=1,ApplyMethod=pending-
reboot
{
    "DBClusterParameterGroupName": "lab-mode-enabled-57"
}

# Assign the custom parameter group to the cluster that's going to use Fast DDL.
$ aws rds modify-db-cluster --db-cluster-identifier tpch100g \
  --db-cluster-parameter-group-name lab-mode-enabled-57
{
  "DBClusterIdentifier": "tpch100g",
  "DBClusterParameterGroup": "lab-mode-enabled-57",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.10.2",
  "Status": "available"
}

# Reboot the primary instance for the cluster tpch100g:
$ aws rds reboot-db-instance --db-instance-identifier instance-2020-12-22-5208
{
  "DBInstanceIdentifier": "instance-2020-12-22-5208",
```

```
  "DBInstanceStatus": "rebooting"
}

$ aws rds describe-db-clusters --db-cluster-identifier tpch100g \
  --query '*[].[DBClusterParameterGroup]' --output text
lab-mode-enabled-57

$ aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name lab-mode-enabled-57 \
    --query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
      --output text | grep aurora_lab_mode
aurora_lab_mode 1
```

次の例では、パラメータグループの変更が有効になった後のステップを示します。クラスターで高速 DDL を使用できることを確認するため、aurora_lab_mode 設定をテストします。次に、ALTER TABLE ステートメントを実行して、別の大きなテーブルの末尾に列を追加します。ここでは、ステートメントは非常に高速で終了します。

```
mysql> select @@aurora_lab_mode;
+-------------------+
| @@aurora_lab_mode |
+-------------------+
|                 1 |
+-------------------+

mysql> ALTER TABLE orders_fast_ddl ADD COLUMN o_refunded boolean;
Query OK, 0 rows affected (1.51 sec)

mysql> ALTER TABLE orders_fast_ddl ADD COLUMN o_coverletter varchar(512);
Query OK, 0 rows affected (0.40 sec)
```

Aurora MySQL DB クラスターのボリュームステータスの表示

Amazon Aurora において、DB クラスターボリュームは、論理的なブロックのコレクションで構成されます。各論理的なブロックは、10 ギガバイトの割り当て済みストレージです。これらのブロックは保護グループと呼ばれます。

各保護グループのデータは、6 つの物理ストレージデバイス (ストレージノードと呼ばれる) にわたってレプリケートされます。これらのストレージノードは、DB クラスターがある AWS リージョン内の 3 つのアベイラビリティーゾーン (AZ) に割り当てられます。また、各ストレージノードには、DB クラスターボリュームに対する 1 つ以上の論理的なデータブロックが含まれます。保護グ

ループおよびストレージノードの詳細については、AWS データベースブログの「Aurora ストレージエンジンの概要」を参照してください。

ストレージノード全体の障害、またはストレージノード内の単一の論理的なデータブロックの障害をシミュレートできます。シミュレートするには、ALTER SYSTEM SIMULATE DISK FAILURE の障害挿入ステートメントを使用します。このステートメントで、特定の論理的なデータブロックまたはストレージノード全体のインデックス値を指定します。ただし、DB クラスターボリュームで使用されている論理的なデータブロックまたはストレージノードの数より大きいインデックス値を指定すると、ステートメントからエラーが返されます。障害挿入クエリの詳細については、「障害挿入クエリを使用した Amazon Aurora MySQL のテスト」を参照してください。

このエラーは、SHOW VOLUME STATUS ステートメントを使用して回避できます。このステートメントからは、2 つのサーバーステータス変数 (Disks および Nodes) が返されます。これらの変数は、DB クラスターボリュームの論理的なデータブロックとストレージノードの合計数をそれぞれ示します。

構文

```
SHOW VOLUME STATUS
```

例

次の例は、SHOW VOLUME STATUS の一般的な結果を示しています。

```
mysql> SHOW VOLUME STATUS;
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Disks         | 96    |
| Nodes         | 74    |
+---------------+-------+
```

Aurora MySQL のチューニング

待機イベントとスレッドの状態は、Aurora MySQL の重要なチューニングツールです。セッションがリソースを待っている理由や、何を実行しているのかを知ることができれば、ボトルネックを減少できます。このセクションの情報を使用して、考えられる原因と修正措置を見つけることができます。

Amazon DevOps Guru for RDS は、Aurora MySQL データベースに、後でさらに大きな問題を引き起こす可能性がある問題条件があるかどうかを事前に判断できます。Amazon DevOps Guru for RDS は、是正措置の説明と推奨事項をプロアクティブインサイトで発行します。このセクションには、一般的な問題に関するインサイトが含まれています。

> **Important**
>
> このセクションの待機イベントとスレッドの状態は、Aurora MySQL に固有です。このセクションの情報は Amazon Aurora の調整にのみ使用し、Amazon RDS for MySQL には使用しないでください。
> このセクションの待機イベントの一部は、これらのデータベースエンジンのオープンソースバージョンに対応するものがありません。他の待機イベントの名前は、オープンソースエンジンのイベントと同じですが、動作が異なります。例えば、Amazon Aurora ストレージはオープンソースストレージとは異なる動作をするため、ストレージ関連の待機イベントは異なるリソース条件を示します。

トピック

Aurora MySQL チューニングの基本的な概念

Aurora MySQL データベースを調整する前に、待機イベントとスレッドの状態はどうなっているか、またその発生理由を確認してください。InnoDB ストレージエンジンを使用するときは、Aurora MySQL の基本的なメモリとディスクアーキテクチャも確認してください。便利なアーキテクチャ図表については、「MySQL リファレンスマニュアル」を参照してください。

トピック

- Aurora MySQL の待機イベント
- Aurora MySQL スレッド状態
- Aurora MySQL メモリ
- Aurora MySQL プロセス

Aurora MySQL の待機イベント

待機イベントはセッションが待っているリソースを示します。例えば、待機イベント `io/socket/sql/client_connection` はスレッドが新しい接続を処理中であることを示します。セッションが待機する一般的なリソースには、次のものがあります。

- バッファへのシングルスレッドアクセス (例えば、セッションがバッファを変更しようとした場合など)
- 別のセッションによって現在ロックされている行
- 読み込まれたデータファイル
- ログファイルの書き込み

例えば、クエリを満たすために、セッションで完全なテーブルスキャンを実行することがあります。データがまだメモリ上にない場合、セッションはディスク I/O が完了するまで待機します。バッファがメモリに読み込まれるときは、他のセッションが同じバッファにアクセスしているため、セッションは待機しなければならないことがあります。データベースは、事前定義された待機イベントを使用して待機を記録します。これらのイベントはカテゴリに分類されます。

待機イベント自体では、パフォーマンスの問題は表示されません。例えば、要求されたデータがメモリ上にない場合は、ディスクからデータを読み出す必要があります。あるセッションが更新のために行をロックすると、別のセッションはその行を更新できるようにロック解除されるまで待機します。コミットは、ログファイルへの書き込みが完了するまで待機する必要があります。待機は、データベースが正常に機能するために不可欠です。

一般的に、大量の待機イベントはパフォーマンスの問題を示します。そのような場合、待機イベントデータを使用して、セッションが時間を費やしている場所を特定できます。例えば、通常は分単位で実行されるレポートが数時間実行される場合、合計待機時間に最も多く寄与する待機イベントを特定できます。上位の待機イベントの原因を特定できる場合は、パフォーマンス向上のための変更を実行できることがあります。例えば、別のセッションによってロックされている行でセッションが待っている場合、ロックセッションを終了します。

Aurora MySQL スレッド状態

一般的なスレッド状態は`State`一般的なクエリ処理に関連付けられている値です。例えばスレッドの状態 `sending data` は、スレッドがクエリの行を読み取り、フィルタリングして正しい結果セットを判断していることを示します。

スレッド状態を使用すると、待機イベントの使用方法と同じような仕様で Aurora MySQL を調整できます。例えば`sending data` の頻繁な発生は通常、クエリがインデックスを使用していないことを示します。スレッド状態の詳細については、MySQL リファレンスマニュアルの一般的なスレッドステートを参照してください。

Performance Insights を使用する場合、以下の条件のいずれかに当てはまります。

- パフォーマンススキーマがオンになっている — Aurora MySQL はスレッド状態ではなく待機イベントを表示します。
- パフォーマンススキーマがオンになっていない — Aurora MySQL はスレッド状態を表示します。

パフォーマンススキーマは、自動管理に設定することをお勧めします。パフォーマンススキーマは、潜在的なパフォーマンスの問題を調査するための追加のインサイトと優れたツールを提供します。詳細については、Aurora MySQL における Performance Insights の Performance Schema の有効化 を参照してください。

Aurora MySQL メモリ

Aurora MySQL では、最も重要なメモリ領域はバッファプールとログバッファです。

トピック

- バッファプール

バッファプール

バッファプールは Aurora MySQL がテーブルとインデックスデータをキャッシュする共有メモリ領域です。クエリはディスクから読み取ることなく、頻繁に使用されるデータにメモリから直接アクセスできます。

バッファプールは、ページのリンクリストとして構成されています。ページは複数の行を保持できます。Aurora MySQL は、プールからページをエージングアウトするために、最近最も使用されていない (LRU) アルゴリズムを使用します。

詳細については、MySQL リファレンスマニュアルの「Buffer Pool」(バッファプール) を参照してください。

Aurora MySQL プロセス

Aurora MySQL は Aurora PostgreSQL とは大きく異なるプロセスモデルを使用しています。

トピック

- MySQLサーバー (mysqld)
- スレッド
- スレッドプール

MySQLサーバー (mysqld)

MySQL サーバーは mysqld という名前の単一のオペレーティングシステムプロセスです。MySQL サーバーは追加のプロセスを生成しません。したがって Aurora MySQL データベースは、 mysqld を使用してほとんどの作業を実行します。

MySQL サーバーが起動すると、MySQL クライアントからのネットワーク接続をリッスンします。クライアントがデータベースに接続すると、mysqld はスレッドを開きます。

スレッド

接続マネージャースレッドは、各クライアント接続を専用スレッドに関連付けます。このスレッドは、認証を管理し、ステートメントを実行し、結果をクライアントに返します。接続マネージャは、必要に応じて新しいスレッドを作成します。

スレッドキャッシュは使用可能なスレッドのセットです。接続が終了すると、キャッシュがいっぱいでない場合、MySQL はスレッドをスレッドキャッシュに返します。`thread_cache_size` システム可変は、スレッドキャッシュのサイズを決定します。

スレッドプール

スレッドプールは複数のスレッドグループで構成されています。各グループは一連のクライアント接続を管理します。クライアントがデータベースに接続すると、スレッドプールはラウンドロビン方式でスレッドグループに接続を割り当てます。スレッドプールは接続とスレッドを分離します。接続と、それらの接続から受信した文を実行するスレッドの間には、固定された関係はありません。

待機イベントを使用した Aurora MySQL のチューニング

次の表は、パフォーマンス問題を最もよく示す Aurora MySQL の待機イベントをまとめたものです。以下の待機イベントは、Aurora MySQL の待機イベント のリストのサブセットです。

待機イベント	説明
cpu	このイベントは、スレッドが CPU でアクティブであるか、CPU を待っているときに発生します。
io/aurora_redo_log_flush	このイベントは、セッションが永続データを Aurora ストレージに書き込むときに発生します。
io/aurora_respond_to_client	イベントは、スレッドが結果セットをクライアントに返すのを待っているときに発生します。
io/redo_log_flush	このイベントは、セッションが永続データを Aurora ストレージに書き込むときに発生します。
io/socket/sql/client_connection	このイベントは、スレッドが新しい接続を処理している最中のときに発生します。
io/table/sql/handler	このイベントは、作業がストレージエンジンに委任されたときに発生します。
synch/cond/innodb/row_lock_wait	このイベントは、あるセッションが更新のために行をロックし、別のセッションが同じ行を更新しようとしたときに発生します。
synch/cond/innodb/row_lock_wait_cond	このイベントは、あるセッションが更新のために行をロックし、別のセッションが同じ行を更新しようとしたときに発生します。

待機イベント	説明
synch/cond/sql/MDL_context::COND_wait_status	このイベントは、テーブルメタデータロックに待機中のスレッドがあるときに発生します。
synch/mutex/innodb/aurora_lock_thread_slot_futex	このイベントは、あるセッションが更新のために行をロックし、別のセッションが同じ行を更新しようとしたときに発生します。
synch/mutex/innodb/buf_pool_mutex	このイベントは、スレッドがメモリ内のページにアクセスするために InnoDB バッファプールのロックを取得したときに発生します。
synch/mutex/innodb/fil_system_mutex	このイベントは、セッションがテーブルスペースのメモリキャッシュへのアクセスを待っているときに発生します。
synch/mutex/innodb/trx_sys_mutex	このイベントは、大量のトランザクションで高いデータベースアクティビティがある場合に発生します。
synch/sxlock/innodb/hash_table_locks	このイベントは、バッファプール内には見つからないページをファイルから読み込む必要がある場合に発生します。

cpu

cpu 待機イベントは、スレッドが CPU でアクティブな場合、または CPU を待っている際に発生します。

トピック

- サポート対象エンジンバージョン
- Context
- 待ち時間増加の考えられる原因
- アクション

サポート対象エンジンバージョン

この待機イベント情報は、以下のエンジンバージョンでサポートされています。

- Aurora MySQL バージョン 2 および 3

Context

すべての vCPU に対して、接続はこの CPU 上で動作を実行できます。状況によっては、実行可能なアクティブな接続の数が vCPUs の数よりも多くなります。この不均衡により、接続が CPU リソースを待機することになります。アクティブな接続の数が vCPUs の数よりも常に多い場合、インスタンスに CPU 競合が発生します。この競合により、cpu 待機イベントが発生します。

> **Note**
>
> CPU の Performance Insights メトリクスは DBLoadCPU です。DBLoadCPU の値は CloudWatch メトリクス CPUUtilization の値とは異なる場合があります。後者のメトリクスは、データベースインスタンスのハイパーバイザーから収集されます。

Performance Insights OS メトリクスは、CPU 使用率に関する詳細情報を提供します。例えば、以下のメトリクスを表示できます。

- os.cpuUtilization.nice.avg
- os.cpuUtilization.total.avg
- os.cpuUtilization.wait.avg
- os.cpuUtilization.idle.avg

Performance Insights は、データベースエンジンによる CPU 使用率を os.cpuUtilization.nice.avg として報告します。

待ち時間増加の考えられる原因

このイベントが通常よりも多く発生し、パフォーマンス問題を示している可能性がある場合、典型的な原因は次のとおりです。

- 分析クエリ
- 高パラレルトランザクション

- 実行時間が長いトランザクション
- ログインストームとして知られる、接続数の急激な増加。
- コンテキスト切り替えの増加

アクション

cpu 待機イベントがデータベースアクティビティを占領している場合でも、必ずしもパフォーマンスの問題を示すわけではありません。パフォーマンスが低下した場合にのみ、このイベントに応答します。

CPU 使用率の増加の原因に応じて、次の戦略を検討してください。

- ホストの CPU 容量を増やします。このアプローチは通常、テンポラリ救済しか与えません。
- 潜在的な最適化のためのトップクエリを特定します。
- 読み取り専用ワークロードをリーダーノードにリダイレクトします (該当する場合)。

トピック

- 問題の原因となっているセッションまたはクエリを特定します。
- 高い CPU ワークロードを分析して最適化する

問題の原因となっているセッションまたはクエリを特定します。

セッションとクエリを見つけるには、Performance Insights のトップ SQL の表で、CPU ロードが最も高い SQL ステートメントを探します。詳細については、「Performance Insights ダッシュボードを使用してメトリクスを分析する」を参照してください。

通常、1 つまたは 2 つの SQL ステートメントは CPU サイクルの大半を消費します。これらのステートメントを中心に確認してください。DB インスタンスに 2 つの vCPUs があり、DB ロードが平均 3.1 のアクティブセッション (AAS) がすべて CPU 状態にあるとします。この場合、インスタンスは CPU バインドされています。以下の戦略を検討して下さい。

- より多くの vCPUs を持つより大きなインスタンスクラスにアップグレードします。
- クエリを調整して CPU ロードを下げます。

この例では、上位 SQL クエリの DB ロードが 1.5 AAS で、すべて CPU 状態です。別の SQL ステートメントの CPU 状態では 0.1 のロードがあります。この例では、lowest-load SQL 文を停止して

も、データベースのロードは大幅に軽減されません。ただし、2 つの高ロードクエリを最適化して 2 倍の効率が得られると、CPU のボトルネックがなくなります。1.5 AAS の CPU ロードを 50% 減らすと、各ステートメントの AAS は 0.75 に減少します。CPU に費やされた DB ロードの合計は 1.6 AAS になりました。この値は、vCPU の最大ラインの 2.0 を下回っています。

Performance Insights を使用したトラブルシューティングの便利な概要については、ブログ記事 Performance Insights を使用した Amazon Aurora MySQL ワークロードの分析 を参照してください。また AWS Support 記事 Amazon RDS for MySQL インスタンスで CPU 使用率が高い CPU 使用率のトラブルシューティングと解決方法を教えてください。も参照してください。

高い CPU ワークロードを分析して最適化する

CPU 使用率を増加させているクエリを特定したら、それらを最適化するか、接続を終了します。次の例は、接続の終了方法を解説しています。

```
CALL mysql.rds_kill(processID);
```

詳細については、「mysql.rds_kill」を参照してください。

セッションを終了すると、アクションによって長いロールバックがトリガーされることがあります。

クエリを最適化するためのガイドラインに従う

クエリを最適化するには、次のガイドラインを考慮してください。

- `EXPLAIN` ステートメントを実行します。

  このコマンドは、クエリの実行に関連する個々のステップを示しています。詳細については、MySQL ドキュメントの「EXPLAIN を使用したクエリの最適化」を参照してください。
- `SHOW PROFILE` ステートメントを実行します。

  このステートメントを使用して、現在のセッション中に実行されるステートメントのリソース使用状況を示すプロファイル詳細を確認します。詳細については、MySQL ドキュメントの SHOW PROFILE ステートメントを参照してください。
- `ANALYZE TABLE` ステートメントを実行します。

  このステートメントを使用して、CPU を大量に消費するクエリによってアクセスされるテーブルのインデックス統計を更新します。ステートメントを分析することで、オプティマイザが適切な実行プランを選択するのに役立ちます。詳細については、MySQL ドキュメントの ANALYZE TABLE ステートメントを参照してください。

CPU 使用率を改善するためのガイドラインに従ってください

データベースインスタンスの CPU 使用率を向上させるには、次のガイドラインに従ってください。

- すべてのクエリが適切なインデックスを使用していることを確認します。
- Aurora パラレルクエリを使用できるかどうかを調べます。このテクニックを使うと、関数処理、行のフィルタリング、および WHERE 句のプロジェクションをプッシュダウンすることにより、ヘッドノードの CPU 使用率を削減することができます。
- 1 秒あたりの SQL 実行数が予想されるしきい値と合っているかを調べます。
- インデックスのメンテナンスまたは新規インデックスの作成が、本番ワークロードで必要な CPU サイクルにかかるかどうかを調べます。ピークアクティビティ時間外のメンテナンスアクティビティをスケジュールします。
- パーティショニングを使用してクエリデータセットを削減できるかどうかを調べます。詳細については、ブログの投稿記事 統合ワークロードに向けて Amazon Aurora with MySQL の互換性を計画、最適化する方法を参照してください。

接続ストームをチェックする

`DBLoadCPU` メトリクスはそれほど高くはないが `CPUUtilization` メトリクスが高い場合、CPU 使用率が高い原因はデータベースエンジンの外部にあります。典型的な例はコネクションストームです。

以下の条件に該当するかどうかを確認してください。

- Performance Insights `CPUUtilization` メトリックスと Amazon CloudWatch `DatabaseConnections` メトリクスの両方で増加が見られる。
- CPU 内のスレッド数が vCPUs 数を超えいる。

上記の条件に当てはまる場合は、データベース接続数を減らすことを検討してください。例えば、RDS プロキシなどの接続プールを使用できます。効果的な接続管理とスケーリングのベストプラクティスを学ぶには、ホワイトペーパー接続管理のための Amazon Aurora MySQL DBA ハンドブックを参照してください。

io/aurora_redo_log_flush

`io/aurora_redo_log_flush` イベントは、セッションが永続データを Amazon Aurora ストレージに書き込むときに発生します。

トピック

サポート対象エンジンバージョン

この待機イベント情報は、以下のエンジンバージョンでサポートされています。

- Aurora MySQL バージョン 2

Context

`io/aurora_redo_log_flush` イベントは Aurora MySQL の書き込み入力/出力 (I/O) オペレーション用です。

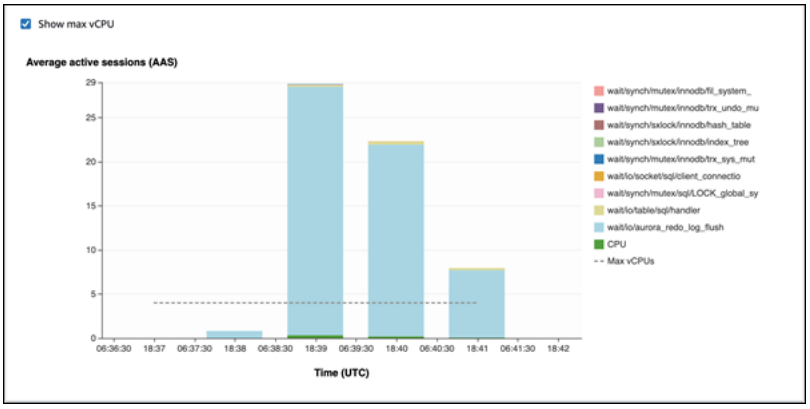
> **Note**
>
> Aurora MySQL バージョン 3 では、この待機イベントの名前は io/redo_log_flush です。

待ち時間増加の考えられる原因

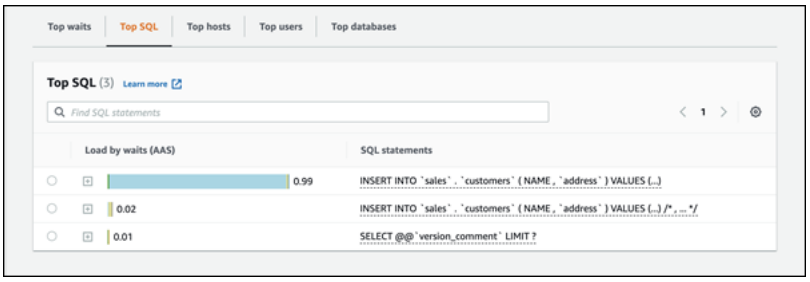
データの永続化のために、コミットはストレージが安定するよう耐久性の高い書き込みを要求とします。データベースがコミットを多くし過ぎると、書き込み I/O オペレーションで待機イベントが発生します、`io/aurora_redo_log_flush` 待機イベント。

次の例では、db.r5.xlarge DB インスタンスクラスを使用して、50,000 レコードが Aurora MySQL DB クラスターに挿入されています。

- 初期の例では、各セッションは行ごとに 10,000 レコードを挿入しています。デフォルトで、データ操作言語 (DML) コマンドがトランザクション内にない場合、Aurora MySQL は暗黙的なコミットを使用します。オートコミットがオンになっています。これは、各行の挿入に対してコミットがあることを意味します。Performance Insights は、コネクションがほとんどの時間`io/aurora_redo_log_flush` 待機イベントで待っていることを示しています。

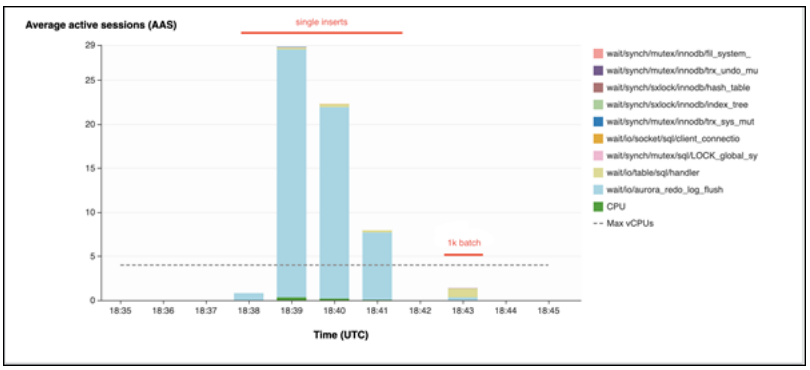


これは、使用される単純な挿入ステートメントが原因です。



50,000 レコードは挿入されるまで 3.5 分 かかります。

- 2 番目の例では、挿入は 1,000 バッチで作成されます。つまり、各接続は 10,000 ではなく 10 のコミットを実行します。Performance Insights は、コネクションがほとんどの時間 `io/aurora_redo_log_flush` 待機イベントで待っていないことを示しています。



50,000 レコードが挿入されるまでに 4 秒かかります。

アクション

待機イベントの原因に応じて、異なるアクションをお勧めします。

問題のあるセッションとクエリを特定する

DB インスタンスにボトルネックが発生している場合、ユーザーの初期のタスクは、その原因となるセッションとクエリを見つけることになります。便利な AWS データベースブログ記事は Performance Insights を使用した Amazon Aurora MySQL ワークロードの分析 を参照してください。

ボトルネックの原因となっているセッションとクエリを特定するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。
4. データベース負荷で、待機でスライスを選択します。
5. ページの下部で トップ SQL を選択します。

   リストの上部にあるクエリは、データベースで最大の負荷を引き起こしています。

書き込みオペレーションをグループ化する

次の例は io/aurora_redo_log_flush 待機イベントをトリガーしています。(オートコミットがオンになっています。)

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
```

io/aurora_redo_log_flush 待機イベントで待機する時間を減らすため、書き込み操作を論理的に 1 つのコミットにグループ化し、ストレージへの永続的な呼び出しを減らします。

オートコミットをオフにする

次の例に示すように、トランザクション内に存在しない大きな変更を加える前に、オートコミットをオフにします。

```
SET SESSION AUTOCOMMIT=OFF;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
-- Other DML statements here
COMMIT;

SET SESSION AUTOCOMMIT=ON;
```

トランザクションの使用

次の例が示すように、トランサクションを使用することができます。

```
BEGIN
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;

-- Other DML statements here
END
```

バッチを使用する

次の例が示すように、バッチで変更することもできます。ただし、大きすぎるバッチを使用すると、特にリードレプリカやポイントインタイムリカバリ (PITR) の実行時にパフォーマンスの問題が発生する可能性があります。

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES
('xxxx','xxxxx'),('xxxx','xxxxx'),...,('xxxx','xxxxx'),('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1 BETWEEN xx AND
 xxx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1<xx;
```

io/aurora_respond_to_client

`io/aurora_respond_to_client` イベントは、スレッドが結果セットをクライアントに返すのを待っているときに発生します。

トピック

- サポート対象エンジンバージョン
- Context
- 待ち時間増加の考えられる原因
- アクション

サポート対象エンジンバージョン

この待機イベント情報は、以下のエンジンバージョンでサポートされています。

- Aurora MySQL バージョン 2

バージョン 2.07.7、2.09.3、2.10.2 より前のバージョンでは、この待機イベントにアイドル時間が誤って含まれます。

Context

`io/aurora_respond_to_client` イベントは、スレッドが結果セットをクライアントに返すのを待っているときに発生します。

クエリの処理が完了し、結果はアプリケーションクライアントに返されます。ただし、DB クラスターには十分なネットワーク帯域幅がないため、スレッドは結果セットを返すのを待っています。

待ち時間増加の考えられる原因

`io/aurora_respond_to_client` イベントが通常よりも多く表示され、パフォーマンスの問題を示している可能性がある場合、典型的な原因は次のとおりです。

DB インスタンスクラスがワークロードに対して不十分

DB クラスターで使用される DB インスタンスクラスには、ワークロードを効率的に処理するために必要なネットワーク帯域幅がありません。

大きな結果セット

クエリがより多くの行数を返すため、返される結果セットのサイズが増加しました。結果セットが大きいほど、より多くのネットワーク帯域幅を消費します。

クライアントへの負荷の増加

クライアントに CPU プレッシャー、メモリプレッシャー、またはネットワークの飽和が発生する可能性があります。クライアントの負荷が増加すると、Aurora MySQL DB クラスターからのデータの受信が遅れます。

ネットワークレイテンシーの増加

Aurora MySQL DB クラスターとクライアント間のネットワークレイテンシーが増加することがあります。ネットワークレイテンシーが大きいほど、クライアントがデータを受信するのに必要な時間が長くなります。

アクション

待機イベントの原因に応じたさまざまなアクションをお勧めします。

トピック

- イベントの原因となるセッションとクエリを特定する
- DB インスタンスクラスのスケール
- 予期しない結果のワークロードをチェックする
- リーダーインスタンスでワークロードを分散する
- SQL_BUFFER_RESULT 修飾子を使用する

イベントの原因となるセッションとクエリを特定する

Performance Insights を使用して、`io/aurora_respond_to_client` 待機イベントによってブロックされたクエリを表示できます。通常、ロードが中程度から重大なデータベースには、待機イベントがあります。パフォーマンスが最適であれば、待機イベントは受け入れられる可能性があります。パフォーマンスが最適でない場合は、データベースが最も長い時間を費やしている場所を調べます。最も高いロードに寄与する待機イベントを調べて、データベースとアプリケーションを最適化してこれらのイベントを削減できるかどうかを調べます。

高ロードの原因となる SQL クエリを検索するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。その DB インスタンスの Performance Insights ダッシュボードが表示されます。
4. データベースロードで、待機でスライスを選択します。
5. ページの下部で トップ SQL を選択します。

   グラフには、ロードの原因となる SQL クエリがリストされます。リスト上部にあるクエリに、最も責任があります。ボトルネックを解決するには、これらのステートメントに注目してください。

Performance Insights を使用したトラブルシューティングの便利な概要については、AWS　データベースブログ記事の Performance Insights を使用した Amazon Aurora MySQL ワークロードの分析を参照してください。

DB インスタンスクラスのスケール

`NetworkReceiveThroughput` や `NetworkTransmitThroughput` などのネットワークスループットに関連する Amazon CloudWatch メトリクスの値の増加を確認します。DB インスタンスクラスのネットワーク帯域幅に達している場合は、DB クラスターを変更することで、DB クラスターで使用される DB インスタンスクラスをスケールできます。より大きなネットワーク帯域幅を持つ DB インスタンスクラスは、データをより効率的にクライアントに返します。

Amazon CloudWatch メトリクスのモニタリングについては、Amazon RDS コンソールでのメトリクスの表示 を参照してください。DB インスタンスクラスの詳細については、「Aurora DB インスタ

ンスクラス」を参照してください。DB クラスターの変更については、「Amazon Aurora DB クラスターの変更」を参照してください。

予期しない結果のワークロードをチェックする

DB クラスターのワークロードをチェックし、予期しない結果が発生していないことを確認します。例えば、予想よりも多くの行を返すクエリがある可能性があります。この場合、Innodb_rows_read などの Performance Insights カウンターメトリクスを使用できます。詳細については、「Performance Insights カウンターメトリクス」を参照してください。

リーダーインスタンスでワークロードを分散する

Aurora レプリカを使用して、読み取り専用ワークロードを配布できます。Aurora レプリカを追加することで、水平方向にスケールできます。そうすると、ネットワーク帯域幅のスロットリング制限が増加する可能性があります。詳細については、「Amazon Aurora DB クラスター」を参照してください。

SQL_BUFFER_RESULT 修飾子を使用する

SQL_BUFFER_RESULT 修飾子 を SELECT ステートメントに追加すると、結果がクライアントに返される前にテンポラリテーブルに強制できます。クエリは io/aurora_respond_to_client 待機状態になっているため、InnoDB ロックが解放されていない時この修飾子はパフォーマンスの問題に役立ちます。詳細については、MySQL ドキュメントの SELECT ステートメントを参照してください。

io/redo_log_flush

io/redo_log_flush イベントは、セッションが永続データを Amazon Aurora ストレージに書き込むときに発生します。

トピック

- サポート対象エンジンバージョン
- Context
- 待ち時間増加の考えられる原因
- アクション

サポート対象エンジンバージョン

この待機イベント情報は、以下のエンジンバージョンでサポートされています。

- Aurora MySQL バージョン 3

Context

`io/redo_log_flush` イベントは Aurora MySQL の書き込み入力/出力 (I/O) オペレーション用です。

> **Note**
>
> Aurora MySQL バージョン 2 では、この待機イベントの名前は io/aurora_redo_log_flush です。

待ち時間増加の考えられる原因

データの永続化のために、コミットはストレージが安定するよう耐久性の高い書き込みを要求とします。データベースがコミットを多くし過ぎると、書き込み I/O オペレーションで待機イベントが発生します、`io/redo_log_flush` 待機イベント。

この待機イベントの動作の例については、「io/aurora_redo_log_flush」を参照してください。

アクション

待機イベントの原因に応じて、異なるアクションをお勧めします。

問題のあるセッションとクエリを特定する

DB インスタンスにボトルネックが発生している場合、ユーザーの初期のタスクは、その原因となるセッションとクエリを見つけることになります。便利な AWS データベースブログ記事は Performance Insights を使用した Amazon Aurora MySQL ワークロードの分析 を参照してください。

ボトルネックの原因となっているセッションとクエリを特定するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。
4. データベース負荷で、待機でスライスを選択します。

5. ページの下部で トップ SQL を選択します。

   リストの上部にあるクエリは、データベースで最大の負荷を引き起こしています。

書き込みオペレーションをグループ化する

次の例は io/redo_log_flush 待機イベントをトリガーしています。(オートコミットがオンになっています。)

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
```

io/redo_log_flush 待機イベントで待機する時間を減らすため、書き込み操作を論理的に 1 つのコミットにグループ化し、ストレージへの永続的な呼び出しを減らします。

オートコミットをオフにする

次の例に示すように、トランザクション内に存在しない大きな変更を加える前に、オートコミットをオフにします。

```
SET SESSION AUTOCOMMIT=OFF;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
-- Other DML statements here
```

```
COMMIT;

SET SESSION AUTOCOMMIT=ON;
```

トランザクションの使用

次の例が示すように、トランサクションを使用することができます。

```
BEGIN
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;

-- Other DML statements here
END
```

バッチを使用する

次の例が示すように、バッチで変更することもできます。ただし、大きすぎるバッチを使用すると、特にリードレプリカやポイントインタイムリカバリ (PITR) の実行時にパフォーマンスの問題が発生する可能性があります。

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES
('xxxx','xxxxx'),('xxxx','xxxxx'),...,('xxxx','xxxxx'),('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1 BETWEEN xx AND
 xxx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1<xx;
```

io/socket/sql/client_connection

`io/socket/sql/client_connection` イベントは、スレッドが新しい接続を処理している最中に発生します。

トピック

- サポート対象エンジンバージョン
- Context
- 待ち時間増加の考えられる原因
- アクション

サポート対象エンジンバージョン

この待機イベント情報は、以下のエンジンバージョンでサポートされています。

- Aurora MySQL バージョン 2 および 3

Context

イベントio/socket/sql/client_connection は、受信する新規クライアント接続を処理するためのスレッド作成で mysqld がビジー状態であることを示します。このシナリオでは、スレッドが割り当てられるのを接続が待っている間、新しいクライアント接続リクエストの対応処理が遅くなります。詳細については、「MySQLサーバー (mysqld)」を参照してください。

待ち時間増加の考えられる原因

この イベントが通常よりも多く表示され、パフォーマンスの問題を示している可能性がある場合、典型的な原因は次のとおりです。

- アプリケーションから Amazon RDS インスタンスへの新しいユーザー接続が突然増加しています。
- ネットワーク、CPU、またはメモリがスロットリングされているため、DB インスタンスは新しい接続を処理できません。

アクション

io/socket/sql/client_connection がデータベースアクティビティを占領している場合でも、必ずしもパフォーマンスの問題を示すわけではありません。アイドル状態でないデータベースでは、待機イベントは常に最上位にあります。パフォーマンスが低下したときにのみ動作してください。待機イベントの原因に応じて、異なるアクションをお勧めします。

トピック

- 問題のあるセッションとクエリを特定する

- 接続管理のベストプラクティス
- リソースがスロットリングされている場合にインスタンスをスケールアップする
- 上位ホストと上位ユーザーを確認する
- performance_schema テーブルのクエリを実行する
- クエリのスレッド状態を確認する
- リクエストとクエリを監査する
- データベース接続をプールする

問題のあるセッションとクエリを特定する

DB インスタンスにボトルネックが発生している場合、ユーザーの初期のタスクは、その原因となるセッションとクエリを見つけることになります。便利なデータベースブログ記事は、Performance Insights を使用した Amazon Aurora MySQL ワークロードの分析を参照してください。

ボトルネックの原因となっているセッションとクエリを特定するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。
4. データベース負荷で、待機でスライスを選択します。
5. ページの下部で トップ SQL を選択します。

   リストの上部にあるクエリは、データベースで最大のロードを引き起こしています。

接続管理のベストプラクティス

接続を管理するには、次の戦略を検討してください。

- 接続プーリングの使用

  必要に応じて、接続数を徐々に増やすことができます。詳細については、ホワイトペーパーの Amazon Aurora MySQL データベース管理者ハンドブックを参照してください。
- リーダーノードを使用して、読み取り専用トラフィックを再配布します。

  詳細については、Aurora レプリカ および Amazon Aurora 接続管理 を参照してください。

リソースがスロットリングされている場合にインスタンスをスケールアップする

次のリソースでスロットリングの例を探してください。

- CPU

  Amazon CloudWatch メトリクスで CPU 使用率が高くなるかどうかを確認してください。
- ネットワーク

  CloudWatch メトリクス `network receive throughput` および `network transmit throughput` の値の増加を確認します。インスタンスがインスタンスクラスのネットワーク帯域幅制限に達した場合は、RDS インスタンスをより高いインスタンスクラスタイプにスケールアップすることを検討してください。詳細については、「Aurora DB インスタンスクラス」を参照してください。
- 解放可能なメモリ

  CloudWatch メトリクス `FreeableMemory` のドロップをチェックします。また、拡張モニタリングをオンにすることを検討してください。詳細については、「拡張モニタリングを使用した OS メトリクスのモニタリング」を参照してください。

上位ホストと上位ユーザーを確認する

Performance Insights を使用して、上位ホストと上位ユーザーを確認します。詳細については、「Performance Insights ダッシュボードを使用してメトリクスを分析する」を参照してください。

performance_schema テーブルのクエリを実行する

現在の接続数と合計接続数の正確な数を取得するには、`performance_schema` テーブルをクエリします。この手法では、多数の接続の作成を担当する出典ユーザーまたはホストを特定します。例えば、次の通り `performance_schema` テーブルをクエリします。

```
SELECT * FROM performance_schema.accounts;
SELECT * FROM performance_schema.users;
SELECT * FROM performance_schema.hosts;
```

クエリのスレッド状態を確認する

パフォーマンスの問題が継続する場合は、クエリのスレッド状態を確認してください。`mysql` クライアントで、次のコマンドを実行します。

```
show processlist;
```

リクエストとクエリを監査する

ユーザーアカウントからのリクエストとクエリの性質を確認するには、AuroraAurora MySQL アドバンスト監査を使用します。監査を有効にする方法については、Amazon Aurora MySQL DB クラスターでのアドバンストな監査の使用 を参照してください。

データベース接続をプールする

接続管理に Amazon RDS プロキシを使用することを検討してください。RDS プロキシーを使用すると、アプリケーションでデータベース接続をプールおよび共有して、アプリケーションのスケール能力を向上させることができます。RDS Proxy は、アプリケーション接続を維持しながらスタンバイ DB インスタンスに自動的に接続することで、データベースの障害に対するアプリケーションの耐障害性を高めます。詳細については、「Amazon RDS Proxy for Aurora の使用」を参照してください。

io/table/sql/handler

`io/table/sql/handler` イベントは、作業がストレージエンジンに委任されたときに発生します。

トピック

- サポート対象エンジンバージョン
- Context
- 待ち時間増加の考えられる原因
- アクション

サポート対象エンジンバージョン

この待機イベント情報は、以下のエンジンバージョンでサポートされています。

- Aurora MySQL バージョン 3: 3.01.0 および 3.01.1
- Aurora MySQL バージョン 2

Context

イベント `io/table` は、テーブルへのアクセスを待っていることを示します。このイベントは、データがバッファプールにキャッシュされているか、ディスク上でアクセスされているかにかかわら

ず、発生します。io/table/sql/handler イベントは、ワークロードアクティビティの増加を示します。

ハンドラーは、特定の種類のデータに特化したルーチンか、特定の特別なタスクに焦点を当てたルーチンです。例えば、イベントハンドラーは、オペレーティングシステムまたはユーザーインターフェイスからイベントとシグナルを受信してダイジェストします。メモリハンドラーは、メモリに関連するタスクを実行します。ファイル入力ハンドラは、ファイル入力を受け取り、コンテキストに応じてデータに対して特別なタスクを実行する関数です。

performance_schema.events_waits_current などの ビューは、実際の待機がロックなどのネストされた待機イベントである場合に io/table/sql/handler をよく表示します。実際の待機が io/table/sql/handler ではない場合、Performance Insights はネストされた待機イベントをレポートします。Performance Insights が io/table/sql/handler をレポートする場合、非表示のネストされた待機イベントではなく I/O リクエストの InnoDB 処理を表します。詳細については、MySQL リファレンスマニュアルの「パフォーマンススキーマの原子および分子イベント」を参照してください。

> **Note**
>
> ただし、Aurora MySQL バージョン 3.01.0 および 3.01.1 では、synch/mutex/innodb/aurora_lock_thread_slot_futex は io/table/sql/handler と見なされています。

io/table/sql/handler イベントは多くの場合、io/aurora_redo_log_flush や io/file/innodb/innodb_data_file のような I/O 待機を伴う上位待機イベントに表示されます。

待ち時間増加の考えられる原因

Performance Insights で、io/table/sql/handler イベントの急増はワークロードアクティビティの増加を示します。アクティビティの増加は、I/O が増加することを意味します。

Performance Insights はネストイベント ID をフィルタリングし、基盤となるネストされたイベントがロック待機である場合、io/table/sql/handler 待機をレポートします。例えば、根本原因イベントが synch/mutex/innodb/aurora_lock_thread_slot_futex である場合、Performance Insights は io/table/sql/handler ではなく 上位待機イベントにこの待機を表示します。

performance_schema.events_waits_current などのビューは、実際の待機がロックなどのネストされた待機イベントである場合によくio/table/sql/handler の待機を表示します。実際の待ち時間が io/table/sql/handler と異なる場合、Performance Insights はネストされた待機を検索し、io/table/sql/handler の代わりに実際の待機を報告します。Performance Insights が

io/table/sql/handler をレポートする場合、実際の待機は非表示のネストされた待機イベントではなく、io/table/sql/handler になります。詳細については、「MySQL 5.7 リファレンスマニュアル」の「パフォーマンススキーマのアトムおよび分子イベント」を参照してください。

> **Note**
>
> ただし、Aurora MySQL バージョン 3.01.0 および 3.01.1 では、synch/mutex/innodb/aurora_lock_thread_slot_futex は io/table/sql/handler と見なされています。

アクション

待機イベントがデータベースアクティビティを占領している場合でも、必ずしもパフォーマンスの問題を示すわけではありません。データベースがアクティブな場合、待機イベントは常に最上位になります。パフォーマンスが低下したときにのみ動作してください。

確認できる他の待機イベントに応じて、異なるアクションをお勧めします。

トピック

- イベントの原因となるセッションとクエリを特定する
- Performance Insights カウンター指標との相関関係をチェックする
- 他の相関待ちイベントがないかチェックする

イベントの原因となるセッションとクエリを特定する

通常、ロードが中程度から重大なデータベースには、待機イベントがあります。パフォーマンスが最適であれば、待機イベントは受け入れられる可能性があります。パフォーマンスが最適でない場合は、データベースが最も長い時間を費やしている場所を調べます。最も高いロードに寄与する待機イベントを調べて、データベースとアプリケーションを最適化してこれらのイベントを削減できるかどうかを調べます。

高ロードの原因となる SQL クエリを検索するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。その DB インスタンスの Performance Insights ダッシュボードが表示されます。

4. データベースロードで、待機でスライスを選択します。
5. ページの下部で トップ SQL を選択します。

   グラフには、ロードの原因となる SQL クエリがリストされます。リスト上部にあるクエリに、最も責任があります。ボトルネックを解決するには、これらのステートメントに注目してください。

Performance Insights を使用したトラブルシューティングの便利な概要については、ブログ記事 Performance Insights を使用した Amazon Aurora MySQL ワークロードの分析 を参照してください。

Performance Insights カウンター指標との相関関係をチェックする

`Innodb_rows_changed` などの Performance Insights カウンターメトリクスをチェックします。カウンタメトリックが `io/table/sql/handler` と相関している場合、以下のステップを実行します。

1. Performance Insights で、`io/table/sql/handler` トップ待機イベントの原因になっている SQL ステートメントを探します。可能であれば、このステートメントを最適化して、返される行数を減らします。
2. `schema_table_statistics` と `x$schema_table_statistics` ビューからトップテーブルを取得します。これらのビューには、テーブルごとに費やされた時間が表示されます。詳細については、MySQL リファレンスマニュアルの schema_table_statistics と x$schema_table_statistics ビュー を参照してください。

   デフォルトでは、行は合計待機時間の降順でソートされます。競合が最も多いテーブルが初期に表示されます。出力は、読み取り、書き込み、フェッチ、挿入、更新、または削除に時間を費やしているかどうかを示します。次の例は Aurora MySQL 2.09.1 インスタンスで実行されました。

```
mysql> select * from sys.schema_table_statistics limit 1\G

*************************** 1. row ***************************
     table_schema: read_only_db
       table_name: sbtest41
    total_latency: 54.11 m
     rows_fetched: 6001557
    fetch_latency: 39.14 m
    rows_inserted: 14833
   insert_latency: 5.78 m
```

```
     rows_updated: 30470
   update_latency: 5.39 m
     rows_deleted: 14833
   delete_latency: 3.81 m
 io_read_requests: NULL
          io_read: NULL
  io_read_latency: NULL
io_write_requests: NULL
         io_write: NULL
 io_write_latency: NULL
 io_misc_requests: NULL
  io_misc_latency: NULL
1 row in set (0.11 sec)
```

他の相関待ちイベントがないかチェックする

`synch/sxlock/innodb/btr_search_latch` と `io/table/sql/handler` が共に DB ロードの異常に最も貢献している場合、`innodb_adaptive_hash_index` 可変がオンになっているかチェックします。もしそうであれば、`innodb_adaptive_hash_index_parts` パラメータ値を増やすことを検討します。

Adaptive Hash インデックスがオフになっている場合、オンにすることを検討します。MySQL adaptive Hash インデックスの詳細については、以下のリソースを参照してください。

- Percona Web サイトの記事「InnoDB の Adaptive Hash Index は私のワークロードに適していますか？」
- MySQL リファレンスマニュアルのアダプティブハッシュインデックス
- Percona ウェブサイトの MySQL InnoDB での競合: セマフォセクションからの有用な情報の記事

> **Note**
>
> Adaptive Hash インデックスは Aurora Reader DB インスタンスではサポートされていません。
> `synch/sxlock/innodb/btr_search_latch` と `io/table/sql/handler` が支配的な場合、リーダーインスタンスのパフォーマンスが低下することがあります。その場合は、ワークロードをライター DB インスタンスに一時的にリダイレクトし、Adaptive Hash インデックスをオンにすることを検討してください。

synch/cond/innodb/row_lock_wait

`synch/cond/innodb/row_lock_wait` イベントは、あるセッションが更新のために行をロックし、別のセッションが同じ行を更新しようとしたときに発生します。詳細については、MySQL リファレンスの InnoDB ロックを参照してください。

サポート対象エンジンバージョン

この待機イベント情報は、以下のエンジンバージョンでサポートされています。

- Aurora MySQL バージョン 3: 3.02.0、3.02.1、3.02.2

待機時間が増加する原因の可能性

複数のデータ操作言語 (DML) ステートメントが同じ行に同時にアクセスしようとしています。

アクション

確認できる他の待機イベントに応じて、異なるアクションをお勧めします。

トピック

- この待機イベントを担当する SQL 文を見つけて応答します。
- ブロッキングセッションを見つけて対応する

この待機イベントを担当する SQL 文を見つけて応答します。

Performance Insights を使用して、この待機イベントの原因になっている SQL ステートメントを特定します。以下の戦略を検討して下さい。

- 行のロックが永続的な問題である場合は、オプティミスティックロックを使用するようにアプリケーションを書き直すことを検討してください。
- 複数行のステートメントの使用。
- ワークロードを異なるデータベースオブジェクトに分散します。パーティショニングによって、これを行うことができます。
- `innodb_lock_wait_timeout` パラメータの値をチェックしてください。これは、タイムアウトエラーを生成する前にトランザクションが待機する時間を制御します。

Performance Insights を使用したトラブルシューティングの便利な概要については、ブログ記事 Performance Insights を使用した Amazon Aurora MySQL ワークロードの分析 を参照してください。

ブロッキングセッションを見つけて対応する

ブロッキングセッションがアイドルかアクティブかを確認します。また、セッションがアプリケーションからかのものか、またはアクティブユーザーからのものであるかを調べます。

ロックを保持しているセッションを識別するには、SHOW ENGINE INNODB STATUS を実行します。次の例は サンプル出力を示しています。

```
mysql> SHOW ENGINE INNODB STATUS;

---TRANSACTION 1688153, ACTIVE 82 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 2 row lock(s)
MySQL thread id 4244, OS thread handle 70369524330224, query id 4020834 172.31.14.179
 reinvent executing
select id1 from test.t1 where id1=1 for update
------- TRX HAS BEEN WAITING 24 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 11 page no 4 n bits 72 index GEN_CLUST_INDEX of table test.t1 trx
 id 1688153 lock_mode X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 5; compact format; info bits 0
```

または、次のクエリを使用して、現在のロックの詳細を抽出することもできます。

```
mysql> SELECT p1.id waiting_thread,
    p1.user waiting_user,
    p1.host waiting_host,
    it1.trx_query waiting_query,
    ilw.requesting_engine_transaction_id waiting_transaction,
    ilw.blocking_engine_lock_id blocking_lock,
    il.lock_mode blocking_mode,
    il.lock_type blocking_type,
    ilw.blocking_engine_transaction_id blocking_transaction,
    CASE it.trx_state
        WHEN 'LOCK WAIT'
        THEN it.trx_state
        ELSE p.state end blocker_state,
    concat(il.object_schema,'.', il.object_name) as locked_table,
    it.trx_mysql_thread_id blocker_thread,
```

```
    p.user blocker_user,
    p.host blocker_host
FROM performance_schema.data_lock_waits ilw
JOIN performance_schema.data_locks il
ON ilw.blocking_engine_lock_id = il.engine_lock_id
AND ilw.blocking_engine_transaction_id = il.engine_transaction_id
JOIN information_schema.innodb_trx it
ON ilw.blocking_engine_transaction_id = it.trx_id join information_schema.processlist p
ON it.trx_mysql_thread_id = p.id join information_schema.innodb_trx it1
ON ilw.requesting_engine_transaction_id = it1.trx_id join
 information_schema.processlist p1
ON it1.trx_mysql_thread_id = p1.id\G

*************************** 1. row ***************************
waiting_thread: 4244
waiting_user: reinvent
waiting_host: 123.456.789.012:18158
waiting_query: select id1 from test.t1 where id1=1 for update
waiting_transaction: 1688153
blocking_lock: 70369562074216:11:4:2:70369549808672
blocking_mode: X
blocking_type: RECORD
blocking_transaction: 1688142
blocker_state: User sleep
locked_table: test.t1
blocker_thread: 4243
blocker_user: reinvent
blocker_host: 123.456.789.012:18156
1 row in set (0.00 sec)
```

セッションを特定する際、次のようなオプションが含まれます。

- アプリケーションの所有者またはユーザーにお問い合わせください。
- ブロッキングセッションがアイドル状態の場合は、ブロッキングセッションを終了することを検討してください。このアクションは、長いロールバックをトリガーする可能性があります。セッションの終了方法については、「セッションやクエリの終了」を参照してください。

ブロックトランザクションの識別の詳細については、MySQL リファレンスマニュアルの InnoDB トランザクションの使用とロック情報を参照してください。

synch/cond/innodb/row_lock_wait_cond

`synch/cond/innodb/row_lock_wait_cond` イベントは、あるセッションが更新のために行を口ックし、別のセッションが同じ行を更新しようとしたときに発生します。詳細については、MySQL リファレンスの InnoDB 口ックを参照してください。

サポート対象エンジンバージョン

この待機イベント情報は、以下のエンジンバージョンでサポートされています。

- Aurora MySQL バージョン 2

待機時間が増加する原因の可能性

複数のデータ操作言語 (DML) ステートメントが同じ行に同時にアクセスしようとしています。

アクション

確認できる他の待機イベントに応じて、異なるアクションをお勧めします。

トピック

- この待機イベントを担当する SQL 文を見つけて応答します。
- ブロッキングセッションを見つけて対応する

この待機イベントを担当する SQL 文を見つけて応答します。

Performance Insights を使用して、この待機イベントの原因になっている SQL ステートメントを特定します。以下の戦略を検討して下さい。

- 行の口ックが永続的な問題である場合は、オプティミスティック口ックを使用するようにアプリケーションを書き直すことを検討してください。
- 複数行のステートメントの使用。
- ワークロードを異なるデータベースオブジェクトに分散します。パーティショニングによって、これを行うことができます。
- `innodb_lock_wait_timeout` パラメータの値をチェックしてください。これは、タイムアウトエラーを生成する前にトランザクションが待機する時間を制御します。

Performance Insights を使用したトラブルシューティングの便利な概要については、ブログ記事 Performance Insights を使用した Amazon Aurora MySQL ワークロードの分析 を参照してください。

ブロッキングセッションを見つけて対応する

ブロッキングセッションがアイドルかアクティブかを確認します。また、セッションがアプリケーションからかのものか、またはアクティブユーザーからのものであるかを調べます。

ロックを保持しているセッションを識別するには、SHOW ENGINE INNODB STATUS を実行します。次の例は サンプル出力を示しています。

```
mysql> SHOW ENGINE INNODB STATUS;

---TRANSACTION 2771110, ACTIVE 112 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 1 row lock(s)
MySQL thread id 24, OS thread handle 70369573642160, query id 13271336 172.31.14.179
 reinvent Sending data
select id1 from test.t1 where id1=1 for update
------- TRX HAS BEEN WAITING 43 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 11 page no 3 n bits 0 index GEN_CLUST_INDEX of table test.t1 trx
 id 2771110 lock_mode X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 5; compact format; info bits 0
```

または、次のクエリを使用して、現在のロックの詳細を抽出することもできます。

```
mysql> SELECT p1.id waiting_thread,
              p1.user waiting_user,
              p1.host waiting_host,
              it1.trx_query waiting_query,
              ilw.requesting_trx_id waiting_transaction,
              ilw.blocking_lock_id blocking_lock,
              il.lock_mode blocking_mode,
              il.lock_type blocking_type,
              ilw.blocking_trx_id blocking_transaction,
              CASE it.trx_state
                WHEN 'LOCK WAIT'
                THEN it.trx_state
                ELSE p.state
              END blocker_state,
              il.lock_table locked_table,
```

```
               it.trx_mysql_thread_id blocker_thread,
               p.user blocker_user,
               p.host blocker_host
        FROM information_schema.innodb_lock_waits ilw
        JOIN information_schema.innodb_locks il
          ON ilw.blocking_lock_id = il.lock_id
         AND ilw.blocking_trx_id = il.lock_trx_id
        JOIN information_schema.innodb_trx it
          ON ilw.blocking_trx_id = it.trx_id
        JOIN information_schema.processlist p
          ON it.trx_mysql_thread_id = p.id
        JOIN information_schema.innodb_trx it1
          ON ilw.requesting_trx_id = it1.trx_id
        JOIN information_schema.processlist p1
          ON it1.trx_mysql_thread_id = p1.id\G

*************************** 1. row ***************************
      waiting_thread: 3561959471
        waiting_user: reinvent
        waiting_host: 123.456.789.012:20485
       waiting_query: select id1 from test.t1 where id1=1 for update
 waiting_transaction: 312337314
       blocking_lock: 312337287:261:3:2
       blocking_mode: X
       blocking_type: RECORD
blocking_transaction: 312337287
       blocker_state: User sleep
        locked_table: `test`.`t1`
      blocker_thread: 3561223876
        blocker_user: reinvent
        blocker_host: 123.456.789.012:17746
1 row in set (0.04 sec)
```

セッションを特定する際、次のようなオプションが含まれます。

- アプリケーションの所有者またはユーザーにお問い合わせください。
- ブロッキングセッションがアイドル状態の場合は、ブロッキングセッションを終了することを検討してください。このアクションは、長いロールバックをトリガーする可能性があります。セッションの終了方法については、「セッションやクエリの終了」を参照してください。

ブロックトランザクションの識別の詳細については、MySQL リファレンスマニュアルの InnoDB トランザクションの使用とロック情報を参照してください。

synch/cond/sql/MDL_context::COND_wait_status

synch/cond/sql/MDL_context::COND_wait_status イベントは、テーブルメタデータロックに待機中のスレッドがあるときに発生します。

トピック

サポート対象エンジンバージョン

この待機イベント情報は、以下のエンジンバージョンでサポートされています。

- Aurora MySQL バージョン 2 および 3

Context

イベント synch/cond/sql/MDL_context::COND_wait_status は、テーブルメタデータロックを待機中のスレッドがあることを示します。場合によっては、あるセッションがテーブルのメタデータロックを保持し、別のセッションが同じテーブルで同じロックを取得しようとする場合があります。このような場合、2 番目のセッションは synch/cond/sql/MDL_context::COND_wait_status 待機イベントで待機します。

MySQL は、メタデータロックを使用して、データベースオブジェクトへの同時アクセスを管理し、データの整合性を確保します。メタデータのロックは、get_lock 関数で取得したスケジュールされたイベント、テーブルスペース、ユーザーロックテーブル、および ストアドプログラムに適用されます。ストアドプログラムには、プロシージャ、関数、トリガーが含まれます。詳細については、MySQL ドキュメントのメタデータロックを参照してください。

MySQL プロセスリストは、このセッションを状態 waiting for metadata lock で表示されます。Performance Insights では、Performance_schema がオンになっている場合、イベント synch/cond/sql/MDL_context::COND_wait_status が表示されます。

メタデータロックで待っているクエリのデフォルトタイムアウトは lock_wait_timeout パラメータ値に基づきます。デフォルトは 31,536,000 秒 (365 日) です。

さまざまな InnoDB ロックと競合を引き起こす可能性のある ロックの種類の詳細については、MySQL ドキュメントの InnoDB ロックを参照してください。

待ち時間増加の考えられる原因

`synch/cond/sql/MDL_context::COND_wait_status` イベントが通常よりも多く表示され、パフォーマンスの問題を示している可能性がある場合、典型的な原因は次のとおりです。

実行時間が長いトランザクション

1 つ以上のトランザクションが大量のデータを変更し、非常に長い間テーブルのロックを保持しています。

Idle トランザクション

1 つ以上のトランザクションは、コミットまたはロールバックされることなく、長い間開いたままです。

大きなテーブルの DDL ステートメント

1 つ以上のデータ定義ステートメント (DDL) ステートメント。`ALTER TABLE`コマンドは、非常に大きなテーブルで実行されました。

明示的なテーブルロック

テーブルには、タイムリーに解放されない明示的なロックがあります。例えば、アプリケーションが実行されているとします。`LOCK TABLE`不適切にステートメント。

アクション

待機イベントの原因と Aurora MySQL DB クラスターのバージョンに応じて、さまざまなアクションをお勧めします。

トピック

- イベントの原因となるセッションとクエリを特定する
- 過去のイベントをチェックする
- Aurora MySQL バージョン 2 でクエリを実行する
- ブロッキングセッションに応答する

イベントの原因となるセッションとクエリを特定する

Performance Insights を使用して、`synch/cond/sql/MDL_context::COND_wait_status` 待機イベントによってブロックされたクエリを表示できます。ただし、ブロッキングセッションを特定するには、からメタデータテーブルをクエリします。`performance_schema`そして`information_schema`DB クラスター上にあります。

通常、ロードが中程度から重大なデータベースには、待機イベントがあります。パフォーマンスが最適であれば、待機イベントは受け入れられる可能性があります。パフォーマンスが最適でない場合は、データベースが最も長い時間を費やしている場所を調べます。最も高いロードに寄与する待機イベントを調べて、データベースとアプリケーションを最適化してこれらのイベントを削減できるかどうかを調べます。

高ロードの原因となる SQL クエリを検索するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。この DB インスタンスに Performance Insights ダッシュボードが表示されます。
4. データベースロードで、待機でスライスを選択します。
5. ページの下部で トップ SQL を選択します。

   グラフには、ロードの原因となる SQL クエリがリストされます。リスト上部にあるクエリに、最も責任があります。ボトルネックを解決するには、これらのステートメントに注目してください。

Performance Insights を使用したトラブルシューティングの便利な概要については、AWS　データベースブログ記事の Performance Insights を使用した Amazon Aurora MySQL ワークロードの分析を参照してください。

過去のイベントをチェックする

この待機イベントについてのインサイトを取得して、過去のイベントが発生していないかどうかをチェックできます。そのためには、以下のアクションを実行します。

- データ操作言語 (DML) と DDL のスループットとレイテンシーを調べて、ワークロードに変更があったかどうかをチェックします。

Performance Insights を使用して、問題発生時にこのイベントで待っているクエリを検索できます。また、発行時に実行されたクエリのダイジェストを表示することもできます。

- DB クラスターの監査ログまたは一般ログがオンになっている場合、待機中のトランザクションに含まれるオブジェクト (schema.table) で実行されるすべてのクエリをチェックできます。また、トランザクションの前に実行が完了したクエリをチェックすることもできます。

過去のイベントのトラブルシューティングに使用できる情報は限られています。これらのチェックを実行しても、どのオブジェクトが情報を待っているのかは示されません。ただし、イベント時にロードが大きいテーブルや、発行時に競合の原因となる頻繁に操作される一連の行を特定できます。この情報を使用して、テスト環境で問題を再現し、その原因に関するインサイトを使用できます。

Aurora MySQL バージョン 2 でクエリを実行する

Aurora MySQL バージョン 2 の場合、performance_schema テーブルないし sys ビューに対してクエリを実行して、ブロックされたセッションを直接識別できます。。例は、ブロックしているクエリとセッションを識別するため、テーブルにクエリを実行する方法を示しています。

次のプロセスリストの出力では、接続 ID 89 がメタデータロックを待っていて、TRUNCATE TABLE コマンドを実行しています。performance_schema テーブルないし sys スキーマビューのクエリの場合、出力が表示するブロッキングセッションは 76 です。

```
MySQL [(none)]> select @@version, @@aurora_version;
+-----------+------------------+
| @@version | @@aurora_version |
+-----------+------------------+
| 5.7.12    | 2.09.0           |
+-----------+------------------+
1 row in set (0.01 sec)

MySQL [(none)]> show processlist;
+----+-----------------+--------------------+-----------+---------+------
+---------------------------------+------------------------------+
| Id | User            | Host               | db        | Command | Time | State
                    | Info                           |
+----+-----------------+--------------------+-----------+---------+------
+---------------------------------+------------------------------+
|  2 | rdsadmin        | localhost          | NULL      | Sleep   |    0 | NULL
                    | NULL                           |
|  4 | rdsadmin        | localhost          | NULL      | Sleep   |    2 | NULL
                    | NULL                           |
```

```
|  5 | rdsadmin        | localhost          | NULL      | Sleep   |    1 | NULL
                    | NULL                             |
| 20 | rdsadmin        | localhost          | NULL      | Sleep   |    0 | NULL
                    | NULL                             |
| 21 | rdsadmin        | localhost          | NULL      | Sleep   |  261 | NULL
                    | NULL                             |
| 66 | auroramysql5712 | 172.31.21.51:52154 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                             |
| 67 | auroramysql5712 | 172.31.21.51:52158 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                             |
| 68 | auroramysql5712 | 172.31.21.51:52150 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                             |
| 69 | auroramysql5712 | 172.31.21.51:52162 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                             |
| 70 | auroramysql5712 | 172.31.21.51:52160 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                             |
| 71 | auroramysql5712 | 172.31.21.51:52152 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                             |
| 72 | auroramysql5712 | 172.31.21.51:52156 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                             |
| 73 | auroramysql5712 | 172.31.21.51:52164 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                             |
| 74 | auroramysql5712 | 172.31.21.51:52166 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                             |
| 75 | auroramysql5712 | 172.31.21.51:52168 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                             |
| 76 | auroramysql5712 | 172.31.21.51:52170 | NULL      | Query   |    0 | starting
                    | show processlist                 |
| 88 | auroramysql5712 | 172.31.21.51:52194 | NULL      | Query   |   22 | User sleep
                    | select sleep(10000)              |
| 89 | auroramysql5712 | 172.31.21.51:52196 | NULL      | Query   |    5 | Waiting for
 table metadata lock | truncate table sbtest.sbtest1 |
+----+-----------------+--------------------+-----------+---------+------
+---------------------------------+------------------------------+
18 rows in set (0.00 sec)
```

次に、performance_schema テーブルないし sys スキーマビューのクエリはブロッキングセッションが 76 であることを示します。

```
MySQL [(none)]> select * from sys.schema_table_lock_waits;

+---------------+-------------+-------------------+-------------
+-------------------------------+-------------------+-----------------------
```

```
+-------------------------------+--------------------+-----------------------------
+-----------------------------+--------------------+-------------
+-------------------------------+--------------------+-----------------------
+-------------------------+-----------------------------+
| object_schema | object_name | waiting_thread_id | waiting_pid | waiting_account
          | waiting_lock_type | waiting_lock_duration | waiting_query
   | waiting_query_secs | waiting_query_rows_affected | waiting_query_rows_examined |
 blocking_thread_id | blocking_pid | blocking_account             | blocking_lock_type
 | blocking_lock_duration | sql_kill_blocking_query | sql_kill_blocking_connection |
+---------------+-------------+-------------------+-------------
+-------------------------------+-------------------+-----------------------
+-------------------------------+--------------------+-----------------------------
+-----------------------------+--------------------+-------------
+-------------------------------+--------------------+-----------------------
+-------------------------+-----------------------------+
| sbtest        | sbtest1     |               121 |          89 |
 auroramysql5712@192.0.2.0    | EXCLUSIVE         | TRANSACTION           | truncate
 table sbtest.sbtest1 |                 10 |                           0 |
               0 |               108 |           76 | auroramysql5712@192.0.2.0    |
 SHARED_READ        | TRANSACTION            | KILL QUERY 76           | KILL 76
              |
+---------------+-------------+-------------------+-------------
+-------------------------------+-------------------+-----------------------
+-------------------------------+--------------------+-----------------------------
+-----------------------------+--------------------+-------------
+-------------------------------+--------------------+-----------------------
+-------------------------+-----------------------------+
1 row in set (0.00 sec)
```

ブロッキングセッションに応答する

セッションを特定する際、次のようなオプションが含まれます。

- アプリケーションの所有者またはユーザーにお問い合わせください。
- ブロッキングセッションがアイドル状態の場合は、ブロッキングセッションを終了することを検討してください。このアクションは、長いロールバックをトリガーする可能性があります。セッションの終了方法については、「セッションやクエリの終了」を参照してください。

ブロックトランザクションの識別の詳細については、MySQL ドキュメントのInnoDB トランザクションの使用と情報のロックを参照してください。

synch/mutex/innodb/aurora_lock_thread_slot_futex

`synch/mutex/innodb/aurora_lock_thread_slot_futex` イベントは、あるセッションが更新のために行をロックし、別のセッションが同じ行を更新しようとしたときに発生します。詳細については、MySQL リファレンスの InnoDB ロックを参照してください。

サポート対象エンジンバージョン

この待機イベント情報は、以下のエンジンバージョンでサポートされています。

- Aurora MySQL バージョン 2

> **Note**
>
> Aurora MySQL バージョン 3.01.0 および 3.01.1 では、この待機イベントは io/table/sql/handler と見なされています。

待機時間が増加する原因の可能性

複数のデータ操作言語 (DML) ステートメントが同じ行に同時にアクセスしようとしています。

アクション

確認できる他の待機イベントに応じて、異なるアクションをお勧めします。

トピック

- この待機イベントを担当する SQL 文を見つけて応答します。
- ブロッキングセッションを見つけて対応する

この待機イベントを担当する SQL 文を見つけて応答します。

Performance Insights を使用して、この待機イベントの原因になっている SQL ステートメントを特定します。以下の戦略を検討して下さい。

- 行のロックが永続的な問題である場合は、オプティミスティックロックを使用するようにアプリケーションを書き直すことを検討してください。
- 複数行のステートメントの使用。

- ワークロードを異なるデータベースオブジェクトに分散します。パーティショニングによって、これを行うことができます。
- innodb_lock_wait_timeout パラメータの値をチェックしてください。これは、タイムアウトエラーを生成する前にトランザクションが待機する時間を制御します。

Performance Insights を使用したトラブルシューティングの便利な概要については、ブログ記事 Performance Insights を使用した Amazon Aurora MySQL ワークロードの分析 を参照してください。

ブロッキングセッションを見つけて対応する

ブロッキングセッションがアイドルかアクティブかを確認します。また、セッションがアプリケーションからかのものか、またはアクティブユーザーからのものであるかを調べます。

ロックを保持しているセッションを識別するには、SHOW ENGINE INNODB STATUS を実行します。次の例は サンプル出力を示しています。

```
mysql> SHOW ENGINE INNODB STATUS;

----------------------TRANSACTION 302631452, ACTIVE 2 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 376, 1 row lock(s)
MySQL thread id 80109, OS thread handle 0x2ae915060700, query id 938819 10.0.4.12
 reinvent updating
UPDATE sbtest1 SET k=k+1 WHERE id=503
------- TRX HAS BEEN WAITING 2 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 148 page no 11 n bits 30 index `PRIMARY` of table
 `sysbench2`.`sbtest1` trx id 302631452 lock_mode X locks rec but not gap waiting
Record lock, heap no 30 PHYSICAL RECORD: n_fields 6; compact format; info bits 0
```

または、次のクエリを使用して、現在のロックの詳細を抽出することもできます。

```
mysql> SELECT p1.id waiting_thread,
              p1.user waiting_user,
              p1.host waiting_host,
              it1.trx_query waiting_query,
              ilw.requesting_trx_id waiting_transaction,
              ilw.blocking_lock_id blocking_lock,
              il.lock_mode blocking_mode,
              il.lock_type blocking_type,
```

```
              ilw.blocking_trx_id blocking_transaction,
              CASE it.trx_state
                WHEN 'LOCK WAIT'
                THEN it.trx_state
                ELSE p.state
              END blocker_state,
              il.lock_table locked_table,
              it.trx_mysql_thread_id blocker_thread,
              p.user blocker_user,
              p.host blocker_host
         FROM information_schema.innodb_lock_waits ilw
         JOIN information_schema.innodb_locks il
           ON ilw.blocking_lock_id = il.lock_id
          AND ilw.blocking_trx_id = il.lock_trx_id
         JOIN information_schema.innodb_trx it
           ON ilw.blocking_trx_id = it.trx_id
         JOIN information_schema.processlist p
           ON it.trx_mysql_thread_id = p.id
         JOIN information_schema.innodb_trx it1
           ON ilw.requesting_trx_id = it1.trx_id
         JOIN information_schema.processlist p1
           ON it1.trx_mysql_thread_id = p1.id\G

*************************** 1. row ***************************
      waiting_thread: 3561959471
        waiting_user: reinvent
        waiting_host: 123.456.789.012:20485
       waiting_query: select id1 from test.t1 where id1=1 for update
 waiting_transaction: 312337314
       blocking_lock: 312337287:261:3:2
       blocking_mode: X
       blocking_type: RECORD
blocking_transaction: 312337287
       blocker_state: User sleep
        locked_table: `test`.`t1`
      blocker_thread: 3561223876
        blocker_user: reinvent
        blocker_host: 123.456.789.012:17746
1 row in set (0.04 sec)
```

セッションを特定する際、次のようなオプションが含まれます。

- アプリケーションの所有者またはユーザーにお問い合わせください。

- ブロッキングセッションがアイドル状態の場合は、ブロッキングセッションを終了することを検討してください。このアクションは、長いロールバックをトリガーする可能性があります。セッションの終了方法については、「セッションやクエリの終了」を参照してください。

ブロックトランザクションの識別の詳細については、MySQL リファレンスマニュアルの InnoDB トランザクションの使用とロック情報を参照してください。

synch/mutex/innodb/buf_pool_mutex

`synch/mutex/innodb/buf_pool_mutex` イベントは、スレッドがメモリ内のページにアクセスするために InnoDB バッファプールのロックを取得したときに発生します。

トピック

- 関連するエンジンバージョン
- Context
- 待ち時間増加の考えられる原因
- アクション

関連するエンジンバージョン

この待機イベント情報は、以下のエンジンバージョンでサポートされています。

- Aurora MySQL バージョン 2

Context

`buf_pool` ミューテックスは、バッファプールの制御データ構造を保護する単一のミューテックスです。

詳細については、MySQL ドキュメントのパフォーマンススキーマを使用した InnoDB Mutex 待機をモニタリングするを参照してください。

待ち時間増加の考えられる原因

これは、ワークロード固有の待機イベントです。`synch/mutex/innodb/buf_pool_mutex` がトップ待機イベント内に出現する一般的な原因は、次のような場合が含まれます。

- バッファープールのサイズが、ワーキングデータセットを保持するのに十分な大きさではない。

- ワークロードが、データベース内の特定のテーブルの特定のページに固有であり、バッファプール内に競合が発生している。

アクション

待機イベントの原因に応じたさまざまなアクションをお勧めします。

イベントの原因となるセッションとクエリを特定する

通常、ロードが中程度から重大なデータベースには、待機イベントがあります。パフォーマンスが最適であれば、待機イベントは受け入れられる可能性があります。パフォーマンスが最適でない場合は、データベースが最も長い時間を費やしている場所を調べます。最も高いロードに寄与する待機イベントを調べて、データベースとアプリケーションを最適化してこれらのイベントを削減できるかどうかを調べます。

AWSマネジメントコンソールの トップ SQL チャートを表示するには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。その DB インスタンスの Performance Insights ダッシュボードが表示されます。
4. データベースロードで、待機でスライスを選択します。
5. データベースロードグラフの下で、トップ SQL を選択します。

   グラフには、ロードの原因となる SQL クエリがリストされます。リスト上部にあるクエリに、最も責任があります。ボトルネックを解決するには、これらのステートメントに注目してください。

Performance Insights を使用したトラブルシューティングの便利な概要については、ブログ記事 Performance Insights を使用した Amazon Aurora MySQL ワークロードの分析 を参照してください。

Performance Insights の使用

このイベントはワークロードに関連しています。Performance Insights を使用して、以下を実行できます。

- 待機イベントがスタートされたタイミングと、その時点でアプリケーションログまたは関連出典からのワークロードに変化があったかどうかを特定します。

- この待機イベントの原因になっている SQL ステートメントを特定します。クエリの実行プランを調べて、これらのクエリが最適化され、適切なインデックスが使用されていることを確認します。

  待機イベントの原因となる上位クエリが同じデータベースオブジェクトまたはテーブルに関連付けられている場合は、そのオブジェクトまたはテーブルをパーティション化することを検討してください。

Aurora レプリカの作成

Aurora レプリカを作成して、読み取り専用トラフィックを処理できます。Aurora オートスケーリングを使用して、読み取りトラフィックのサージを処理することもできます。Aurora レプリカでスケジュールされた読み取り専用タスクと論理的なバックアップを実行してください。

詳細については、「Aurora レプリカでの Amazon Aurora Auto Scaling の使用」を参照してください。

バッファープールサイズの検査

メトリクス `innodb_buffer_pool_wait_free` を確認して、バッファ・プール・サイズがワークロードに十分かどうかをチェックします。このメトリクスの値が高く継続的に増加している場合は、バッファ・プールのサイズがワークロードを処理するのに十分でないことを示します。`innodb_buffer_pool_size` が適切に設定されている場合、`innodb_buffer_pool_wait_free` の値は小さくなります。詳細については、MySQL ドキュメントのInnodb_buffer_pool_wait_freeを参照してください。

DB インスタンス上に、セッションバッファとオペレーティングシステムのタスクに十分なメモリがある場合は、バッファプールサイズを増やします。そうでない場合は、DB インスタンスを大きな DB インスタンスクラスに変更して、バッファプールに割り当てられる追加のメモリを取得します。

> **Note**
>
> Aurora MySQL は、構成された `innodb_buffer_pool_size` に基づいて`innodb_buffer_pool_instances` の値を自動的に調整します

グローバルステータス履歴をモニタリングする

ステータス可変の変更率をモニタリングすることで、DB インスタンスのロックまたはメモリの問題を検出できます。グローバルステータス履歴 (GoSH) がまだオンになっていない場合は、オンにします。GoSH の詳細については、グローバルステータス履歴の管理を参照してください。

カスタムの Amazon CloudWatch メトリクスを作成して、ステータス可変をモニタリングすることもできます。詳細については、カスタムメトリクスの発行を参照してください。

synch/mutex/innodb/fil_system_mutex

synch/mutex/innodb/fil_system_mutex イベントは、セッションがテーブルスペースのメモリキャッシュへのアクセスを待っているときに発生します。

トピック

- サポート対象エンジンバージョン
- Context
- 待ち時間増加の考えられる原因
- アクション

サポート対象エンジンバージョン

この待機イベント情報は、以下のエンジンバージョンでサポートされています。

- Aurora MySQL バージョン 2 および 3

Context

InnoDB はテーブルスペースを使用して、テーブルとログファイルのストレージ領域を管理します。テーブルスペースのメモリキャッシュは、テーブルスペースに関する情報を保持するグローバル・メモリ構造です。MySQL は synch/mutex/innodb/fil_system_mutex 待機を使用して、テーブルスペースのメモリー・キャッシュへの同時アクセスを制御します。

イベント synch/mutex/innodb/fil_system_mutex は、テーブルスペースメモリーキャッシュにある情報を、同じテーブルスペースに対して取得または操作しないといけないオペレーションが現在複数存在することを示します。

待ち時間増加の考えられる原因

synch/mutex/innodb/fil_system_mutex イベントが通常よりも多く表示され、パフォーマンスの問題を示している可能性がある場合、典型的な原因は次のとおりです。

- 同じテーブル内のデータを更新または削除する、同時データ操作言語 (DML) オペレーションの増加。

- このテーブルのテーブルスペースは非常に大きく、多くのデータページがあります。
- これらのデータページの塗りつぶし係数は低いです。

アクション

待機イベントの原因に応じたさまざまなアクションをお勧めします。

トピック

- イベントの原因となるセッションとクエリを特定する
- オフピーク時に大きなテーブルを再編成する

イベントの原因となるセッションとクエリを特定する

通常、ロードが中程度から重大なデータベースには、待機イベントがあります。パフォーマンスが最適であれば、待機イベントは受け入れられる可能性があります。パフォーマンスが最適でない場合は、データベースが最も長い時間を費やしている場所を調べます。最も高いロードに寄与する待機イベントを調べて、データベースとアプリケーションを最適化してこれらのイベントを削減できるかどうかを調べます。

高ロードの原因となる SQL クエリを検索するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。この DB インスタンスに Performance Insights ダッシュボードが表示されます。
4. データベースロードで、待機でスライスを選択します。
5. ページの下部で トップ SQL を選択します。

   グラフには、ロードの原因となる SQL クエリがリストされます。リスト上部にあるクエリに、最も責任があります。ボトルネックを解決するには、これらのステートメントに注目してください。

Performance Insights を使用したトラブルシューティングの便利な概要については、ブログ記事 Performance Insights を使用した Amazon Aurora MySQL ワークロードの分析 を参照してください。

どのクエリが多数のsynch/mutex/innodb/fil_system_mutex 待機原因になっているかを調べる別の方法は、、以下の例のように performance_schema をチェックする方法です。

```
mysql> select * from performance_schema.events_waits_current where EVENT_NAME='wait/
synch/mutex/innodb/fil_system_mutex'\G
*************************** 1. row ***************************
            THREAD_ID: 19
             EVENT_ID: 195057
         END_EVENT_ID: 195057
           EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
               SOURCE: fil0fil.cc:6700
          TIMER_START: 1010146190118400
            TIMER_END: 1010146196524000
           TIMER_WAIT: 6405600
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
     NESTING_EVENT_ID: NULL
   NESTING_EVENT_TYPE: NULL
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: NULL
*************************** 2. row ***************************
            THREAD_ID: 23
             EVENT_ID: 5480
         END_EVENT_ID: 5480
           EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
               SOURCE: fil0fil.cc:5906
          TIMER_START: 995269979908800
            TIMER_END: 995269980159200
           TIMER_WAIT: 250400
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
     NESTING_EVENT_ID: NULL
   NESTING_EVENT_TYPE: NULL
            OPERATION: lock
```

```
      NUMBER_OF_BYTES: NULL
                FLAGS: NULL
*************************** 3. row ***************************
            THREAD_ID: 55
             EVENT_ID: 23233794
         END_EVENT_ID: NULL
           EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
               SOURCE: fil0fil.cc:449
          TIMER_START: 1010492125341600
            TIMER_END: 1010494304900000
           TIMER_WAIT: 2179558400
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
     NESTING_EVENT_ID: 23233786
   NESTING_EVENT_TYPE: WAIT
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: NULL
```

オフピーク時に大きなテーブルを再編成する

本番時間外のメンテナンスウィンドウで、多数の synch/mutex/innodb/fil_system_mutex 待機イベントの出典として識別するラージテーブルを再編成する。これにより、テーブルへのクイックアクセスが必修な際に、内部テーブルスペースマップのクリーンアップが行われないようにします。テーブルの再編成については、MySQL リファレンスの TABLE ステートメントの最適化を参照してください。

synch/mutex/innodb/trx_sys_mutex

synch/mutex/innodb/trx_sys_mutex イベントは、大量のトランザクションで高いデータベースアクティビティがある場合に発生します。

トピック

- 関連するエンジンバージョン
- Context
- 待ち時間増加の考えられる原因
- アクション

関連するエンジンバージョン

この待機イベント情報は、以下のエンジンバージョンでサポートされています。

- Aurora MySQL バージョン 2 および 3

Context

内部的には、InnoDB データベースエンジンは、繰り返し可能な読み取り分離レベルとスナップショットを使用して、読み取りの整合性を提供します。これにより、スナップショットが作成された時点でのデータベースのポイントインタイムビューが表示されます。

InnoDB では、コミットされているかどうかにかかわらず、すべての変更が到着するとすぐにデータベースに適用されます。このアプローチは、マルチバージョン同時実行制御 (MVCC) を使用しないと、データベースに接続されているすべてのユーザーに、すべての変更と最新の行が表示されることを意味します。したがって、InnoDB ではロールバックする内容を理解するために、変更を追跡する方法が必要に応じて必要です。

これを行うために、InnoDB はトランザクションシステム (`trx_sys`) を使用してスナップショットを追跡します。トランザクションシステムは、以下を実行します。

- 元に戻すログの各行に対するトランザクション ID を追跡します。
- `ReadView` 呼ばれる内部 InnoDB 構造体を使用すると、スナップショットで表示されるトランザクション ID の特定を容易に行えます。

待ち時間増加の考えられる原因

一貫性のある制御されたトランザクション ID 処理 (作成、読み取り、更新、および削除) を必要とする全てのデータベースオペレーションは、`trx_sys` からミューテックスに呼び出しを行います。

これらの呼び出しは、次の 3 つの関数内で発生します。

- `trx_sys_mutex_enter` - ミューテックスを作成する。
- `trx_sys_mutex_exit` - ミューテックスを解放する。
- `trx_sys_mutex_own` - ミューテックスが所有されているかどうかをテストする。

InnoDB パフォーマンススキーマインストルメンテーションは、すべての `trx_sys` ミューテックスの呼び出しを追跡します。追跡には管理が含まれますが、これらに限定されません。`trx_sys`データベースの起動時またはシャットダウン、ロールバック操作、クリーンアップの取り消し、行の読

み取りアクセス、およびバッファプールのロード。トランザクション数が多く、データベースアクティビティが高い場合、synch/mutex/innodb/trx_sys_mutex は上位の待機イベント中に現れます。

詳細については、MySQL ドキュメントのパフォーマンススキーマを使用した InnoDB Mutex 待機をモニタリングするを参照してください。

アクション

待機イベントの原因に応じたさまざまなアクションをお勧めします。

イベントの原因となるセッションとクエリを特定する

通常、ロードが中程度から重大なデータベースには、待機イベントがあります。パフォーマンスが最適であれば、待機イベントは受け入れられる可能性があります。パフォーマンスが最適でない場合は、データベースが最も長い時間を費やしている場所を調べます。最も高いロードに寄与する待機イベントを見てください。これらのイベントを減らすためにデータベースとアプリケーションを最適化できるかどうかを調べます。

AWS Management Console で上位 SQL チャートを表示するには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。その DB インスタンスの Performance Insights ダッシュボードが表示されます。
4. データベースロードで、待機でスライスを選択します。
5. データベースロードグラフで、上位の SQLを選択します。

   グラフには、ロードの原因となる SQL クエリがリストされます。リスト上部にあるクエリに、最も責任があります。ボトルネックを解決するには、これらのステートメントに注目してください。

Performance Insights を使用したトラブルシューティングの便利な概要については、ブログ記事 Performance Insights を使用した Amazon Aurora MySQL ワークロードの分析 を参照してください。

他の待機イベントを検査する

synch/mutex/innodb/trx_sys_mutex 待機イベントに関連する他の待機イベントを調べます。これにより、ワークロードの性質に関する詳細情報が提供されます。トランザクションの数が多い

とスループットが低下する可能性がありますが、ワークロードによってはこれが必要な場合もあります。

トランザクションを最適化する方法の詳細については、MySQL ドキュメントの InnoDB トランザクション管理の最適化を参照してください。

synch/sxlock/innodb/hash_table_locks

`synch/sxlock/innodb/hash_table_locks` イベントは、バッファプール内に見つからないページをストレージから読み込む必要があるときに発生します。

トピック

- サポート対象エンジンバージョン
- Context
- 待ち時間増加の考えられる原因
- アクション

サポート対象エンジンバージョン

この待機イベント情報は、以下のバージョンでサポートされています:

- Aurora MySQL バージョン 2 および 3

Context

イベント `synch/sxlock/innodb/hash_table_locks` は、ワークロードがバッファプールに保存されていないデータに頻繁にアクセスしていることを示します。この待機イベントは、バッファプールからの新しいページの追加と古いデータの削除に関連付けられます。バッファプールに格納されたデータは、古いデータおよび新しいデータをキャッシュする必要があります。そのため、古いページは削除され、新しいページのキャッシュが可能になります。MySQL は、バッファプールからページを削除するために、最近一番使用されていない (LRU) アルゴリズムを使用します。ワークロードは、バッファプールにロードされていないデータ、またはバッファプールから削除されたデータにアクセスしようとしています。

この待機イベントは、ワークロードがディスク上のファイル内のデータにアクセスする必要がある場合、またはブロックがバッファプールの LRU リストから解放または追加されたときに発生します。これらのオペレーションは、共有除外ロック (SX-Lock) の取得を待ちます。この SX-Lock は、ハッ

シュテーブル上での同期化に使われます。これは、バッファプールのアクセスパフォーマンスを向上させるために設計されたメモリ内のテーブルです。

詳細については、MySQL ドキュメントのバッファプールを参照してください。

待ち時間増加の考えられる原因

synch/sxlock/innodb/hash_table_locks 待機イベントが通常よりも多く表示され、パフォーマンスの問題を示している可能性がある場合、典型的な原因は次のとおりです。

サイズの小さいバッファープール

バッファプールのサイズが小さすぎて、頻繁にアクセスされるすべてのページをメモリ内に保持できません。

ヘビーワークロード

ワークロードによって、バッファキャッシュで頻繁にエビクションとデータページがリロードされます。

ページの読み取り中にエラーが発生しました

バッファープール内のページの読み取り中にエラーが発生しました。これは、データの破損を示している可能性があります。

アクション

待機イベントの原因に応じて、異なるアクションをお勧めします。

トピック

- バッファプールのサイズを増やす
- データアクセスパターンの改善
- フルテーブルスキャンの削減または回避
- エラーログでページの破損を確認します。

バッファプールのサイズを増やす

バッファプールがワークロードに合わせて適切なサイズになっていることを確認します。そのためには、バッファプールのキャッシュヒット率を確認できます。通常、値が 95% を下回った場合は、バッファプールのサイズを大きくすることを検討してください。バッファプールが大きいほど、

頻繁にアクセスされるページをメモリ内に長く保持できます。バッファプールのサイズを増やすには、innodb_buffer_pool_size パラメータの値を変更します。このパラメータのデフォルト値は、DB インスタンスクラスのサイズに基づいています。詳細については、Amazon Aurora MySQL データベース設定のベストプラクティスを参照してください。

データアクセスパターンの改善

この待機の影響を受けるクエリとその実行プランを確認します。データアクセスパターンの改善を検討してください。例えばmysqli_result:: fetch_array を使用している場合には、配列のフェッチサイズを大きくしてみるなどが可能です。

Performance Insights を使用して、synch/sxlock/innodb/hash_table_locks 待機イベントの原因となっている可能性のあるクエリとセッションを表示できます。

高ロードの原因となる SQL クエリを検索するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Performance Insights] を選択します。
3. DB インスタンスを選択します。その DB インスタンスの Performance Insights ダッシュボードが表示されます。
4. データベースロードで、待機でスライスを選択します。
5. ページの下部で トップ SQL を選択します。

   グラフには、ロードの原因となる SQL クエリがリストされます。リスト上部にあるクエリに、最も責任があります。ボトルネックを解決するには、これらのステートメントに注目してください。

Performance Insights を使用したトラブルシューティングの便利な概要については、AWS　データベースブログ記事の Performance Insights を使用した Amazon Aurora MySQL ワークロードの分析を参照してください。

フルテーブルスキャンの削減または回避

ワークロードをモニタリングして、フルテーブルスキャンが実行されているかどうかを確認し、実行している場合はそれを減らすか回避します。例えば、Handler_read_rnd_next のようなステータス可変をモニタリングできます。詳細については、MySQL ドキュメントのサーバーステータス可変を参照してください。

エラーログでページの破損を確認します。

mysql-error.log をチェックして、問題の発生時に検出された破損関連のメッセージを確認できます。問題を解決するために作業できるメッセージは、エラーログに記録されます。破損として報告されたオブジェクトを再作成する必要がある場合があります。

スレッド状態を使用した Aurora MySQL のチューニング

次の表は Aurora MySQL の最も一般的なスレッド状態をまとめたものです。

一般的なスレッド状態	説明
???	このスレッド状態は、データをソートするために内部テンポラリテーブルを使用する必要がある SELECT ステートメントを、スレッドが処理中であることを示します。
???	このスレッド状態は、スレッドがクエリの行を読み取り、フィルタリングして、正しい結果セットを判断していることを示します。

ソートインデックスの作成

`creating sort index` スレッド状態は、データをソートするために内部テンポラリテーブルを使用する必要がある SELECT ステートメントを、スレッドが処理中であることを示します。

トピック

- サポート対象エンジンバージョン
- Context
- 待ち時間増加の考えられる原因
- アクション

サポート対象エンジンバージョン

このスレッド状態情報は、以下のバージョンでサポートされています。

- Aurora MySQL バージョン 2 から 2.09.2 まで

Context

`creating sort index` 状態は、`ORDER BY` および `GROUP BY` 句を持つクエリが既存のインデックスを使用して操作を実行できない場合に表示されます。この場合、MySQL はより高価な

filesort オペレーションを実行する必要があります. 通常、この操作は結果セットが大きすぎない場合にメモリ内で実行されます。それ以外の場合は、ディスク上にファイルを作成する必要があります。

待ち時間増加の考えられる原因

creating sort index の表示は、それ自体が問題を示しているわけではありません。パフォーマンスが低下し、頻繁に creating sort index の症例が表示される場合、最も可能性の高い原因は ORDER BY または GROUP BY 演算子を使った遅いクエリです。

アクション

一般的なガイドラインは、creating sort index 状態の増加と関連付けられる ORDER BY と GROUP BY 句を持つクエリを見つけることです。次に、インデックスを追加するか、ソートバッファサイズを大きくしても問題が解決するかどうかを確認します。

トピック

- パフォーマンススキーマ がオンになっていない場合は、オンにします。
- 問題のあるクエリを特定する
- ファイルソートの使用に関する説明プランを調べる
- ソートバッファサイズを増やす

パフォーマンススキーマ がオンになっていない場合は、オンにします。

Performance Insights は、パフォーマンススキーマインストゥルメントがオンになっていない場合にのみ、スレッドの状態を報告します。パフォーマンススキーマインストゥルメントがオンの場合、Performance Insights は代わりに待機イベントをレポートします。パフォーマンススキーマインストゥルメントは、潜在的なパフォーマンスの問題を調査するための、追加のインサイトと優れたツールを提供します。したがって、パフォーマンススキーマをオンにすることをお勧めします。詳細については、「Aurora MySQL における Performance Insights の Performance Schema の有効化」を参照してください。

問題のあるクエリを特定する

creating sort index 状態の増加の原因となっている現在のクエリを特定するには、show processlist を実行して ORDER BY または GROUP BY を持つクエリがないか確認します。任意で、filesort を持つクエリのプロセスリスト ID に N がなっている場所で、explain for connection N を実行します。

これらの増加の原因となっている過去のクエリを特定するには、スロークエリログをオンにして、ORDER BY のクエリを検索します。遅いクエリでEXPLAINを実行し、「filesort を使用しています。」を探します。詳細については、「ファイルソートの使用に関する説明プランを調べる」を参照してください。

ファイルソートの使用に関する説明プランを調べる

creating sort index状態を引き起こすORDER BY と GROUP BY 句を持つステートメントを特定する。

次の例は、クエリで explain を実行する方法を解説しています。Extra 列は、このクエリが filesort を使用していることを示しています。

```
mysql> explain select * from mytable order by c1 limit 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: mytable
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 2064548
     filtered: 100.00
        Extra: Using filesort
1 row in set, 1 warning (0.01 sec)
```

次の例は、c1 カラムにインデックスが作成された後、同じクエリで EXPLAIN を実行した結果を示しています。

```
mysql> alter table mytable add index (c1);
```

```
mysql> explain select * from mytable order by c1 limit 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: mytable
   partitions: NULL
         type: index
possible_keys: NULL
```

```
          key: c1
      key_len: 1023
          ref: NULL
         rows: 10
     filtered: 100.00
        Extra: Using index
1 row in set, 1 warning (0.01 sec)
```

ソート順の最適化にインデックスを使用する方法については、MySQL ドキュメントの最適化による注文を参照してください。

ソートバッファサイズを増やす

特定のクエリで、ディスク上にファイルを作成した filesort プロセスが必要かどうかを確認するには、クエリの実行後、sort_merge_passes 可変値をチェックします。例を以下に示します。

```
mysql> show session status like 'sort_merge_passes';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| Sort_merge_passes | 0     |
+-------------------+-------+
1 row in set (0.01 sec)

--- run query
mysql> select * from mytable order by u limit 10;
--- run status again:

mysql> show session status like 'sort_merge_passes';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| Sort_merge_passes | 0     |
+-------------------+-------+
1 row in set (0.01 sec)
```

sort_merge_passes の値が高い場合は、ソートバッファサイズを大きくすることを検討してください。この増加をセッションレベルで適用します。グローバルに増やすと、RAM MySQL の使用量が大幅に増加する可能性があるためです。次の例は、クエリを実行する前にソートバッファサイズを変更する方法を示しています。

```
mysql> set session sort_buffer_size=10*1024*1024;
```

```
Query OK, 0 rows affected (0.00 sec)
-- run query
```

データの送信

sending data スレッド状態は、スレッドがクエリの行を読み取り、フィルタリングして、正しい結果セットを決定していることを示します。名前が誤解を招くのは、状態がデータを転送しており、後で送信するデータを収集して準備していないことを意味するためです。

トピック

- サポート対象エンジンバージョン
- Context
- 待ち時間増加の考えられる原因
- アクション

サポート対象エンジンバージョン

このスレッド状態情報は、以下のバージョンでサポートされています。

- Aurora MySQL バージョン 2 から 2.09.2 まで

Context

多くのスレッド状態は短続きします。中に発生するオペレーションsending data大量のディスクまたはキャッシュ読み取りを実行する傾向があります。したがって、sending data多くの場合、特定のクエリの存続期間中で最も長い実行状態です。この状態は、Aurora MySQL が次のことをしているときに表示されます。

- の口ーの読み取りと処理SELECTステートメント
- ディスクまたはメモリから大量の読み取りを実行する
- 特定のクエリからのすべてのデータの完全な読み取りを完了する
- テーブル、インデックス、またはストアドプロシージャの作業からデータを読み込む
- データのソート、グループ化、または順序付け

Aftersending datastate はデータの準備を終了し、スレッドの状態writing to netは、クライアントにデータを返すことを示します。通常、writing to netは、結果セットが非常に大きい

か、または深刻なネットワークレイテンシーによって転送が遅くなる場合にのみキャプチャされます。

待ち時間増加の考えられる原因

sending data の表示は、それ自体が問題を示しているわけではありません。パフォーマンスが低下し、頻繁に次のインスタンスが表示される場合sending dataの場合、最も可能性の高い原因は以下のとおりです。

トピック

- 非効率なクエリ
- 最適でないサーバー設定

非効率なクエリ

ほとんどの場合、この状態の原因となるのは、特定のクエリの結果セットを見つけるために適切なインデックスを使用していないクエリです。例えば、カリフォルニアで行われたすべての注文について 1,000 万件のレコードテーブルを読み取るクエリについて考えてみます。ここでは、州の列にインデックスが付けられていないか、インデックスが不十分です。後者の場合、インデックスは存在する可能性がありますが、カーディナリティが低いためオプティマイザはそれを無視します。

最適でないサーバー設定

複数のクエリがsending data状態の場合、データベースサーバーの設定が不十分である可能性があります。具体的には、サーバーには次の問題が発生する可能性があります。

- データベースサーバーには、ディスク I/O、ディスクタイプと速度、CPU、または CPU の数など、十分なコンピューティング性能がありません。
- InnoDB テーブルの InnoDB バッファプールや MyISAM テーブルのキーバッファなど、割り当てられたリソースでサーバーが不足しています。
- スレッドごとのメモリ設定 (など)sort_buffer、read_buffer、 およびjoin_buffer必要以上に多くの RAM を消費し、物理サーバのメモリリソースが不足しています。

アクション

一般的なガイドラインは、パフォーマンススキーマをチェックして、多数の行を返すクエリを検索することです。インデックスを使用しないクエリのログがオンの場合、スローログの結果を調べることもできます。

トピック

- パフォーマンススキーマ がオンになっていない場合は、オンにします。
- メモリ設定の確認
- インデックスの使用状況に関する説明プランを調べる
- 返されたデータの量をチェックする
- 並行性の問題をチェックする
- クエリの構文を確認します。

パフォーマンススキーマ がオンになっていない場合は、オンにします。

Performance Insights は、パフォーマンススキーマインストゥルメントがオンになっていない場合にのみ、スレッドの状態を報告します。パフォーマンススキーマインストゥルメントがオンの場合、Performance Insights は代わりに待機イベントをレポートします。パフォーマンススキーマインストゥルメントは、潜在的なパフォーマンスの問題を調査するための、追加のインサイトと優れたツールを提供します。したがって、パフォーマンススキーマをオンにすることをお勧めします。詳細については、「Aurora MySQL における Performance Insights の Performance Schema の有効化」を参照してください。

メモリ設定の確認

プライマリバッファプールのメモリ設定を確認します。これらのプールがワークロードに合わせて適切なサイズになっていることを確認します。データベースで複数のバッファプールインスタンスを使用している場合は、それらが多数の小さなバッファプールに分割されていないことを確認してください。スレッドが一度に使用できるバッファプールは 1 つだけです。

各スレッドで使用される次のメモリ設定が、適切なサイズであることを確認します。

- read_buffer
- read_rnd_buffer
- sort_buffer
- join_buffer
- binlog_cache

設定を変更する特定の理由がない限り、デフォルト値を使用します。

インデックスの使用状況に関する説明プランを調べる

sending data スレッド状態のクエリの場合、プランを調べて、適切なインデックスが使用されているかどうかを判断します。クエリが有用なインデックスを使用していない場合は、USE INDEX や FORCE INDEX のようなヒントを追加することを検討してください。ヒントは、クエリの実行にかかる時間を大幅に増減できるので、追加する前に注意してください。

返されたデータの量をチェックする

クエリ対象のテーブルと、そのテーブルに含まれるデータの量をチェックします。このデータのいずれかをアーカイブできますか? 多くの場合、クエリの実行時間が短くなる原因は、クエリプランの結果ではなく、処理されるデータの量です。多くのデベロッパーはデータベースにデータを追加するのに非常に効率的ですが、設計および開発段階でデータセットのライフサイクルを考慮することはほとんどありません。

低ボリュームデータベースではうまく機能するが、現在のシステムではパフォーマンスが低下するクエリを探します。特定のクエリを設計するデベロッパーは、これらのクエリが 350,000 行を返していることに気付かないことがあります。デベロッパーは、実稼働環境よりも小さいデータセットを持つ低ボリュームの環境でクエリを開発した可能性があります。

並行性の問題をチェックする

同じタイプの複数のクエリが同時に実行されているかどうかを確認します。一部の形式のクエリは、単独で実行すると効率的に実行されます。ただし、同様の形式のクエリを一緒に、または大量に実行すると、同時実行の問題が発生する可能性があります。多くの場合、これらの問題はデータベースがテンポラリテーブルを使用して結果をレンダリングするときに発生します。制限的なトランザクション分離レベルは、同時実行性の問題を引き起こすこともあります。

テーブルが同時に読み書きされる場合、データベースがロックを使用している可能性があります。低いパフォーマンス期間を特定するには、大規模なバッチプロセスでデータベースの使用状況を調べます。最近のロックとロールバックを確認するには、SHOW ENGINE INNODB STATUS コマンドの出力を確認します。

クエリの構文を確認します。

これらの状態からキャプチャされたクエリがサブクエリを使用しているかどうかを確認します。このタイプのクエリは、データベースが内部的に結果をコンパイルし、それらを再びクエリに戻してデータレンダリングするため、パフォーマンスが低下することがあります。このプロセスは、データベースの追加ステップです。多くの場合このステップは、高い同時ロード条件下でパフォーマンスが低下する可能性があります。

また大量の ORDER BY および GROUP BY句 をクエリが使用していないかも、確認してください。このような操作では、多くの場合、データベースはまずデータセット全体をメモリ内に形成する必要があります。その後、クライアントに戻す前に、特定の方法で注文またはグループ化する必要があります。

Amazon DevOps Guru のプロアクティブインサイトによる Aurora MySQL のチューニング

DevOps Guru のプロアクティブインサイトは、Aurora MySQL DB クラスターで既知の問題が発生する前に検出します。DevOps Guru では、次のことができます。

- データベース構成を一般的な推奨設定と照合することで、データベースに関する多くの一般的な問題を防ぎます。
- 未チェックのままにしておくと、後で大きな問題につながる可能性があるフリート内の重大な問題について警告します。
- 新しく発見された問題について警告します。

すべてのプロアクティブインサイトには、問題の原因の分析と是正措置の推奨事項が含まれています。

トピック

- InnoDB 履歴リストの長さが大幅に増加しました
- データベースはディスク上にテンポラリテーブルを作成している

InnoDB 履歴リストの長さが大幅に増加しました

date を過ぎると、行変更の履歴リストが大幅に増加し、*db-instance* では最大 *length* になりました。この増加は、クエリとデータベースのシャットダウンパフォーマンスに影響します。

トピック

- サポート対象エンジンバージョン
- Context
- この問題の考えられる原因
- アクション
- 関連するメトリクス

サポート対象エンジンバージョン

このインサイト情報は、Aurora MySQL のすべてのバージョンでサポートされています。

Context

InnoDB トランザクションシステムは、マルチバージョン同時実行制御 (MVCC) を維持します。行が変更されると、変更中のデータの修正前のバージョンが、undo レコードとして undo ログに保存されます。すべての undo レコードには、以前の redo レコードへの参照があり、リンクリストを形成します。

InnoDB 履歴リストは、コミットされたトランザクションの undo ログのグローバルリストです。MySQL は、トランザクションで履歴が不要になったときに、履歴リストを使用してレコードとログページを削除します。履歴リストの長さは、履歴リスト内の変更を含む undo ログの総数です。各ログには、1 つ以上の変更が含まれます。InnoDB 履歴リストの長さが大きくなりすぎると、古い行バージョンが多数存在することになり、クエリやデータベースのシャットダウンが遅くなります。

この問題の考えられる原因

履歴リストが長くなる一般的な原因には次のものがあります。

- 実行時間の長いトランザクション (読み取りまたは書き込み)
- 書き込み負荷が高い

アクション

インサイトの原因に応じて、異なるアクションをお勧めします。

トピック

- InnoDB 履歴リストが減るまで、データベースのシャットダウンを伴う操作を開始しない
- 長時間実行されるトランザクションを特定して終了する
- Performance Insights を使用して、上位ホストと上位ユーザーを特定します。

InnoDB 履歴リストが減るまで、データベースのシャットダウンを伴う操作を開始しない

InnoDB 履歴リストが長くなるとデータベースのシャットダウンが遅くなるため、データベースシャットダウンを伴う操作を開始する前にリストサイズを小さくしてください。これらの操作には、データベースのメジャーバージョンのアップグレードが含まれます。

長時間実行されるトランザクションを特定して終了する

`information_schema.innodb_trx` クエリを実行すると、実行時間の長いトランザクションを見つけることができます。

> **Note**
>
> リードレプリカで長時間実行されるトランザクションも必ず探してください。

長時間実行されるトランザクションを特定して終了するには

1. SQL クライアントで次のクエリを実行します。

```
SELECT a.trx_id,
      a.trx_state,
      a.trx_started,
      TIMESTAMPDIFF(SECOND,a.trx_started, now()) as "Seconds Transaction Has Been
 Open",
      a.trx_rows_modified,
      b.USER,
      b.host,
      b.db,
      b.command,
      b.time,
      b.state
FROM  information_schema.innodb_trx a,
      information_schema.processlist b
WHERE a.trx_mysql_thread_id=b.id
  AND TIMESTAMPDIFF(SECOND,a.trx_started, now()) > 10
ORDER BY trx_started
```

2. 実行時間の長いトランザクションのそれぞれを `COMMIT` または `ROLLBACK` コマンドで終了します。

Performance Insights を使用して、上位ホストと上位ユーザーを特定します。

トランザクションを最適化して、変更された多数の行がすぐにコミットされるようにします。

関連するメトリクス

このインサイトに関連するメトリクスは次のとおりです。

- trx_rseg_history_len

  詳細については、「MySQL 5.7 Reference Manual」(MySQL 5.7 リファレンスマニュアル) の「InnoDB INFORMATION_SCHEMA Metrics Table」(InnoDB INFORMATION_SCHEMA メトリクステーブル) を参照してください。

データベースはディスク上にテンポラリテーブルを作成している

最近のディスク上のテンポラリテーブルの使用率が大幅に増加し、最大 *percentage* に達しています。データベースは、1 秒あたり約 *number* 個のテンポラリテーブルを作成しています。これにより、パフォーマンスに影響が及び、*db-instance* に対するディスク操作が増える可能性があります。

トピック

- サポート対象エンジンバージョン
- Context
- この問題の考えられる原因
- アクション
- 関連するメトリクス

サポート対象エンジンバージョン

このインサイト情報は、Aurora MySQL のすべてのバージョンでサポートされています。

Context

MySQL サーバーがクエリの処理中に内部一時テーブルを作成する必要がある場合があります。Aurora MySQL は、内部一時テーブルをメモリに保持できます。このテーブルは、TempTable または MEMORY ストレージエンジンで処理するか、InnoDB によってディスクに保存したりできます。詳細については、MySQL リファレンスマニュアルの「Internal Temporary Table Use in MySQL」(MySQL での内部一時テーブルの使用) を参照してください。

この問題の考えられる原因

ディスク上の一時テーブルの増加は、複雑なクエリの使用を示しています。設定されたメモリが一時テーブルをメモリに格納するには不十分な場合、Aurora MySQL はテーブルをディスク上に作成します。これにより、パフォーマンスに影響が及び、ディスク操作が増える可能性があります。

アクション

インサイトの原因に応じて、異なるアクションをお勧めします。

- Aurora MySQL バージョン 3 の場合、TempTable ストレージエンジンを使用することをお勧めします。
- 必要な列のみを選択して、クエリを最適化して、返されるデータを減らします。

  すべての `statement` 計測が有効で時間制限のある状態でパフォーマンススキーマを有効にすると、`SYS.statements_with_temp_tables` クエリを実行して、一時テーブルを使用するクエリのリストを取得できます。詳細については、MySQL ドキュメントの「Prerequisites for Using the sys Schema」(sys スキーマを使用するための前提条件) を参照してください。
- ソートやグループ化の操作に関係する列にインデックスを付けることを検討してください。
- `BLOB` および `TEXT` 列を避けるように、クエリを書き直します。これらの列は常にディスクを使用します。
- `tmp_table_size` および `max_heap_table_size` データベースパラメータをチューニングします。

  これらのパラメータのデフォルト値は 16 MiB です。メモリ内一時テーブルに MEMORY ストレージエンジンを使用する場合、最大サイズは、`tmp_table_size` または `max_heap_table_size` 値のいずれか小さい方によって定義されます。この最大サイズに達すると、MySQL はインメモリ内部一時テーブルを InnoDB オンディスク内部一時テーブルに自動的に変換します。詳細については、「Use the TempTable storage engine on Amazon RDS for MySQL and Amazon Aurora MySQL」(Amazon RDS for MySQL および Amazon Aurora MySQL で TempTable ストレージ エンジンを使用する) を参照してください。

> **Note**
>
> CREATE TABLE を使用して MEMORY テーブルを明示的に作成する場合、テーブルをどれだけ大きくできるかを決めるのは `max_heap_table_size` 変数だけです。また、オンディスク形式への変換もありません。

関連するメトリクス

以下の Performance Insights メトリクスがこのインサイトに関連しています。

- Created_tmp_disk_tables

- Created_tmp_tables

詳細については、MySQL ドキュメントの「Created_tmp_disk_tables」を参照してください。

Amazon Aurora MySQL のパラレルクエリの使用

このトピックでは、Amazon Aurora MySQL 互換エディションのパラレルクエリの最適化機能について説明しています。この機能は、特定のデータ集約型クエリに対して特別な処理パスを使用し、Aurora 共有ストレージアーキテクチャを利用します。パラレルクエリは、数百万行のテーブルと完了までに数分または数時間かかる分析クエリを持つテーブルを持つ Aurora MySQL DB クラスターで最も効果的です。

目次

- SQL 構造でのパラレルクエリの動作
  - EXPLAIN ステートメント
  - WHERE 句
  - データ定義言語 (DDL)
  - 列のデータ型
  - パーティションテーブル
  - 集計関数、GROUP BY 句、HAVING 句
  - WHERE 句での関数呼び出し
  - LIMIT 句
  - 比較演算子
  - Joins
  - サブクエリ
  - UNION
  - ビュー
  - データ操作言語 (DML) ステートメント
  - トランザクションとロック
  - B ツリーインデックス
  - 全文検索 (FTS) インデックス
  - 仮想列
  - 組み込みキャッシュメカニズム
  - オプティマイザヒント
  - MyISAM テンポラリテーブル

Aurora MySQL の並列クエリの概要

Aurora MySQL パラレルクエリは、データ集約的なクエリの処理に関連する I/O と計算の一部をパラレル処理する最適化です。パラレル処理される作業には、ストレージから行を取得し、列の値を抽出して、WHERE 句と結合句の条件に一致する行を判別することが含まれます。このデータ集約型の作業は、Aurora 分散ストレージレイヤー内の複数のノードに委譲されます (データベース最適化の用語では、プッシュダウンされます)。並行クエリがないと、各クエリはすべてのスキャンされたデータを Aurora MySQL クラスター (ヘッドノード) 内の単一のノードに持ち込み、そこですべてのクエリ処理を実行します。

> **Tip**
>
> PostgreSQL データベースエンジンにも「パラレルクエリ」と呼ばれる機能があります。この機能は、Aurora のパラレルクエリとは異なります。

パラレルクエリ特徴を有効にすると、ヒントやテーブル属性などの SQL の変更を必要とせずに、Aurora MySQL エンジンはクエリがいつ利点を得られるかを自動的に判断します。以降のセクションで、パラレルクエリがいつクエリに適用されるかについて説明します。パラレルクエリが最も効果的な場所に適用されていることを確認する方法についても説明します。

> **Note**
>
> パラレルクエリの最適化は、完了までに数分や数時間といった長い時間がかかるクエリの場合に最もメリットがあります。Aurora MySQL は、一般的に安価なクエリのためにはパラレルクエリの最適化を実行しません。また、クエリのキャッシュ、バッファプールのキャッシュ、インデックスの検索などの別の最適化手法を使用したほうが効果的な場合も、通常はパラレルクエリの最適化を行いません。パラレルクエリが適切に使用されていない場合は、「パラレルクエリを使用しているステートメントの確認」を参照してください。

トピック

利点

パラレルクエリを使用すると、Aurora MySQL のテーブルに対してデータ集約型の分析クエリを実行できます。多くの場合、従来のようにクエリの処理を分ける場合よりもパフォーマンスが大幅に向上します。

パラレルクエリを使用すると、次のような利点があります。

- 複数のストレージノード間で物理的な読み取りリクエストをパラレル処理するため、I/O パフォーマンスが向上しました。
- ネットワークトラフィックの削減。Aurora は、ストレージノードからヘッドノードへデータページ全体を送信せず、その後不要な行および列をフィルタリングで除去します。代わりに、Aurora は、結果セットに必要な列値のみを含むコンパクトなタプルを送信します。
- 関数の処理、行のフィルタリング、および WHERE 句の列射影をプッシュダウンすることにより、ヘッドノードの CPU 使用率を削減しました。
- バッファプールのメモリ負荷が低減されました。パラレルクエリによって処理されたページは、バッファプールに追加されません。これにより、データ集約型のスキャンで、頻繁に使用されるデータがバッファプールから削除されることが少なくなります。
- 既存のデータに対して長時間実行される分析クエリを実用的に実行することで、抽出、変換、ロード (ETL) パイプラインでのデータ重複を潜在的に削減します。

アーキテクチャ

パラレルクエリ機能は、Aurora MySQL の主なアーキテクチャ原則を使用して、ストレージサブシステムからデータベースエンジンをデカップリング、通信プロトコルを効率化してネットワークトラフィックを削減します。Aurora MySQL は、これらの技術を使用して、REDO ログ処理などの書き込み負荷の高いオペレーションを高速化します。パラレルクエリは、同じ原則を読み取り操作に適用します。

> **Note**
>
> Aurora MySQL パラレルクエリのアーキテクチャは、他のデータベースシステムで同様の名前を持つ機能のアーキテクチャとは異なります。Aurora MySQL のパラレルクエリは、対称型マルチプロセッシング (SMP) を利用しないため、データベースサーバーの CPU 容量に依存しません。パラレル処理は、クエリコーディネーターとして機能する Aurora MySQL サーバーとは独立して、ストレージレイヤーで行われます。

デフォルトでは、パラレルクエリを使用しない場合の Aurora のクエリ処理では、Aurora クラスター内の単一のノード (ヘッドノード) に raw データが送られます。Aurora は、その単一ノード上の単一のスレッドで、その後のクエリの処理をすべて実行します。パラレルクエリでは、この I/O 集約型および CPU 集約型の作業の多くは、ストレージレイヤー内のノードに委譲されます。結果セットのコンパクトな行のみがヘッドノードに戻されます (行は既にフィルタリングされ、列の値は既に抽出され、変換されています)。パフォーマンスの利点は、ネットワークトラフィックの削減、ヘッド

ノードでの CPU 使用率の削減、ストレージノード間の I/O のパラレル化から得られます。パラレル I/O、フィルタリング、および射影の量は、クエリを実行する Aurora クラスター内の DB インスタンスの数に依存しません。

前提条件

パラレルクエリのすべての機能を使用するには、バージョン 2.09 以上を実行している Aurora MySQL DB クラスターが必要です。パラレルクエリを使用したいクラスターが既にある場合は、互換性のあるバージョンにアップグレードしてからパラレルクエリを有効にすることができます。その場合、これらの新しいバージョンでは設定名とデフォルト値が異なるため、「パラレルクエリのアップグレードに関する考慮事項」のアップグレード手順に従ってください。

クラスターの DB インスタンスでは、`db.r*` インスタンスクラスを使用する必要があります。

クラスターでは、ハッシュ結合の最適化を必ず有効にしてください。この方法については、「パラレルクエリクラスターのハッシュ結合の有効化」を参照してください。

`aurora_parallel_query` や `aurora_disable_hash_join` などのパラメータをカスタマイズするには、クラスターで使用するカスタムパラメータグループが必要です。これらのパラメータは、DB パラメータグループを使用して DB インスタンスごとに個別に指定することもできます。ただし、DB クラスターパラメータグループで指定することをお勧めします。これにより、クラスターのすべての DB インスタンスでこれらのパラメータの同じ設定が継承されます。

制限事項

パラレルクエリ機能には、次の制限が適用されます。

- 並列クエリは Aurora I/O-Optimized DB クラスターのストレージ設定ではサポートされていません。
- db.t2 または db.t3 インスタンスクラスでは、パラレルクエリは使用できません。この制限は、`aurora_pq_force` セッション変数を使用してパラレルクエリを行う場合にも適用されます。
- パラレルクエリは、`COMPRESSED` または `REDUNDANT` の行形式を使用するテーブルには適用されません。パラレルクエリを使用するテーブルには、`COMPACT` または `DYNAMIC` の行形式を使用してください。
- Aurora では、コストに基づくアルゴリズムを使用して、それぞれの SQL ステートメントにパラレルクエリを使用するかどうかを判断します。ステートメントで特定の SQL コンストラクトを使用すると、パラレルクエリを防止したり、そのステートメントでパラレルクエリが行われる可能性

を低くしたりすることができます。SQL コンストラクトとパラレルクエリの互換性については、「SQL 構造でのパラレルクエリの動作」を参照してください。

- 各 Aurora DB インスタンスは、一度に特定の数のパラレルクエリセッションのみを実行できます。クエリにサブクエリ、結合、または UNION 演算子などのパラレルクエリを使用する複数の部分がある場合、それらのフェーズは順番に実行されます。このステートメントは、一度に 1 つのパラレルクエリセッションとしてカウントされます。パラレルクエリステータス可変を使用して、アクティブなセッションの数をモニタリングできます。ステータス可変 Aurora_pq_max_concurrent_requests を照会することで、特定の DB インスタンスの同時セッションの制限を確認できます。
- パラレルクエリは、Aurora がサポートされるすべての AWS リージョンでご利用になれます。ほとんどの AWS リージョンでは、パラレルクエリを使用するために必要な Aurora MySQL の最小バージョンは 2.09 です。
- パラレルクエリは、データ集約型クエリのパフォーマンスを向上させるように設計されています。軽量のクエリ用向けには設計されていません。
- SELECT ステートメント、特にデータ量の多いステートメントにはリーダーノードを使用することをお勧めします。

並列クエリの I/O コスト

Aurora MySQL クラスターが並列クエリを使用している場合、VolumeReadIOPS 値が増加することがあります。パラレルクエリでは、バッファプールは使用されません。したがって、クエリは高速ですが、この最適化された処理により、読み取りオペレーションが増加し、関連する料金が増加する可能性があります。

クエリのパラレルクエリ I/O コストは、ストレージレイヤーで計測され、パラレルクエリがオンになっている場合と同じかそれ以上になります。利点は、クエリのパフォーマンスが向上することです。パラレルクエリで I/O コストが高くなる可能性がある理由は 2 つあります。

- テーブル内のデータの一部がバッファプールにある場合でも、パラレルクエリでは、すべてのデータをストレージレイヤーでスキャンする必要があり、I/O コストが発生します。
- パラレルクエリを実行しても、バッファプールはウォームアップされません。その結果、同じパラレルクエリを連続して実行すると、完全な I/O コストが発生します。

パラレルクエリクラスターの計画

パラレルクエリが有効な DB クラスターを計画するには、いくつかの選択を行う必要があります。これには、セットアップステップ (完全な Aurora MySQL クラスターの作成または復元) の実行、および DB クラスター全体でパラレルクエリを有効にする範囲の決定が含まれます。

計画の一環として、以下の点を検討してください。

- MySQL 5.7 と互換性がある Aurora MySQL を使用する場合は、Aurora MySQL 2.09 以上を選択する必要があります。その場合、必ずプロビジョニングされたクラスターを作成します。その上で、aurora_parallel_query パラメータを使用してパラレルクエリを有効にします。

  バージョン 2.09 以上を実行している既存の Aurora MySQL クラスターがある場合は、パラレルクエリを使用するために新しいクラスターを作成する必要はありません。クラスターまたはクラスター内の特定の DB インスタンスを、 aurora_parallel_query パラメータが有効になっているパラメータグループに関連付けることができます。これにより、パラレルクエリで使用する関連データを設定する時間と手間を減らすことができます。
- アクセス時にパラレルクエリを使用できるように見直す必要がある大きなテーブルについての計画を立てます。いくつかの大きなテーブルでは、パラレルクエリが役立つように新しいものを作成する必要がある場合があります。例えば、全文検索インデックスを削除するなどが必要になる場合があります。詳細については、「パラレルクエリを利用するためのスキーマオブジェクトの作成」を参照してください。

パラレルクエリと Aurora MySQL のバージョンの互換性の確認

パラレルクエリを使用するクラスターと互換性のある Aurora MySQL のバージョンを確認するには、AWS CLI コマンドの describe-db-engine-versions を使用して、SupportsParallelQuery フィールドの値を確認します。次のコード例は、指定された AWS リージョンのパラレルクエリクラスターで使用可能な組み合わせを確認する方法を示しています。--query パラメータのすべての文字列は、必ず 1 行で指定してください。

```
aws rds describe-db-engine-versions --region us-east-1 --engine aurora-mysql \
--query '*[]|[?SupportsParallelQuery == `true`].[EngineVersion]' --output text
```

上記のコマンドを実行すると、次のような出力が返されます。この出力は、指定した AWS リージョンで使用可能な Aurora MySQL のバージョンによって異なります。

```
5.7.mysql_aurora.2.11.1
```

```
8.0.mysql_aurora.3.01.0
8.0.mysql_aurora.3.01.1
8.0.mysql_aurora.3.02.0
8.0.mysql_aurora.3.02.1
8.0.mysql_aurora.3.02.2
8.0.mysql_aurora.3.03.0
```

クラスターでパラレルクエリの使用をスタートしたら、パフォーマンスをモニタリングして、パラレルクエリを使用する上での障害を取り除くことができます。これらの手順については、「パラレルクエリのパフォーマンスチューニング」を参照してください。

パラレルクエリを使用する DB クラスターの作成

パラレルクエリを使用した Aurora MySQL クラスターの作成や、そのクラスターへの新しいインスタンスの追加、あるいは他の管理操作の実行には、他の Aurora MySQL クラスターと同様な、AWS Management Console や AWS CLI のテクニックを使用します。パラレルクエリを処理するための新しいクラスターを作成できます。また、パラレルクエリを使用する DB クラスターは、MySQL と互換性がある Aurora DB クラスターのスナップショットから復元することによって作成することもできます。新しい Aurora MySQL クラスターを作成するプロセスに詳しくない場合は、「Amazon Aurora DB クラスターの作成」の背景情報と前提条件を参照してください。

Aurora MySQL のエンジンのバージョンを選択する場合は、利用可能な最新のバージョンを選択することをお勧めします。現在、Aurora MySQL バージョン 2.09 以降はパラレルクエリをサポートしています。Aurora MySQL 2.09 以上を使用している場合、パラレルクエリの有効と無効を切り替えたり、既存のクラスターでパラレルクエリを使用したりできるため、柔軟性が増します。

新しいクラスターを作成する場合でも、スナップショットから復元する場合でも、同じテクニックを使用して、他の Aurora MySQL クラスターで行う新しい DB インスタンスを追加できます。

コンソールを使用したパラレルクエリクラスターの作成

次のように、コンソールで新しいパラレルクエリクラスターを作成できます。

AWS Management Console コンソールでパラレルクエリクラスターを作成するには

1. 「AWS Management Console」一般的な Amazon Aurora DB クラスターの作成 の手順に従います。
2. [Select engine (エンジンの選択)] 画面で、Aurora MySQL を選択します。

[エンジンバージョン] で、Aurora MySQL 2.09 以降を選択します。これらのバージョンでは、パラレルクエリの使用に関する制限が最も少なくなります。また、これらのバージョンは、いつでもパラレルクエリを有効または無効にできる最も高い柔軟性を備えています。

クラスターで Aurora MySQL の最新のバージョンを使用するのが現実的でない場合は、[Show versions that support the parallel query feature (パラレルクエリ機能がサポートされているバージョンを表示)] をオンにします。これにより、[Version (バージョン)] メニューがフィルタリングされ、パラレルクエリと互換性がある特定の Aurora MySQL のバージョンのみが表示されます。

3. [追加設定] で、[DB クラスターパラメータグループ] のために作成したパラメータグループを選択します。Aurora MySQL 2.09 以上では、このようなカスタムパラメータグループを使用する必要があります。DB クラスターパラメータグループで、パラメータ設定の `aurora_parallel_query=ON` と `aurora_disable_hash_join=OFF` を指定します。これにより、クラスターでパラレルクエリが有効になり、パラレルクエリと組み合わせて使用するハッシュ結合の最適化が有効になります。

新しいクラスターがパラレルクエリを使用できることを確認するには

1. 上記の方法を使用してクラスターを作成します。
2. (Aurora MySQL バージョン 2 または 3 の場合) `aurora_parallel_query` の設定が true であることを確認します。

```
mysql> select @@aurora_parallel_query;
+-------------------------+
| @@aurora_parallel_query |
+-------------------------+
|                       1 |
+-------------------------+
```

3. (Aurora MySQL バージョン 2 の場合) `aurora_disable_hash_join` 設定が false になっていることを確認します。

```
mysql> select @@aurora_disable_hash_join;
+----------------------------+
| @@aurora_disable_hash_join |
+----------------------------+
|                          0 |
+----------------------------+
```

4. いくつかの大きなテーブルとデータ集約型のクエリについて、クエリの計画を確認して、一部のクエリでパラレルクエリの最適化を使用していることを確認します。これを行うには、「パラレルクエリを使用しているステートメントの確認」の手順に従います。

CLI を使用したパラレルクエリクラスターの作成

次のように、CLI で新しいパラレルクエリクラスターを作成できます。

AWS CLI コンソールでパラレルクエリクラスターを作成するには

1. (オプション) パラレルクエリを使用するクラスターと互換性のある Aurora MySQL のバージョンを確認します。これを行うには、describe-db-engine-versions コマンドを使用して、SupportsParallelQuery フィールドの値を確認します。例については、「パラレルクエリと Aurora MySQL のバージョンの互換性の確認」を参照してください。
2. (オプション) aurora_parallel_query=ON 設定と aurora_disable_hash_join=OFF 設定を使用して、カスタム DB クラスターパラメータグループを作成します。以下のようなコマンドを使用します。

```
aws rds create-db-cluster-parameter-group --db-parameter-group-family aurora-
mysql5.7 --db-cluster-parameter-group-name pq-enabled-57-compatible
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name pq-
enabled-57-compatible \
  --parameters
 ParameterName=aurora_parallel_query,ParameterValue=ON,ApplyMethod=pending-reboot
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name pq-
enabled-57-compatible \
  --parameters
 ParameterName=aurora_disable_hash_join,ParameterValue=OFF,ApplyMethod=pending-
reboot
```

   このステップを行う場合は、後続の --db-cluster-parameter-group-name *my_cluster_parameter_group* ステートメントで create-db-cluster オプションを指定します。パラメータグループの名前は、使用するものに置き換えてください。このステップを省略する場合は、「パラレルクエリのオン/オフを切り替える」の説明に従って、後でパラメータグループを作成してクラスターに関連付けます。
3. 「AWS CLI」一般的な Amazon Aurora DB クラスターの作成 の手順に従います。
4. 以下のオプションのセットを指定します。

- --engine オプションでは、aurora-mysql を使用します。これらの値は、MySQL 5.7 または 8.0 と互換性があるパラレルクエリクラスターを生成します。
- --db-cluster-parameter-group-name オプションには、作成してパラメータの値に aurora_parallel_query=ON を指定した DB クラスターパラメータグループの名前を指定します。このオプションを省略すると、デフォルトのパラメータグループを使用してクラスターを作成してから、後でこのようなカスタムパラメータグループを使用するように変更できます。
- --engine-version オプションには、パラレルクエリと互換性がある Aurora MySQL のバージョンを使用します。必要に応じて、「パラレルクエリクラスターの計画」の手順に従ってバージョンの一覧を取得します。少なくとも、バージョン 2.09.0 を使用します。これらのバージョンでは、パラレルクエリが大幅に強化されています。

  次のサンプルはその方法を示しています。*$CLUSTER_ID* などの環境可変は、それぞれ使用する値に置き換えてください。この例では、--manage-master-user-password オプションも指定して、マスターユーザーパスワードを生成し、Secrets Manager で管理します。詳細については、「Amazon Aurora および AWS Secrets Manager によるパスワード管理」を参照してください。または、--master-password オプションを使用して、自分でパスワードを指定して管理することもできます。

```
aws rds create-db-cluster --db-cluster-identifier $CLUSTER_ID \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --master-username $MASTER_USER_ID --manage-master-user-password \
  --db-cluster-parameter-group-name $CUSTOM_CLUSTER_PARAM_GROUP

aws rds create-db-instance --db-instance-identifier ${INSTANCE_ID}-1 \
  --engine same_value_as_in_create_cluster_command \
  --db-cluster-identifier $CLUSTER_ID --db-instance-class $INSTANCE_CLASS
```

5. 作成または復元したクラスターにパラレルクエリ機能が使用可能であることを確認します。

   aurora_parallel_query 設定が存在することを確認します。この設定の値が 1 の場合は、パラレルクエリを使用する準備ができています。この設定の値が 0 の場合は、パラレルクエリを使用するために 1 に設定します。どちらの場合も、クラスターでパラレルクエリを実行できます。

```
mysql> select @@aurora_parallel_query;
+------------------------+
| @@aurora_parallel_query|
```

```
+-------------------------+
|                       1 |
+-------------------------+
```

AWS CLI を使用してスナップショットをパラレルクエリクラスターに復元するには。

1. パラレルクエリを使用するクラスターと互換性のある Aurora MySQL のバージョンを確認します。これを行うには、describe-db-engine-versions コマンドを使用して、SupportsParallelQuery フィールドの値を確認します。例については、「パラレルクエリと Aurora MySQL のバージョンの互換性の確認」を参照してください。復元したクラスターで使用するバージョンを決定します。MySQL 5.7 互換クラスターの場合、Aurora MySQL 2.09.0 を選択します。
2. Aurora MySQL 互換クラスターのスナップショットの位置を特定します。
3. 「AWS CLI」一般的な DB クラスターのスナップショットからの復元 の手順に従います。

```
aws rds restore-db-cluster-from-snapshot \
  --db-cluster-identifier mynewdbcluster \
  --snapshot-identifier mydbclustersnapshot \
  --engine aurora-mysql
```

4. 作成または復元したクラスターにパラレルクエリ機能が使用可能であることを確認します。CLI を使用したパラレルクエリクラスターの作成 と同じ確認手順を使用してください。

パラレルクエリのオン/オフを切り替える

並列クエリが有効になっている場合、Aurora MySQL はランタイムにクエリごとにそれを使用するかを決定します。結合、ユニオン、サブクエリなどの場合、Aurora MySQL は各クエリブロックに対して実行時にパラレルクエリを使用するかどうかを決定します。詳細については、「パラレルクエリを使用しているステートメントの確認」および「SQL 構造でのパラレルクエリの動作」を参照してください。

aurora_parallel_query オプションを使用して、DB インスタンスのグローバルレベルとセッションレベルの両方でパラレルクエリのオン/オフを動的に切り替えることができます。DB クラスターグループの aurora_parallel_query の設定を変更すると、デフォルトのパラレルクエリの有効と無効を切り替えることができます。

```
mysql> select @@aurora_parallel_query;
```

```
+-------------------------+
| @@aurora_parallel_query|
+-------------------------+
|                       1 |
+-------------------------+
```

aurora_parallel_query パラメータをセッションレベルで切り替えるには、スタンダードの方法を使用してクライアントの設定を変更します。例えば、mysql コマンドラインや JDBC または ODBC アプリケーションで変更できます。このスタンダードの MySQL クライアントのコマンドは set session aurora_parallel_query = {'ON'/'OFF'} です。セッションレベルのパラメータを JDBC 構成またはアプリケーションコード内に追加して、パラレルクエリを動的にオンまたはオフにすることもできます。

aurora_parallel_query パラメータの設定は、特定の DB インスタンスまたはクラスター全体に対して永続的に変更することができます。DB パラメータグループでこのパラメータの値を指定すると、その値はクラスターの特定の DB インスタンスにのみ適用されます。DB クラスターパラメータグループでこのパラメータの値を指定すると、クラスターのすべての DB インスタンスで同じ設定が継承されます。aurora_parallel_query パラメータをクラスターレベルで切り替えるには、「「パラメータグループを使用する」」で説明されているパラメータグループの使用方法に従います。以下のステップに従ってください。

1. カスタムクラスターパラメータグループ (推奨) またはカスタム DB パラメータグループを作成します。
2. このパラメータグループで、parallel_query を必要な値に更新します。
3. DB クラスターパラメータグループと DB パラメータグループのどちらを作成したかによって、パラレルクエリ機能を使用する Aurora クラスターまたは特定の DB インスタンスにパラメータグループをアタッチします。

> **Tip**
>
> aurora_parallel_query は動的パラメータであるため、この設定を変更した後にクラスターを再起動する必要はありません。ただし、このオプションを切り替える前に並行クエリを使用していた接続は、接続が閉じられるか、インスタンスが再起動されるまで、引き続き実行されます。

パラレルクエリパラメータは、API オペレーションの ModifyDBClusterParameterGroup や ModifyDBParameterGroup または AWS Management Consoleを使用して変更できます。

パラレルクエリクラスターのハッシュ結合の有効化

パラレルクエリは通常、ハッシュ結合の最適化による利点がある、大量のリソースを使用する種類のクエリに使用されます。そのため、パラレルクエリを使用する予定のクラスターでハッシュ結合を有効にしておくと便利です。ハッシュ結合を効果的に使用する方法については、ハッシュ結合を使用した大規模な Aurora MySQL 結合クエリの最適化 を参照してください。

コンソールを使用したパラレルクエリのオン/オフを切り替える

パラメータグループを使用して、DB インスタンスレベルまたは DB クラスターレベルでパラレルクエリをオン/オフに切り替えることができます。

AWS Management Console で DB クラスターのパラレルクエリをオンまたはオフにする方法

1. 「「パラメータグループを使用する」」の説明に従って、カスタムパラメータグループを作成します。
2. aurora_parallel_query を 1 (オン) または 0 (オフ) に更新します。パラレルクエリ特徴が利用可能なクラスターでは、aurora_parallel_query はデフォルトで無効になっています。
3. カスタムクラスターパラメータグループを使用する場合は、パラレルクエリ機能を使用する Aurora DB クラスターにアタッチします。カスタム DB パラメータグループを使用する場合は、クラスターの 1 つ以上の DB インスタンスにアタッチします。クラスターパラメータグループを使用することをお勧めします。そうすることにより、クラスターのすべての DB インスタンスでパラレルクエリとそれに関連するハッシュ結合などの機能の設定が同じになります。

CLI を使用したパラレルクエリのオン/オフ切り替え

パラレルクエリのパラメータは、`modify-db-cluster-parameter-group` または `modify-db-parameter-group` コマンドを使用して変更することができます。DB クラスターパラメータグループまたは DB パラメータグループのどちらを使用して `aurora_parallel_query` の値を指定するかに応じて、適切なコマンドを選択してください。

CLI で DB クラスターのパラレルクエリを有効または無効にする方法

- パラレルクエリパラメータは、`modify-db-cluster-parameter-group` コマンドを使用して変更します。以下のようなコマンドを使用します。カスタムパラメータグループの名前は、使用する適切なものに置き換えてください。`ON` オプションの `OFF` の部分の `ParameterValue` または `--parameters` も置き換えてください。

```
$ aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-
name cluster_param_group_name \
  --parameters
 ParameterName=aurora_parallel_query,ParameterValue=ON,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "cluster_param_group_name"
}

aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-
name cluster_param_group_name \
  --parameters ParameterName=aurora_pq,ParameterValue=ON,ApplyMethod=pending-reboot
```

またセッションレベルでパラレルクエリを有効または無効にすることも可能で、例えば mysql コマンドライン経由や JDBC や ODBC アプリケーション内などで実行できます。これを行うには、スタンダードメソッドを使用してクライアントの構成設定を変更します。例えば、Aurora MySQL では、スタンダード MySQL クライアントに対するコマンドは `set session aurora_parallel_query = {'ON'/'OFF'}` です。

セッションレベルのパラメータを JDBC 構成またはアプリケーションコード内に追加して、パラレルクエリを動的にオンまたはオフにすることもできます。

パラレルクエリオプティマイザの上書き

aurora_pq_force セッション変数を使用して、パラレルクエリオプティマイザを上書きし、クエリごとにパラレルクエリを行うことができます。これはテスト目的でのみ行うことをお勧めします。次の例では、セッションで aurora_pq_force を使用する方法を示します。

```
set SESSION aurora_parallel_query = ON;
set SESSION aurora_pq_force = ON;
```

上書きを無効にするには、以下の手順を実行します。

```
set SESSION aurora_pq_force = OFF;
```

パラレルクエリのアップグレードに関する考慮事項

パラレルクエリクラスターをアップグレードする際の元のバージョンと移行先のバージョンによっては、パラレルクエリで最適化できるクエリのタイプが強化される場合があります。またパラレルクエリに特別なエンジンモードパラメータを指定する必要がないこともあります。次のセクションでは、

パラレルクエリが有効になっているクラスターをアップグレードする際の考慮事項について説明します。

Aurora MySQL バージョン 3 への パラレルクエリクラスターのアップグレード

SQL ステートメント、句、およびデータタイプのいくつかには、Aurora MySQL バージョン 3 からスタートした新しいパラレルクエリサポートまたは改善されたサポートが含まれています。バージョン 3 より前のリリースからアップグレードする場合は、追加のクエリがパラレルクエリ最適化の恩恵を受けることができるかどうかをチェックしてください。これらのパラレルクエリの強化については、列のデータ型、パーティションテーブル、および 集計関数、GROUP BY 句、HAVING 句 を参照してください。

Aurora MySQL 2.08 以前からパラレルクエリクラスターをアップグレードする場合は、パラレルクエリを有効にする方法の変更についても確認してください。これを行うには、Aurora MySQL 2.09 以降へのアップグレード を読んでください。

Aurora MySQL バージョン 3 では、ハッシュ結合の最適化はデフォルトで有効になっています。以前のバージョンからの `aurora_disable_hash_join` 設定オプションは使用されません。

Aurora MySQL 2.09 以降へのアップグレード

Aurora MySQL バージョン 2.09 以上では、パラレルクエリはプロビジョニングされたクラスターで使用できるため、`parallelquery` エンジンモードパラメータは必要ありません。そのため、これらのバージョンでパラレルクエリを使用するために新しいクラスターを作成したり、既存のスナップショットから復元したりする必要はありません。これらのバージョンでは、「Aurora MySQL DB クラスターのマイナーバージョンまたはパッチレベルのアップグレード」で説明されているアップグレード手順に従ってクラスターをアップグレードできます。古いクラスターでは、パラレルクエリを使用するクラスターかプロビジョニングされたクラスターかにかかわらずアップグレードできます。[Engine version (エンジンバージョン)] メニューの選択肢の数を減らすには、[Show versions that support the parallel query feature (パラレルクエリ機能がサポートされているバージョンを表示)] をオンにしてメニューのエントリをフィルタリングします。その上で、Aurora MySQL 2.09 以上を選択します。

以前のパラレルクエリクラスターを Aurora MySQL 2.09 以上にアップグレードした後、アップグレードしたクラスターでパラレルクエリを有効にします。これらのバージョンではパラレルクエリはデフォルトで無効になっており、有効にする手順も異なります。またハッシュ結合の最適化もデフォルトで無効になっているため、個別に有効にする必要があります。そのため、アップグレード後にはこれらの設定をもう一度有効にしてください。その手順については、「パラレルクエリのオン/オフを切り替える」および「パラレルクエリクラスターのハッシュ結合の有効化」を参照してください。

特に、aurora_pq_supported と aurora_pq ではなく、aurora_parallel_query=ON と aurora_disable_hash_join=OFF の設定パラメータを使用してパラレルクエリを有効にするようにしてください。aurora_pq_supported パラメータと aurora_pq パラメータは、Aurora MySQL の新しいバージョンでは非推奨になっています。

アップグレードしたクラスターでは、EngineMode 属性の値は provisioned ではなく、parallelquery になります。指定したエンジンバージョンでパラレルクエリが使用できるかどうかを確認するには、SupportsParallelQuery コマンドの describe-db-engine-versions の出力の AWS CLI フィールドの値を確認します。Aurora MySQL の以前のバージョンでは、parallelquery の一覧に SupportedEngineModes があることを確認していました。

Aurora MySQL バージョン 2.09 以上にアップグレードすると、以下の機能を利用できるようになります。これらの機能は、Aurora MySQL の古いバージョンが実行されているパラレルクエリを使用するクラスターでは使用できません。

- パフォーマンスインサイト。詳細については、「Amazon Aurora での Performance Insights を使用したDB 負荷のモニタリング」を参照してください。
- バックトラック。詳細については、「Aurora DB クラスターのバックトラック」を参照してください。
- クラスターの停止とスタート。詳細については、「Amazon Aurora DB クラスターの停止と開始」を参照してください。

パラレルクエリのパフォーマンスチューニング

パラレルクエリを使用してワークロードのパフォーマンスを管理するには、この最適化が最も役立つクエリに対してパラレルクエリが使用されていることを確認します。

そのためには、以下を実行できます。

- 大きなテーブルがパラレルクエリと互換性があるようにします。テーブルのプロパティを変更したり、一部のテーブルを作成し直したりして、それらのテーブルに対するクエリでパラレルクエリの最適化を利用できるようにします。この方法については、「パラレルクエリを利用するためのスキーマオブジェクトの作成」を参照してください。
- パラレルクエリを使用するクエリをモニタリングします。この方法については、「パラレルクエリのモニタリング」を参照してください。

- パラレルクエリが最もデータ集約型で実行に時間がかかるクエリに使用され、ワークロードに適したレベルの同時実行性があることを確認します。この方法については、「パラレルクエリを使用しているステートメントの確認」を参照してください。
- SQL コードを調整し、パラレルクエリを有効にして目的のクエリに適用します。この方法については、「SQL 構造でのパラレルクエリの動作」を参照してください。

パラレルクエリを利用するためのスキーマオブジェクトの作成

パラレルクエリで使用するテーブルを作成または変更するには、「前提条件」および「制限事項」で説明されている要件を理解しておく必要があります。

パラレルクエリでは、テーブルに ROW_FORMAT=Compact または ROW_FORMAT=Dynamic の設定が使用されている必要があるため、Aurora の設定で INNODB_FILE_FORMAT 設定オプションへの変更を確認してください。SHOW TABLE STATUS ステートメントを発行して、データベース内のすべてのテーブルの行形式を確認します。

より多くのテーブルを処理するためにパラレルクエリをオンにするようにスキーマを変更する前に、必ずテストを行ってください。テストでは、パラレルクエリによってそれらのテーブルのパフォーマンスが実際に向上するかどうかを確認する必要があります。また、パラレルクエリのスキーマ要件が目標に適合していることを確認してください。

例えば、ROW_FORMAT=Compressed から ROW_FORMAT=Compact または ROW_FORMAT=Dynamic に切り替える前には、元のテーブルと新しいテーブルでワークロードのパフォーマンスをテストします。また、データ量の増加などの潜在的な影響も考慮してください。

パラレルクエリを使用しているステートメントの確認

一般的な操作では、パラレルクエリを利用するために特別なアクションを実行する必要はありません。クエリがパラレルクエリの必須要件を満たした後、クエリオプティマイザは、特定のクエリごとにパラレルクエリを使用するかどうかを自動的に決定します。

開発環境やテスト環境で実験を行うと、テーブルの行数や全体のデータ量が少ないためにパラレルクエリが使用されないことがあります。特に実験を実行するために最近作成したテーブルの場合、テーブルのデータも完全にバッファプールにある可能性があります。

クラスターのパフォーマンスをモニタリングまたは調整する場合は、パラレルクエリが適切な状況で使用されているかどうかを確認する必要があります。この機能を利用するには、データベースのスキーマ、設定、SQL クエリ、またはクラスタートポロジとアプリケーションの接続設定を調整する必要があります。

クエリでパラレルクエリが使用されているかどうかを確認するには、EXPLAIN ステートメントを実行してクエリプラン (explain プラン) を確認します。パラレルクエリの EXPLAIN 出力に SQL ステートメント、句および表現がどのように影響するかの例は、「SQL 構造でのパラレルクエリの動作」を参照してください。

次の例は、従来のクエリプランとパラレルクエリプランの違いを示しています。この explain プランは、TPC-H ベンチマークのクエリ 3 のものです。このセクションのサンプルクエリの多くは、TPC-H データセットのテーブルを使用しています。テーブル定義、クエリ、サンプルデータを生成する dbgen プログラムは、TPC-H のウェブサイトから入手できます。

```
EXPLAIN SELECT l_orderkey,
  sum(l_extendedprice * (1 - l_discount)) AS revenue,
  o_orderdate,
  o_shippriority
FROM customer,
  orders,
  lineitem
WHERE c_mktsegment = 'AUTOMOBILE'
AND c_custkey = o_custkey
AND l_orderkey = o_orderkey
AND o_orderdate < date '1995-03-13'
AND l_shipdate > date '1995-03-13'
GROUP BY l_orderkey,
  o_orderdate,
  o_shippriority
ORDER BY revenue DESC,
  o_orderdate LIMIT 10;
```

デフォルトでは、クエリには以下のようなプランがあります。クエリプランでハッシュ結合が使用されていない場合は、初期に最適化がオンになっていることを確認してください。

```
+----+-------------+----------+------------+------+---------------+------+---------
+------+----------+----------+----------------------------------------------------+
| id | select_type | table    | partitions | type | possible_keys | key  | key_len |
 ref  | rows     | filtered | Extra                                              |
+----+-------------+----------+------------+------+---------------+------+---------
+------+----------+----------+----------------------------------------------------+
|  1 | SIMPLE      | customer | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL |  1480234 |    10.00 | Using where; Using temporary; Using filesort       |
|  1 | SIMPLE      | orders   | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL | 14875240 |     3.33 | Using where; Using join buffer (Block Nested Loop) |
```

```
|  1 | SIMPLE      | lineitem | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL | 59270573 |     3.33 | Using where; Using join buffer (Block Nested Loop) |
+----+-------------+----------+------------+------+---------------+------+---------
+------+----------+----------+----------------------------------------------------+
```

Aurora MySQL バージョン 3 の場合、次のステートメントを発行すると、セッションレベルでハッシュ結合を有効にできます。

```
SET optimizer_switch='block_nested_loop=on';
```

Aurora MySQL バージョン 2.09 以降では、aurora_disable_hash_join DB パラメーターまたは DB クラスターパラメータを 0 (オフ) に設定します。aurora_disable_hash_join をオフにすると、optimizer_switch の値が hash_join=on に設定されます。

ハッシュ結合を有効にした後、EXPLAIN ステートメントをもう一度実行してみてください。ハッシュ結合を効果的に使用する方法については、ハッシュ結合を使用した大規模な Aurora MySQL 結合クエリの最適化 を参照してください。

ハッシュ結合がオンになっているがパラレルクエリがオフになっている場合、クエリには次のようなプランが含まれる可能性があります。これは、ハッシュ結合を使用しますが、パラレルクエリは使用しません。

```
+----+-------------+----------+...+-----------
+-------------------------------------------------------------------+
| id | select_type | table    |...| rows      | Extra
                          |
+----+-------------+----------+...+-----------
+-------------------------------------------------------------------+
|  1 | SIMPLE      | customer |...|   5798330 | Using where; Using index; Using
 temporary; Using filesort       |
|  1 | SIMPLE      | orders   |...| 154545408 | Using where; Using join buffer (Hash
 Join Outer table orders)   |
|  1 | SIMPLE      | lineitem |...| 606119300 | Using where; Using join buffer (Hash
 Join Outer table lineitem) |
+----+-------------+----------+...+-----------
+-------------------------------------------------------------------+
```

パラレルクエリが有効になると、EXPLAIN 出力の Extra カラムに表示されるように、このクエリプランの 2 つのステップでパラレルクエリの最適化が使用できます。これらのステップに対する I/O 集約型および CPU 集約型の処理は、ストレージレイヤーにプッシュダウンされます。

```
+----+...
+-------------------------------------------------------------------------------------------------------
+
| id |...| Extra
                                                    |
+----+...
+-------------------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using index; Using temporary; Using filesort
                                                    |
|  1 |...| Using where; Using join buffer (Hash Join Outer table orders); Using
 parallel query (4 columns, 1 filters, 1 exprs; 0 extra)   |
|  1 |...| Using where; Using join buffer (Hash Join Outer table lineitem); Using
 parallel query (4 columns, 1 filters, 1 exprs; 0 extra) |
+----+...
+-------------------------------------------------------------------------------------------------------
+
```

パラレルクエリの出力およびパラレルクエリが適用できる SQL ステートメントの部分の EXPLAIN 出力を解釈する方法については、「SQL 構造でのパラレルクエリの動作」を参照してください。

次の出力例は、コールドバッファプールを持つ db.r4.2xlarge インスタンスで前のクエリを実行した結果を示しています。パラレルクエリを使用すると、クエリが大幅に高速化されます。

> **Note**
>
> タイミングは多くの環境要因に依存するため、結果が異なる場合があります。自分の環境、ワークロードなどで結果を確認するには、常に独自のパフォーマンステストを実施してください。

```
-- Without parallel query
+------------+-------------+-------------+----------------+
| l_orderkey | revenue     | o_orderdate | o_shippriority |
+------------+-------------+-------------+----------------+
|   92511430 | 514726.4896 | 1995-03-06  |              0 |
.
.
|   28840519 | 454748.2485 | 1995-03-08  |              0 |
+------------+-------------+-------------+----------------+
10 rows in set (24 min 49.99 sec)
```

```
-- With parallel query
+------------+-------------+-------------+----------------+
| l_orderkey | revenue     | o_orderdate | o_shippriority |
+------------+-------------+-------------+----------------+
|   92511430 | 514726.4896 | 1995-03-06  |              0 |
.
.
|   28840519 | 454748.2485 | 1995-03-08  |              0 |
+------------+-------------+-------------+----------------+
10 rows in set (1 min 49.91 sec)
```

このセクション全体のサンプルクエリの多くは、この TPC-H データセットのテーブル、特に 2000 万行と以下の定義を持つ PART テーブルを使用しています。

```
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| p_partkey     | int(11)       | NO   | PRI | NULL    |       |
| p_name        | varchar(55)   | NO   |     | NULL    |       |
| p_mfgr        | char(25)      | NO   |     | NULL    |       |
| p_brand       | char(10)      | NO   |     | NULL    |       |
| p_type        | varchar(25)   | NO   |     | NULL    |       |
| p_size        | int(11)       | NO   |     | NULL    |       |
| p_container   | char(10)      | NO   |     | NULL    |       |
| p_retailprice | decimal(15,2) | NO   |     | NULL    |       |
| p_comment     | varchar(23)   | NO   |     | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
```

ワークロードを試して、個々の SQL ステートメントがパラレルクエリを利用できるかどうかを理解してください。次に、以下のモニタリング方法を使用して、実際のワークロードでパラレルクエリが使用される頻度を時間をかけて確認します。実際のワークロードでは、同時実行制限などの追加の要素が適用されます。

パラレルクエリのモニタリング

Aurora MySQL クラスターがパラレルクエリを使用している場合、VolumeReadIOPS 値が増加することがあります。パラレルクエリでは、バッファプールは使用されません。したがって、クエリは高速ですが、この最適化された処理により、読み取り操作とそれに関連する料金が増加する可能性があります。

Amazon RDS コンソールでのメトリクスの表示 で説明されている Amazon CloudWatch メトリクスに加えて、Aurora は他のグローバルなステータス可変を提供します。これらのグローバルステータス可変を使用して、パラレルクエリの実行のモニタリングに役立てることができます。これらの変数からは、オプティマイザが特定の状況でパラレルクエリを使用したり、使用しなかったりする理由についてのインサイトを得ることができます。これらの可変にアクセスするには、SHOW GLOBAL STATUS コマンドを使用します。これらの可変は次のとおりです。

パラレルクエリセッションは、データベースによって実行されるクエリと必ずしも 1 対 1 のマッピングにはなっていません。例えば、クエリプランにパラレルクエリを使用する 2 つのステップがあるとします。この場合、クエリには 2 つのパラレルセッションが含まれ、試行された要求と成功したリクエストのカウンターは 2 つずつ増分されます。

EXPLAIN ステートメントを発行してパラレルクエリを試してみると、実際にはクエリが実行されていなくても「選択されていません」と指定されたカウンターの増加が見込まれます。本稼働環境でパラレルクエリを処理する場合、「選択されていない」カウンターが予想どおりに高速に増加しているかどうかを確認できます。その段階で、目的のクエリでパラレルクエリが実行されるように調整できます。これを行うには、パラレルクエリが有効になっている クラスターの設定、クエリの組み合わせ、DB インスタンスなどを変更します。

これらのカウンターは、DB インスタンスレベルで追跡されます。別のエンドポイントに接続すると、各 DB インスタンスが独自のパラレルクエリセットを実行するため、別のメトリクスが表示されることがあります。リーダーエンドポイントがセッションごとに異なる DB インスタンスに接続すると、別のメトリクスが表示されることもあります。

名前	説明
Aurora_pq_bytes_returned	パラレルクエリ中にヘッドノードに送信されたタプルデータ構造のバイト数。Aurora_pq_pages_pushed_down と比較するために 16,384 で割ります。
Aurora_pq_max_concurrent_requests	この Aurora DB インスタンスで同時に実行できるパラレルクエリセッションの最大数。これは、AWS の DB インスタンスクラスによって異なる固定の数です。

名前	説明
Aurora_pq_pages_pushed_down	パラレルクエリがヘッドノードへのネットワーク送信を回避したデータページ数 (それぞれ 16 KiB の固定サイズ)。
Aurora_pq_request_attempted	リクエストされたパラレルクエリセッションの数。この値は、サブクエリや結合などの SQL 構成に応じて、クエリごとに複数のセッションを表す場合があります。
Aurora_pq_request_executed	パラレルクエリセッションの数は正常に実行されます。
Aurora_pq_request_failed	クライアントにエラーを戻したパラレルクエリセッションの数。場合によっては、例えば、ストレージレイヤーの問題のために、パラレルクエリのリクエストが失敗することがあります。このような場合、失敗したクエリ部分は、非パラレルクエリメカニズムを使用して再試行されます。再試行されたクエリも失敗すると、エラーがクライアントに返され、このカウンターが増分されます。
Aurora_pq_request_in_progress	現在進行中のパラレルクエリセッションの数。この数は、Aurora DB クラスター全体ではなく、接続している特定の Aurora DB インスタンスのものが適用されます。DB インスタンスが同時実行の制限に近いかどうかを調べるには、この値を Aurora_pq_max_concurrent_requests と比較します。
Aurora_pq_request_not_chosen	クエリを満たすためにパラレルクエリが選択されなかった回数。この値は、他のいくつかのより細かいカウンターの合計です。EXPLAIN ステートメントでは、クエリが実際に実行されていない場合でもこのカウンターは増加します。

名前	説明
Aurora_pq_request_not_chosen_below_min_rows	テーブル内の行数のためにパラレルクエリが選択されなかった回数。EXPLAIN ステートメントでは、クエリが実際に実行されていない場合でもこのカウンターは増加します。
Aurora_pq_request_not_chosen_column_bit	射影された列の中にサポートされていないデータ型があるために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。
Aurora_pq_request_not_chosen_column_geometry	テーブルに GEOMETRY データ型の列があるために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。この制限を解除する Aurora MySQL のバージョンについては、Aurora MySQL バージョン 3 への パラレルクエリクラスターのアップグレード を参照してください。
Aurora_pq_request_not_chosen_column_lob	LOB データタイプ、または宣言された長さのため外部に保存された VARCHAR カラムをテーブルが持っていることが原因で、非パラレルクエリの処理パスを使用したパラレルクエリのリクエスト数。この制限を解除する Aurora MySQL のバージョンについては、Aurora MySQL バージョン 3 への パラレルクエリクラスターのアップグレード を参照してください。
Aurora_pq_request_not_chosen_column_virtual	テーブルに仮想列があるために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。
Aurora_pq_request_not_chosen_custom_charset	テーブルにカスタム文字セットの列があるために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。

名前	説明
Aurora_pq_request_not_chosen_fast_ddl	テーブルが高速 DDL の ALTER ステートメントによって変更中であるために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。
Aurora_pq_request_not_chosen_few_pages_outside_buffer_pool	パラレルクエリを価値のあるものにするためのバッファされていないテーブルデータが十分ないため、テーブルデータの 95 パーセント未満がバッファプールにあったにもかかわらず、パラレルクエリの回数は選択されませんでした。
Aurora_pq_request_not_chosen_full_text_index	テーブルに全文インデックスがあるために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。
Aurora_pq_request_not_chosen_high_buffer_pool_pct	テーブルデータの高パーセンテージ (現在は 95 パーセント以上) が既にバッファプールに入っていたため、パラレルクエリが選択されなかった回数。このような場合、オプティマイザは、バッファプールからのデータの読取りがより効率的であると判断します。EXPLAIN ステートメントでは、クエリが実際に実行されていない場合でもこのカウンターは増加します。
Aurora_pq_request_not_chosen_index_hint	クエリにインデックスヒントが含まれているために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。
Aurora_pq_request_not_chosen_innodb_table_format	テーブルが、サポートされていない InnoDB の行形式を使用しているために、パラレルクエリ以外の処理方法が適用されるパラレルクエリのリクエスト数。Aurora のパラレルクエリは、COMPACT、REDUNDANT ,および DYNAMIC の行形式にのみ適用されます。

名前	説明
Aurora_pq_request_not_chosen_long_trx	長時間実行トランザクション内でクエリがスタートされているために、非パラレルクエリ処理パスを使用したパラレルクエリリクエストの数。EXPLAIN ステートメントでは、クエリが実際に実行されていない場合でもこのカウンターは増加します。
Aurora_pq_request_not_chosen_no_where_clause	クエリに WHERE 句がないために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。
Aurora_pq_request_not_chosen_range_scan	インデックスの範囲スキャンを使用しているために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。
Aurora_pq_request_not_chosen_row_length_too_long	すべての列の合計長が長すぎるために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。
Aurora_pq_request_not_chosen_small_table	行数および平均行長によって決定される、テーブルの全体的なサイズのためにパラレルクエリが選択されなかった回数。EXPLAIN ステートメントでは、クエリが実際に実行されていない場合でもこのカウンターは増加します。
Aurora_pq_request_not_chosen_temporary_table	クエリでサポートされていない MyISAM または memory テーブルタイプを使用しているテンポラリテーブルを参照しているために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。

名前	説明
Aurora_pq_request_not_chosen_tx_isolation	クエリでサポートされていないトランザクション分離レベルを使用しているために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。リーダー DB インスタンスでは、パラレルクエリは REPEATABLE READ および READ COMMITTED 分離レベルにのみ適用されます。
Aurora_pq_request_not_chosen_update_delete_stmts	クエリが UPDATE または DELETE ステートメントの一部であるために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。
Aurora_pq_request_not_chosen_unsupported_access	WHERE 句がパラレルクエリの基準を満たしていないために、非パラレルクエリ処理パスを使用するパラレルクエリリクエストの数。この結果は、クエリがデータ集約型スキャンを必要としない場合、またはクエリが DELETE または UPDATE ステートメントである場合に発生します。
Aurora_pq_request_not_chosen_unsupported_storage_type	Aurora MySQL DB クラスターがサポートされている Aurora クラスターストレージ設定を使用していないために非並列クエリ処理パスを使用する並列クエリリクエストの数。このパラメータは、Aurora MySQL バージョン 3.04 以降で使用できます。詳細については、「制限事項」を参照してください。
Aurora_pq_request_throttled	特定の Aurora DB インスタンスで既に実行されている同時パラレルクエリの最大数のために、パラレルクエリが選択されなかった回数。

SQL 構造でのパラレルクエリの動作

このセクションでは、特定の SQL ステートメントでパラレルクエリが使用される理由と使用されない理由について詳しく説明します。また、Aurora MySQL の機能とパラレルクエリのインタラクションについても説明します。これらの内容は、パラレルクエリを使用するクラスターのパフォーマンスの問題を診断したり、特定のワークロードにパラレルクエリがどのように適用されるかを理解したりするために役立ちます。

パラレルクエリを使用するかどうかの決定は、ステートメントが実行される時点で発生する多くの要因に依存します。したがって、パラレルクエリは、特定の条件の下で常に使用される、特定の条件の下では決して使用されない、または特定の条件の下でのみ使用される特定のクエリに対して使用される可能性があります。

> **Tip**
>
> 以下の例を HTML で表示している場合は、記載されている各コードの右上隅にあるコピーウィジェットを使用して SQL コードをコピーして使用することができます。このコピーウィジェットを使用すると、`mysql>` プロンプトとそれに続く `->` の行の周りの余分な文字がコピーされません。

トピック

- ビュー
- データ操作言語 (DML) ステートメント
- トランザクションとロック
- B ツリーインデックス
- 全文検索 (FTS) インデックス
- 仮想列
- 組み込みキャッシュメカニズム
- オプティマイザヒント
- MyISAM テンポラリテーブル

EXPLAIN ステートメント

このセクションの例で示すように、EXPLAIN ステートメントは、クエリの各ステージが現在パラレルクエリに適しているかどうかを示します。また、クエリのどの側面をストレージレイヤーにプッシュダウンできるかを示します。以下に、クエリプランで最も重要な点を示します。

- NULL 列の key 以外の値は、インデックスのルックアップを使用してクエリを効率的に実行できることを示しています。またパラレルクエリは効率的には実行できません。
- rows 列の値が小さい場合 (値が百万単位に達しない場合) は、クエリでアクセスしているデータがパラレルクエリが役立つほどの量ではないことを示しています。そのため、パラレルクエリが使用されることはあまりありません。
- Extra 列には、パラレルクエリを使用することが予想されるかどうかを示します。この出力は、次の例のようになります。

  ```
  Using parallel query (A columns, B filters, C exprs; D extra)
  ```

  columns の数は、クエリブロックで参照される列の数を表します。

  filters の数は、列の値と定数の簡単な比較を表す WHERE 述語の数を表します。比較には、等価、不等値、または範囲を使用することができます。Aurora は、これらの種類の述語を最も効果的にパラレル化できます。

  exprs の数は、関数呼び出し、演算子、またはパラレル化できる他の表現の数を表しますが、フィルター条件ほど効果的ではありません。

  extra の数は、プッシュダウンできず、ヘッドノードによって実行される表現の数を表します。

例えば、次の EXPLAIN 出力を考えてみます。

```
mysql> explain select p_name, p_mfgr from part
    -> where p_brand is not null
    -> and upper(p_type) is not null
    -> and round(p_retailprice) is not null;
+----+-------------+-------+...+----------
+-------------------------------------------------------------------------------+
| id | select_type | table |...| rows     | Extra
                                |
+----+-------------+-------+...+----------
+-------------------------------------------------------------------------------+
|  1 | SIMPLE      | part  |...| 20427936 | Using where; Using parallel query (5
 columns, 1 filters, 2 exprs; 0 extra) |
+----+-------------+-------+...+----------
+-------------------------------------------------------------------------------+
```

Extra 列の情報は、各行から 5 つの列が抽出され、クエリ条件を評価し結果セットを構成することを示しています。1 つの WHERE 述語にはフィルター、つまり WHERE 句で直接テストされた列が含まれます。2 つの WHERE 句では、この場合は関数呼び出しを含む、より複雑な表現を評価する必要があります。0 extra フィールドは、WHERE 句のすべての操作がパラレルクエリ処理の一部としてストレージレイヤーにプッシュダウンされることを確認します。

パラレルクエリが選択されていない場合は、通常、EXPLAIN 出力の他の列から理由を推測できます。例えば、rows 値が小さすぎたり、possible_keys 列が、データ集約型スキャンではなくインデックスルックアップを使用できることを示します。次の例は、オプティマイズがクエリでスキャンする行の数が少ないとみなすクエリを示しています。これは、主キーの文字数に基づいて判断されます。この場合、パラレルクエリは不要です。

```
mysql> explain select count(*) from part where p_partkey between 1 and 100;
+----+-------------+-------+-------+---------------+---------+---------+------+------
+--------------------------+
| id | select_type | table | type  | possible_keys | key     | key_len | ref  | rows |
 Extra                    |
+----+-------------+-------+-------+---------------+---------+---------+------+------
+--------------------------+
|  1 | SIMPLE      | part  | range | PRIMARY       | PRIMARY | 4       | NULL |   99 |
 Using where; Using index |
+----+-------------+-------+-------+---------------+---------+---------+------+------
+--------------------------+
```

パラレルクエリを使用するかどうかを示す出力には、EXPLAIN ステートメントが実行された時点で利用可能なすべての要素が考慮されます。その間に状況が変わった場合、オプティマイザはクエリが実際に実行されたときに別の選択肢を作る場合もあります。例えば、EXPLAIN は、ステートメントがパラレルクエリを使用すると報告する場合があります。しかし、クエリが実際に後で実行される場合は、条件に基づいてパラレルクエリを使用しないことがあります。このような条件には、同時に実行されている他の複数のパラレルクエリが含まれます。また、テーブルから削除される行、作成される新しいインデックス、実行中のトランザクションに時間がかかりすぎていることなども含まれます。

WHERE 句

クエリがパラレルクエリの最適化を使用するには、WHERE 句を含める必要があります。

パラレルクエリの最適化は、WHERE 句で使用される多くの種類の表現を高速化します。

- 列の値と定数の簡単な比較は、フィルターとして知られています。これらの比較は、ストレージレイヤーへのプッシュダウンを最大限に活用します。クエリのフィルター表現の数は、EXPLAIN 出力でレポートされます。
- WHERE 句にある他の種類の表現も、可能であれば、ストレージレイヤーにプッシュダウンされます。クエリ内のそのような表現の数は、EXPLAIN 出力でレポートされます。これらの表現は、関数呼び出し、LIKE 演算子、CASE 表現などです。
- 特定の関数と演算子は、現在、パラレルクエリによってプッシュダウンされていません。クエリ内のそのような表現の数は、extra 出力の EXPLAIN カウンターとしてレポートされます。残りのクエリは引き続きパラレルクエリを使用できます。
- 選択リスト内の表現はプッシュダウンされませんが、そのような関数を含むクエリは、パラレルクエリの中間結果のネットワークトラフィックが減少しても有益です。例えば、選択リスト内の集計関数を呼び出すクエリでは、集計関数がプッシュダウンされていなくても、パラレルクエリを使用できます。

例えば、次のクエリはテーブル全体のスキャンを実行し、P_BRAND 列のすべての値を処理します。ただし、クエリには WHERE 句が含まれていないため、パラレルクエリは使用されません。

```
mysql> explain select count(*), p_brand from part group by p_brand;
+----+-------------+-------+------+---------------+------+---------+------+----------
+----------------------------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows     |
 Extra                            |
```

```
+----+-------------+-------+------+---------------+------+---------+------+----------
+---------------------------------+
|  1 | SIMPLE      | part  | ALL  | NULL          | NULL | NULL    | NULL | 20427936 |
 Using temporary; Using filesort |
+----+-------------+-------+------+---------------+------+---------+------+----------
+---------------------------------+
```

対照的に、次のクエリには、結果をフィルタリングする WHERE 述語が含まれているため、パラレルクエリを適用できます。

```
mysql> explain select count(*), p_brand from part where p_name is not null
    ->   and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
    -> group by p_brand;
+----+...+----------
+-------------------------------------------------------------------------------------------------------
+
| id |...| rows     | Extra
                                          |
+----+...+----------
+-------------------------------------------------------------------------------------------------------
+
|  1 |...| 20427936 | Using where; Using temporary; Using filesort; Using parallel
 query (5 columns, 1 filters, 2 exprs; 0 extra) |
+----+...+----------
+-------------------------------------------------------------------------------------------------------
+
```

オプティマイザが、クエリブロックの戻される行数が少ないと見積もった場合、そのクエリブロックに対してパラレルクエリは使用されません。次の例は、プライマリキー列のより大きい演算子が数百万行にも適用され、パラレルクエリが使用される場合を示しています。その反対に数の少ないテストでは、ほんの数行にしか適用されず、パラレルクエリは使用されません。

```
mysql> explain select count(*) from part where p_partkey > 10;
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs;
 0 extra) |
```

```
+----+...+----------
+-------------------------------------------------------------------------------+

mysql> explain select count(*) from part where p_partkey < 10;
+----+...+------+--------------------------+
| id |...| rows | Extra                    |
+----+...+------+--------------------------+
|  1 |...|    9 | Using where; Using index |
+----+...+------+--------------------------+
```

データ定義言語 (DDL)

Aurora MySQL バージョン 2 では、パラレルクエリは、高速データ定義言語 (DDL) オペレーションが保留されていないテーブルでのみ利用可能です。Aurora MySQL バージョン 3 では、インスタント DDL オペレーションと同時にテーブルに対してパラレルクエリを使用できます。

Aurora MySQL バージョン 3 のインスタント DDL では、Aurora MySQL バージョン 2 の高速 DDL 機能が置き換えられます。DDL ステートメントの詳細については、インスタント DDL (Aurora MySQL バージョン 3) を参照してください。

列のデータ型

Aurora MySQL バージョン 3 では、パラレルクエリはデータタイプ TEXT、BLOB、JSON、および GEOMETRY のカラムを含むテーブルで使用できます。また、宣言された長さの最大数が 768 バイト以上の VARCHAR 、および CHAR のカラムでも使用することが可能です。クエリがそのようなラージオブジェクトタイプを含む列を参照している場合、それを取得するための追加作業によってクエリ処理にオーバーヘッドが発生します。その場合、それらの列への参照をクエリが省略できるかチェックしてください。そうでない場合は、ベンチマークを実行し、パラレルクエリをオンまたはオフにしてた状態でこのようなクエリが高速であるかどうかを確認します。

Aurora MySQL バージョン 2 では、ラージオブジェクトタイプに対してパラレルクエリは次の制限があります。

- TEXT、BLOB、JSON、GEOMETRY データ型は、並列クエリではサポートされていません。これらの型の列を参照するクエリは、並列クエリを使用できません。
- 可変長の列 (VARCHAR および CHAR) は、最大 768 バイトの宣言された最大長までの並列クエリと互換性があります。上記より長い最大長で宣言された型の列を参照するクエリは、並列クエリを使用できません。マルチバイト文字セットを使用する列の場合、バイト制限には文字セット内の最大バイト数が考慮されます。例えば、最大文字長が 4 バイトの文字セット utf8mb4 の場

合、VARCHAR(192) 列はパラレルクエリと互換性がありますが、VARCHAR(193) 列は互換性はありません。

パーティションテーブル

Aurora MySQL バージョン 3 では、パーティショニングされたテーブルをパラレルクエリで使用できます。パーティショニングテーブルは内部的に複数の小さなテーブルとして表されるため、非パーティションテーブルに対してパラレルクエリを使用するクエリでは、同一のパーティショニングテーブルに対してパラレルクエリを使用しない場合があります。Aurora MySQL は、テーブル全体のサイズを評価するのではなく、各パーティションがパラレルクエリ最適化の対象となるのに十分な大きさがあるかどうかを検討します。パーティショニングテーブルのクエリがパラレルクエリを使用しない場合、Aurora_pq_request_not_chosen_small_table ステータス 可変がインクリメントされているかどうかをチェックしてください。

例えば、PARTITION BY HASH (*column*) PARTITIONS 2 でパーティショニングされている一つのテーブルと、PARTITION BY HASH (*column*) PARTITIONS 10 でパーティション分散されている別のテーブルについて考えてみます。2 つのパーティションがあるテーブルでは、パーティションは 10 個のパーティションを持つテーブルの 5 倍になります。したがって、パラレルクエリは、パーティションがより少ないテーブルに対するクエリに使用される可能性が高くなります。次の例では、テーブル PART_BIG_PARTITIONS には 2 つのパーティションがあり、PART_SMALL_PARTITIONS には 10 個のパーティションがあります。同一データでは、大きなパーティションがより少ないテーブルに対してパラレルクエリは使用される可能性が高くなります。

```
mysql> explain select count(*), p_brand from part_big_partitions where p_name is not
 null
    ->   and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
 group by p_brand;
+----+-------------+---------------------+------------
+-------------------------------------------------------------------------------------------
+
| id | select_type | table               | partitions | Extra
                                                                                          |
+----+-------------+---------------------+------------
+-------------------------------------------------------------------------------------------
+
|  1 | SIMPLE      | part_big_partitions | p0,p1      | Using where; Using temporary;
 Using parallel query (4 columns, 1 filters, 1 exprs; 0 extra; 1 group-bys, 1 aggrs) |
```

```
+----+-------------+---------------------+------------
+-------------------------------------------------------------------------------------------------
+

mysql> explain select count(*), p_brand from part_small_partitions where p_name is not
 null
    ->   and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
 group by p_brand;
+----+-------------+----------------------+------------------------------
+------------------------------+
| id | select_type | table                | partitions                   | Extra
                   |
+----+-------------+----------------------+------------------------------
+------------------------------+
|  1 | SIMPLE      | part_small_partitions | p0,p1,p2,p3,p4,p5,p6,p7,p8,p9 | Using
 where; Using temporary |
+----+-------------+----------------------+------------------------------
+------------------------------+
```

集計関数、GROUP BY 句、HAVING 句

集計関数を含むクエリは、大規模なテーブル内の多数の行をスキャンするため、パラレルクエリに適しています。

Aurora MySQL 3 では、パラレルクエリは選択リストと HAVING 句内の集計関数呼び出しを最適化できます。

Aurora MySQL 3 より前では、選択リストまたは HAVING 句の集計関数呼び出しはストレージレイヤーにプッシュダウンされませんでした。ただし、パラレルクエリは、集計関数を使用してこのようなクエリのパフォーマンスを向上させることができます。これは、初期にストレージレイヤーでパラレルに raw データページから列値を抽出することによって行われます。次に、それらの値をデータページ全体ではなくコンパクトなタプル形式でヘッドノードに戻します。今回も、クエリにはパラレルクエリを有効にするための少なくとも 1 つの WHERE 述語が必要です。

次の簡単な例は、パラレルクエリの利点を受ける集約クエリの種類を示しています。これは、中間結果をコンパクト形式でヘッドノードに戻し、一致しない行を中間結果から除外するか、またはその両方を行うことによって行います。

```
mysql> explain select sql_no_cache count(distinct p_brand) from part where p_mfgr =
 'Manufacturer#5';
+----+...+-------------------------------------------------------------------------+
```

```
| id |...| Extra                                                                                |
+----+...+--------------------------------------------------------------------------------------+
|  1 |...| Using where; Using parallel query (2 columns, 1 filters, 0 exprs; 0 extra) |
+----+...+--------------------------------------------------------------------------------------+

mysql> explain select sql_no_cache p_mfgr from part where p_retailprice > 1000 group by
 p_mfgr having count(*) > 100;
+----+...
+-----------------------------------------------------------------------------------------------------
+
| id |...| Extra
                                  |
+----+...
+-----------------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using temporary; Using filesort; Using parallel query (3
 columns, 0 filters, 1 exprs; 0 extra) |
+----+...
+-----------------------------------------------------------------------------------------------------
+
```

WHERE 句での関数呼び出し

Aurora は、WHERE 句のほとんどの組み込み関数への呼び出しにパラレルクエリの最適化を適用できます。これらの関数呼び出しをパラレル化すると、いくつかの CPU 作業がヘッドノードからオフロードされます。最も早いクエリステージで述語関数を並行して評価することで、Aurora は後のステージで送信および処理されるデータ量を最小限に抑えることができます。

現在、パラレル化は選択リストの関数呼び出しには適用されません。これらの関数は、同じ関数呼び出しが WHERE 句に現れても、ヘッドノードによって評価されます。関連する列からの元の値は、ストレージノードからヘッドノードに送信されたタプルに含まれます。ヘッドノードは、UPPER、CONCATENATE などの変換を行って、結果セットの最終的な値を生成します。

次の例では、LOWER 句にあるため、パラレルクエリは WHERE の呼び出しをパラレル化します。SUBSTR と UPPER の呼び出しは選択したリストにあるため、パラレルクエリには影響されません。

```
mysql> explain select sql_no_cache distinct substr(upper(p_name),1,5) from part
    -> where lower(p_name) like '%cornflower%' or lower(p_name) like '%goldenrod%';
+----+...
+-----------------------------------------------------------------------------------------------------
+
```

```
| id |...| Extra
                |
+----+...
+-------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using temporary; Using parallel query (2 columns, 0 filters, 1
 exprs; 0 extra) |
+----+...
+-------------------------------------------------------------------------------------------
+
```

CASE 表現や LIKE 演算子など、他の表現にも同じ考慮事項が適用されます。次の例では、パラレルクエリは、CASE 句の LIKE 表現と WHERE演算子を評価します。

```
mysql> explain select p_mfgr, p_retailprice from part
    -> where p_retailprice > case p_mfgr
    ->   when 'Manufacturer#1' then 1000
    ->   when 'Manufacturer#2' then 1200
    ->   else 950
    -> end
    -> and p_name like '%vanilla%'
    -> group by p_retailprice;
+----+...
+-------------------------------------------------------------------------------------------
+
| id |...| Extra
                                  |
+----+...
+-------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using temporary; Using filesort; Using parallel query (4
 columns, 0 filters, 2 exprs; 0 extra) |
+----+...
+-------------------------------------------------------------------------------------------
+
```

LIMIT 句

現在のところ、LIMIT 句を含むクエリブロックでは、パラレルクエリは使用されません。GROUP by、ORDER BY、または結合を使用して、以前のクエリフェーズでもパラレルクエリを使用することができます。

比較演算子

オプティマイザは、比較演算子を評価するためにスキャンする行数を推定し、その推定値に基づいてパラレルクエリを使用するかどうかを決定します。

次の初期の例は、プライマリキー列との等価比較をパラレルクエリなしで効率的に実行できることを示しています。次の 2 番目の例では、インデックス作成されていない列に対する同様の比較では数百万行のスキャンが必要なため、パラレルクエリのメリットが得られます。

```
mysql> explain select * from part where p_partkey = 10;
+----+...+------+-------+
| id |...| rows | Extra |
+----+...+------+-------+
|  1 |...|    1 | NULL  |
+----+...+------+-------+

mysql> explain select * from part where p_type = 'LARGE BRUSHED BRASS';
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (9 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+
```

等しくないテストや、より小さい、より大きい、等しい、または BETWEEN などの範囲比較にも同じ考慮事項が適用されます。オプティマイザは、スキャンする行数を推定し、I/O 全体の量に基づいてパラレルクエリが有効かどうかを判断します。

Joins

大きなテーブルを使用した結合クエリには、通常、パラレルクエリの最適化のメリットを受けるデータ集約型操作が含まれます。現在、複数のテーブル (つまり、結合述語自体) 間の列値の比較パラレルパラレル化されません。ただし、パラレルクエリは、ハッシュ結合中に Bloom フィルターを構築するなど、他の結合フェーズの内部処理の一部をプッシュダウンできます。パラレルクエリは、WHERE 句がなくても結合クエリに適用できます。したがって、結合クエリは、パラレルクエリを使用するために WHERE 句が必要であるという規則に対する例外です。

結合処理の各フェーズが評価され、パラレルクエリに適格であるかどうかがチェックされます。複数のフェーズでパラレルクエリを使用できる場合は、これらのフェーズが順番に実行されます。したがって、各結合クエリは、同時実行制限に関して単一のパラレルクエリセッションとしてカウントされます。

例えば、結合クエリが結合テーブルの 1 つから行をフィルタリングする WHERE 述語を含む場合、そのフィルタリングオプションはパラレルクエリを使用できます。別の例として、結合クエリがハッシュ結合メカニズムを使用するとします。例えば、大きなテーブルを小さなテーブルに結合する場合などです。この場合、Bloom フィルターデータ構造を生成するためのテーブルスキャンは、パラレルクエリを使用することができます。

> **Note**
>
> パラレルクエリは通常、ハッシュ結合の最適化による利点がある、大量のリソースを使用する種類のクエリに使用されます。ハッシュ結合の最適化を有効にする方法は、Aurora MySQL のバージョンによって異なります。各バージョンの詳細については、パラレルクエリクラスターのハッシュ結合の有効化 を参照してください。ハッシュ結合を効果的に使用する方法については、ハッシュ結合を使用した大規模な Aurora MySQL 結合クエリの最適化 を参照してください。

```
mysql> explain select count(*) from orders join customer where o_custkey = c_custkey;
+----+...+----------+------+--------------+-------------+...+-----------
+-------------------------------------------------------------------------------------------------
+
| id |...| table    | type  | possible_keys | key         |...| rows      | Extra

              |
+----+...+----------+------+--------------+-------------+...+-----------
+-------------------------------------------------------------------------------------------------
+
|  1 |...| customer | index | PRIMARY       | c_nationkey |...|  15051972 | Using index

              |
|  1 |...| orders   | ALL   | o_custkey     | NULL        |...| 154545408 | Using join
 buffer (Hash Join Outer table orders); Using parallel query (1 columns, 0 filters, 1
 exprs; 0 extra) |
+----+...+----------+------+--------------+-------------+...+-----------
+-------------------------------------------------------------------------------------------------
+
```

ネストされたループメカニズムを使用する結合クエリの場合、最も外側のネストされたループブロックはパラレルクエリを使用する場合があります。パラレルクエリの使用は、WHERE 句に追加のフィルター条件が存在するなど、通常と同じ要素に依存します。

```
mysql> -- Nested loop join with extra filter conditions can use parallel query.
mysql> explain select count(*) from part, partsupp where p_partkey != ps_partkey and
 p_name is not null and ps_availqty > 0;
+----+-------------+----------+...+----------
+-------------------------------------------------------------------------------+
| id | select_type | table    |...| rows     | Extra
                                  |
+----+-------------+----------+...+----------
+-------------------------------------------------------------------------------+
|  1 | SIMPLE      | part     |...| 20427936 | Using where; Using parallel query (2
 columns, 1 filters, 0 exprs; 0 extra) |
|  1 | SIMPLE      | partsupp |...| 78164450 | Using where; Using join buffer (Block
 Nested Loop)                       |
+----+-------------+----------+...+----------
+-------------------------------------------------------------------------------+
```

サブクエリ

外部クエリブロックと内部サブクエリブロックでは、そのそれぞれでパラレルクエリを使用するかどうかが決定されます。各ブロックで使用するかどうかは、テーブルの通常の特徴や WHERE 句などに基づきます。例えば、次のクエリでは、外部ブロックではなくサブクエリブロックに対してパラレルクエリが使用されます。

```
mysql> explain select count(*) from part where
   --> p_partkey < (select max(p_partkey) from part where p_name like '%vanilla%');
+----+-------------+...+----------
+-------------------------------------------------------------------------------+
| id | select_type |...| rows     | Extra
                       |
+----+-------------+...+----------
+-------------------------------------------------------------------------------+
|  1 | PRIMARY     |...|     NULL | Impossible WHERE noticed after reading const tables
                        |
|  2 | SUBQUERY    |...| 20427936 | Using where; Using parallel query (2 columns, 0
 filters, 1 exprs; 0 extra) |
+----+-------------+...+----------
+-------------------------------------------------------------------------------+
```

現在、相関サブクエリはパラレルクエリ最適化を使用できません。

UNION

UNION クエリの各クエリブロックは、WHERE の各部分に対して、テーブルの通常の特性、または UNION 句などに基づいてパラレルクエリを使用するかどうかを指定できます。

```
mysql> explain select p_partkey from part where p_name like '%choco_ate%'
    -> union select p_partkey from part where p_name like '%vanil_a%';
+----+---------------+...+----------
+-------------------------------------------------------------------------------+
| id | select_type    |...| rows     | Extra
                          |
+----+---------------+...+----------
+-------------------------------------------------------------------------------+
|  1 | PRIMARY        |...| 20427936 | Using where; Using parallel query (2 columns, 0
 filters, 1 exprs; 0 extra) |
|  2 | UNION          |...| 20427936 | Using where; Using parallel query (2 columns, 0
 filters, 1 exprs; 0 extra) |
| NULL | UNION RESULT | <union1,2> |...|     NULL | Using temporary
                        |
+----+--------------+...+----------
+-------------------------------------------------------------------------------+
```

> **Note**
>
> クエリ内の各 UNION 句は順番に実行されます。クエリにすべてがパラレルクエリを使用する複数のステージが含まれていても、常に 1 つのパラレルクエリしか実行されません。したがって、複雑な複数ステージのクエリであっても、同時並行クエリの制限として 1 つだけカウントされます。

ビュー

オプティマイザは、基になるテーブルを使用して、より長いクエリとしてビューを使用するクエリをすべて書き換えます。したがって、パラレルクエリは、テーブル参照がビューでも実テーブルであっても同じように機能します。クエリに対してパラレルクエリを使用するかどうか、およびプッシュダウンする部分については、最終的に書き直されたクエリに同じ考慮事項が適用されます。

例えば、次のクエリプランは、通常はパラレルクエリを使用しないビューの定義を示しています。追加の WHERE 句でビューがクエリされると、Aurora MySQL はパラレルクエリを使用します。

```
mysql> create view part_view as select * from part;
mysql> explain select count(*) from part_view where p_partkey is not null;
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (1 columns, 0 filters, 0 exprs;
 1 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+
```

データ操作言語 (DML) ステートメント

INSERT 部分がパラレルクエリの他の条件を満たしている場合、SELECT ステートメントは処理の SELECT フェーズに対してパラレルクエリを使用できます。

```
mysql> create table part_subset like part;
mysql> explain insert into part_subset select * from part where p_mfgr =
 'Manufacturer#1';
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (9 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+
```

> **Note**
>
> 通常、INSERT ステートメントの後に、新しく挿入された行のデータがバッファプールにあります。したがって、多数の行を挿入した直後に、テーブルがパラレルクエリに適格でない可能性があります。後で、通常の操作中にバッファプールからデータが除去された後に、テーブルに対するクエリがパラレルクエリを再び使用し始める可能性があります。

ステートメントの CREATE TABLE AS SELECT 部分がパラレルクエリに適格であっても、SELECT ステートメントはパラレルクエリを使用しません。このステートメントの DDL 側面は、パラレルクエリ処理と互換性がありません。対照的に、INSERT ... SELECT ステートメントでは、SELECT 部分はパラレルクエリを使用できます。

パラレルクエリは、DELETE 句のテーブルおよび述語のサイズに関係なく、UPDATEステートメントまたは WHERE ステートメントには使用されません。

```
mysql> explain delete from part where p_name is not null;
+----+-------------+...+----------+-------------+
| id | select_type |...| rows     | Extra       |
+----+-------------+...+----------+-------------+
|  1 | SIMPLE      |...| 20427936 | Using where |
+----+-------------+...+----------+-------------+
```

トランザクションとロック

すべての分離レベルは、Aurora プライマリインスタンスで使用できます。

Aurora リーダー DB インスタンスでは、パラレルクエリは REPEATABLE READ の分離レベルの下で実行されるステートメントに適用されます。Aurora MySQL バージョン 2.09 以降では、リーダー DB インスタンスに対して READ COMMITTED 分離レベルも使用されます。REPEATABLE READ は、Aurora リーダー DB インスタンスのデフォルトの分離レベルです。リーダー DB インスタンスで READ COMMITTED 分離レベルを使用するには、セッションレベルで aurora_read_replica_read_committed 設定オプションを設定する必要があります。READ COMMITTEDリーダーインスタンスの分離レベルは SQL のスタンダード動作に準拠しています。ただし、この分離は、クエリがライターインスタンスで READ COMMITTED 分離レベルを使用する場合よりも、リーダーインスタンスでは厳密ではありません。

Aurora の分離レベル、特にライターインスタンスとリーダーインスタンス間での READ COMMITTED の違いについては、Aurora MySQL の分離レベル を参照してください。

大きなトランザクションが終了した後、テーブルの統計情報は古くなっている可能性があります。このような古くなった統計では、Aurora が正確に行数を見積もるには、ANALYZE TABLE ステートメントが必要になることがあります。大規模な DML ステートメントでは、テーブルデータのかなりの部分がバッファプールに持ち込まれる可能性があります。このデータをバッファプールに入れると、データがプールから削除されるまで、パラレルクエリの選択頻度が低くなります。

セッションが長時間実行トランザクション (デフォルトでは 10 分) 内にある場合、そのセッション内の以降のクエリはパラレルクエリを使用しません。1 つの長期実行クエリ中にもタイムアウトが発生

する可能性があります。このタイプのタイムアウトは、パラレルクエリ処理がスタートされるまでの最大間隔 (現在は 10 分) より長くクエリが実行された場合に発生する可能性があります。

autocommit=1 セッションに mysql を設定して、偶発的に長時間実行トランザクションがスタートされる機会を減らすことができます。テーブルに対する SELECT ステートメントでさえ、読み取りビューを作成してトランザクションをスタートします。読み取りビューは、トランザクションがコミットされるまで続くクエリ用の一貫したデータセットです。このようなアプリケーションは autocommit 設定をオフにして実行する可能性があるため、Aurora で JDBC または ODBC アプリケーションを使用する場合にもこの制限に注意してください。

次の例は、autocommit 設定をオフにして、テーブルに対してクエリを実行すると、暗黙的にトランザクションをスタートする読み取りビューが作成される方法を示しています。すぐ後で実行されるクエリは引き続きパラレルクエリを使用できます。ただし、数分の休止後は、クエリはもはやパラレルクエリの対象となりません。COMMIT または ROLLBACK を使用してトランザクションを終了すると、パラレルクエリの適格性が復元されます。

```
mysql> set autocommit=0;

mysql> explain select sql_no_cache count(*) from part where p_retailprice > 10.0;
+----+...+---------
+-------------------------------------------------------------------------------+
| id |...| rows    | Extra
         |
+----+...+---------
+-------------------------------------------------------------------------------+
|  1 |...| 2976129 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+---------
+-------------------------------------------------------------------------------+

mysql> select sleep(720); explain select sql_no_cache count(*) from part where
 p_retailprice > 10.0;
+------------+
| sleep(720) |
+------------+
|          0 |
+------------+
1 row in set (12 min 0.00 sec)

+----+...+---------+-------------+
| id |...| rows    | Extra       |
+----+...+---------+-------------+
```

```
|  1 |...| 2976129 | Using where |
+----+...+---------+-------------+

mysql> commit;

mysql> explain select sql_no_cache count(*) from part where p_retailprice > 10.0;
+----+...+---------
+-------------------------------------------------------------------------------+
| id |...| rows    | Extra
         |
+----+...+---------
+-------------------------------------------------------------------------------+
|  1 |...| 2976129 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+---------
+-------------------------------------------------------------------------------+
```

クエリが長時間実行トランザクションのためにパラレルクエリに適格でなかった回数を確認するには、ステータス可変 Aurora_pq_request_not_chosen_long_trx を確認します。

```
mysql> show global status like '%pq%trx%';
+---------------------------------------+-------+
| Variable_name                         | Value |
+---------------------------------------+-------+
| Aurora_pq_request_not_chosen_long_trx | 4     |
+---------------------------------+-------+
```

SELECT 構文や SELECT FOR UPDATE 構文などの口ックを取得するすべての SELECT LOCK IN SHARE MODE ステートメントでは、パラレルクエリを使用できません。

パラレルクエリは、LOCK TABLES ステートメントによって口ックされているテーブルに対して機能します。

```
mysql> explain select o_orderpriority, o_shippriority from orders where o_clerk =
 'Clerk#000095055';
+----+...+-----------
+-------------------------------------------------------------------------------+
| id |...| rows      | Extra
           |
+----+...+-----------
+-------------------------------------------------------------------------------+
```

```
|  1 |...| 154545408 | Using where; Using parallel query (3 columns, 1 filters, 0
 exprs; 0 extra) |
+----+...+-----------
+-------------------------------------------------------------------------------+

mysql> explain select o_orderpriority, o_shippriority from orders where o_clerk =
 'Clerk#000095055' for update;
+----+...+-----------+-------------+
| id |...| rows      | Extra       |
+----+...+-----------+-------------+
|  1 |...| 154545408 | Using where |
+----+...+-----------+-------------+
```

B ツリーインデックス

ANALYZE TABLE ステートメントによって収集される統計は、各列のデータの特性に基づいて、パラレルクエリまたはインデックスのルックアップをいつ使用するかをオプティマイザが決定するのに役立ちます。テーブル内のデータを大幅に変更する DML 操作の後で ANALYZE TABLE を実行することにより、統計情報を最新の状態に保ちます。

インデックスのルックアップがデータ集約型スキャンなしで効率的にクエリを実行できる場合は、Aurora は インデックスのルックアップを使用する可能性があります。そうすることにより、パラレルクエリ処理のオーバーヘッドが回避されます。また、どの Aurora DB クラスターでも同時に実行できるパラレルクエリの数には同時実行制限があります。テーブルのインデックス付けにベストプラクティスを使用して、最も頻繁で最も並行性の高いクエリがインデックスのルックアップを使用するようにしてください。

全文検索 (FTS) インデックス

現在のところ、全文検索インデックスを含むテーブルでは、クエリで全文検索インデックスのある列を参照しているか MATCH 演算子を使用しているかどうかにかかわらず、パラレルクエリは使用されません。

仮想列

現在のところ、仮想列を含むテーブルでは、クエリで仮想列を参照しているかどうかにかかわらず、パラレルクエリは使用されません。

組み込みキャッシュメカニズム

Aurora には、組み込みキャッシュメカニズム、つまりバッファプールとクエリキャッシュが組み込まれています。Aurora オプティマイザは、どのクエリが特定のクエリに対して最も効果的かに応じて、これらのキャッシュメカニズムとパラレルクエリを選択します。

パラレルクエリが行をフィルタリングし、列の値を変換して抽出すると、データはデータページではなくタプルとしてヘッドノードに返されます。したがって、パラレルクエリを実行しても、バッファプールにはページが追加されず、既にバッファプールにあるページは削除されます。

Aurora は、バッファプール内に存在するテーブルデータのページ数と、その番号が表すテーブルデータの割合を検証します。Aurora はその情報を使用して、パラレルクエリを使用する方が効率的かどうかを判断します (また、バッファプール内のデータをバイパスします)。または、Aurora はバッファプールにキャッシュされたデータを使用する非パラレルクエリ処理パスを使用することがあります。キャッシュされるページと、データ集約型のクエリがキャッシュおよび削除に与える影響は、バッファプールに関連する構成設定によって異なります。したがって、バッファプール内の常に変化するデータに依存するため、特定のクエリでパラレルクエリが使用されているかどうかを予測することは困難です。

また、Aurora はパラレルクエリに同時実行制限を課します。すべてのクエリがパラレルクエリを使用するわけではないので、複数のクエリによって同時にアクセスされるテーブルは、通常、バッファプール内のデータのかなりの部分を占めます。したがって、Aurora はパラレルクエリに対してこれらのテーブルを選択しないことがよくあります。

同じテーブルで非パラレルクエリのシーケンスを実行すると、データがバッファプールにないため、初期のクエリが遅くなる可能性があります。これでバッファプールが「ウォームアップ」状態になるため、2 番目以降のクエリは非常に高速になります。パラレルクエリは、通常、テーブルに対する初期のクエリからの一貫したパフォーマンスを示します。パフォーマンステストを実行するときは、コールドバッファプールとウォームバッファプールの両方を使用して、非パラレルクエリを評価します。場合によっては、ウォームバッファプールを使用した結果は、パラレルクエリ時間とよく比較できます。その場合、そのテーブルに対するクエリの頻度などの要因を考慮してください。また、そのテーブルのデータをバッファプールに保持するメリットがあるかどうかも考慮してください。

クエリキャッシュは、同じクエリが送信されたときに基になるテーブルのデータに変更がない場合に、クエリがもう一度実行されることを防ぎます。パラレルクエリ機能によって最適化されたクエリは、クエリキャッシュに入り、効果的に再度実行することができます。

> **Note**
>
> パフォーマンスの比較を行うとき、クエリキャッシュは意図的に低いタイミング数を生成する可能性があります。したがって、ベンチマークのような状況では、`sql_no_cache` ヒントを使用できます。このヒントは、以前に同じクエリが実行された場合でも、クエリキャッシュから結果が提供されるのを防ぎます。ヒントは、クエリの `SELECT` ステートメントの直後に表示されます。このトピック内のパラレルクエリ例の多くにはこのヒントが含まれているので、パラレルクエリで有効化されているクエリとそうでないクエリのバージョン間で、クエリ時間を比較できます。
> パラレルクエリの本稼働使用に移行するときは、出典からこのヒントを削除するようにしてください。

オプティマイザヒント

オプティマイザを制御するもう 1 つの方法は、オプティマイザヒントを使用することです。オプティマイザヒントは個々のステートメント内で指定できます。例えば、ステートメント内の 1 つのテーブルの最適化を有効にして、別のテーブルの最適化を無効にすることができます。これらのヒントの詳細については、MySQL リファレンスマニュアルの「オプティマイザヒント」を参照してください。

Aurora MySQL クエリで SQL ヒントを使用して、パフォーマンスを微調整できます。ヒントを使用して、重要なクエリの実行計画が予測不可能な条件のために変更されないようにすることもできます。

SQL ヒント機能を拡張して、クエリプランのオプティマイザの選択を制御できるようにしました。これらのヒントは、パラレルクエリ最適化を使用するクエリに適用されます。詳細については、「Aurora MySQL のヒント」を参照してください。

MyISAM テンポラリテーブル

パラレルクエリの最適化は、InnoDB テーブルにのみ適用されます。Aurora MySQL はテンポラリテーブルの背後で MyISAM を使用するため、テンポラリテーブルを含む内部クエリフェーズではパラレルクエリは使用されません。これらのクエリフェーズは、`Using temporary` 出力に `EXPLAIN` によって示されています。

Amazon Aurora MySQL DB クラスターでのアドバンストな監査の使用

Amazon Aurora MySQL では、パフォーマンスの高い高度な監査機能を使用して、データベースアクティビティを監査できます。そのためには、複数の DB クラスターパラメータを設定することによって監査ログの収集を有効にします。高度な監査を有効にすると、この機能を使用して、サポートされているイベントの任意の組み合わせを記録できます。

監査ログを表示またはダウンロードして、一度に 1 つの DB インスタンスの監査情報を確認することができます。そのためには、「Amazon Aurora ログファイルのモニタリング」に記載された手順を使用します。

> **Tip**
>
> 複数の DB インスタンスを含む Aurora DB クラスターの場合、この方法の方が、クラスター内のすべてのインスタンスの監査ログを調べるために便利なことがあります。これを行うために、CloudWatch Logs を使用します。クラスターレベルで設定を有効にし、Aurora MySQL での監査ログデータを CloudWatch のロググループに発行します。その後、CloudWatch インターフェイスを使用して、監査ログの表示やフィルタリング、および検索が行えます。詳細については、「Amazon CloudWatch Logs への Amazon Aurora MySQL ログの発行」を参照してください。

アドバンストな監査の有効化

DB クラスターの高度な監査を有効にして設定するには、このセクションで説明されているパラメータを使用します。

より高度な監査を有効化または無効化するには、`server_audit_logging` パラメータを使用します。

ログ記録するイベントを指定するには、`server_audit_events` パラメータを使用します。

`server_audit_incl_users` パラメータと `server_audit_excl_users` パラメータを使用して、監査するユーザーを指定します。デフォルトでは、すべてのユーザーが監査されます。一方または両方を空のままにした場合、または両方で同じユーザ名が指定されている場合に、これらのパラメータがどのように機能するかについては、「server_audit_incl_users」ならびに「server_audit_excl_users」を参照してください。

高度な監査は、DB クラスターにより使用されているパラメータグループでこれらのパラメータを設定することにより設定します。「DB パラメータグループのパラメータの変更」に示されている手順を使用し、AWS Management Consoleを使用して DB クラスターパラメータを変更できます。AWS CLI コマンドの modify-db-cluster-parameter-group 、または Amazon RDS API の ModifyDBClusterParameterGroup オペレーションを使用して、プログラム的に DB クラスターパラメータを変更できます。

パラメータグループが既にクラスターに関連付けられていれば、これらのパラメータを変更しても、DB クラスターを再起動する必要はありません。パラメータグループをクラスターに初めて関連付ける場合は、クラスターの再起動が必要です。

トピック

- server_audit_logging
- server_audit_events
- server_audit_incl_users
- server_audit_excl_users

server_audit_logging

高度な監査を有効または無効にします。このパラメータのデフォルトは OFF です。高度な監査を有効にするには、ON に設定します。

監査するイベントのタイプを、`server_audit_events` パラメータを使用して 1 つ以上定義しない限り、監査データは口グに表示されません。

DB インスタンスの監査データが口グに記録されていることを確認するには、そのインスタンスの口グファイルに、`audit/audit.log.`*`other_identifying_information`* の形式で名前が付けられたものがあることを確認します。口グファイルの名前を表示するには、「データベース口グファイルの表示とリスト化」に記載されてい手順に従ってください。

server_audit_events

記録するイベントのコンマ区切りリストが含まれています。イベントはすべて大文字で指定する必要があります。リスト要素間に空白があってはいけません。例: `CONNECT,QUERY_DDL`。このパラメータのデフォルトは空の文字列です。

次のイベントの任意の組み合わせを記録できます。

- CONNECT - 成功した接続と失敗した接続の両方、および切断を記録します。このイベントにはユーザー情報が含まれています。
- QUERY - すべてのクエリをプレーンテキストで記録します (構文またはアクセス権限エラーで失敗したエラーを含む)。

> **Tip**
>
> このイベントタイプを有効にすると、監査データには、Aurora が自動的に行う継続的なモニタリングとヘルスチェックに関する情報が含まれます。特定の種類のオペレーションのみを確認したい場合は、イベントの種類をより具体的に指定できます。また、CloudWatch インターフェイスを使用して、ログ内の特定のデータベース、テーブル、またはユーザーに関連するイベントを検索することもできます。

- QUERY_DCL - QUERY イベントと同様ですが、データ制御言語 (DCL) クエリ (GRANT、REVOKE など) のみ返します。
- QUERY_DDL - QUERY イベントと同様ですが、データ定義言語 (DDL) クエリ (CREATE、ALTER など) のみ返します。
- QUERY_DML - QUERY イベントと同様ですが、データ操作言語 (DML) クエリ (INSERT、UPDATE などと、SELECT) のみ返します。
- TABLE - クエリ実行の影響を受けたテーブルを記録します。

server_audit_incl_users

アクティビティを記録するユーザーのカンマ区切りのユーザー名リストが含まれています。リストの要素間にスペースは挿入しないでください。例えば、`user_3,user_4` などです。このパラメータのデフォルトは空の文字列です。最大長は 1024 文字です。指定したユーザー名は、`User` テーブルの `mysql.user` 列の対応する値と一致する必要があります。ユーザー名の詳細については、MySQL のドキュメントの「アカウントユーザー名とパスワード」を参照してください。

`server_audit_incl_users` と `server_audit_excl_users` の両方が空 (デフォルト) の場合、すべてのユーザーが監査されます。

ユーザーを `server_audit_incl_users` に追加して `server_audit_excl_users` を空のままにした場合、それらのユーザーだけが検査されます。

ユーザーを server_audit_excl_users に追加して server_audit_incl_users を空のままにした場合、server_audit_excl_users にリストされているものを除き、すべてのユーザーが監査されます。

server_audit_excl_users と server_audit_incl_users の両方に同じユーザーを追加した場合、それらのユーザーが監査されます。同じユーザーが両方の設定にリストされている場合、server_audit_incl_users に対し、より高い優先順位が与えられます。

接続および切断イベントは、この可変の影響を受けません。指定された場合は常に記録されます。server_audit_incl_users の方が優先順位が高いため、ユーザーが server_audit_excl_users パラメータでも指定されている場合も、そのユーザーは記録されます。

server_audit_excl_users

アクティビティがを記録しないユーザーのカンマ区切りのユーザー名リストが含まれています。リストの要素間にスペースは挿入しないでください。例えば、rdsadmin,user_1,user_2 などです。このパラメータのデフォルトは空の文字列です。最大長は 1024 文字です。指定したユーザー名は、User テーブルの mysql.user 列の対応する値と一致する必要があります。ユーザー名の詳細については、MySQL のドキュメントの「アカウントユーザー名とパスワード」を参照してください。

server_audit_incl_users と server_audit_excl_users の両方が空 (デフォルト) の場合、すべてのユーザーが監査されます。

ユーザーを server_audit_excl_users に追加して server_audit_incl_users を空のままにすると、server_audit_excl_users にリストしたユーザーだけを監査せずに、他のすべてのユーザーを監査することができます。

server_audit_excl_users と server_audit_incl_users の両方に同じユーザーを追加した場合、それらのユーザーが監査されます。同じユーザーが両方の設定にリストされている場合、server_audit_incl_users に対し、より高い優先順位が与えられます。

接続および切断イベントは、この可変の影響を受けません。指定された場合は常に記録されます。ユーザーが server_audit_incl_users パラメータでも指定されている場合、そのユーザーは記録されます。この設定の方が server_audit_excl_users より優先順位が高いためです。

監査ログの表示

監査ログを表示およびダウンロードするには、コンソールを使用します。[データベース] ページで、DB インスタンスをクリックして詳細を表示し、[ログ] セクションま

でスクロールします。高度な監査機能によって生成される監査ログには、`audit/audit.log.other_identifying_information` の形式で名前が付けられています。

ログファイルをダウンロードするには、[ログ] セクションでファイルを選択してから、[ダウンロード] を選択します。

describe-db-log-files AWS CLI コマンドを使用して、ログファイルのリストを取得することもできます。ログファイルの内容は、AWS CLI の download-db-log-file-portion コマンドを実行してダウンロードできます。詳細については、「データベースログファイルの表示とリスト化」および「データベースログファイルのダウンロード」を参照してください。

監査ログの詳細

ログファイルは、UTF-8 形式のカンマ区切り変数 (CSV) ファイルとして表されます。クエリも一重引用符 (') で囲まれます。

監査ログは、各インスタンスのローカルストレージに個別に保存されます。各 Aurora インスタンスは、一度に 4 つのログファイルに書き込みを分散します。ログの最大サイズは、合計で 100 MB です。この設定不可能な制限に達すると、Aurora はファイルを回転し、4 つの新しいファイルを生成します。

> **Tip**
>
> ログファイルのエントリは、順番になっていません。エントリを順序付けするには、タイムスタンプ値を使用します。最新のイベントを表示するには、すべてのログファイルの確認が必要な場合があります。ログデータの並べ替えと検索をより柔軟に行うためには、監査ログを CloudWatch にアップロードするための設定を有効にし、CloudWatch インターフェイスを使用してそれらを表示します。
> より多くのタイプのフィールドを含み、JSON 形式で出力された監査データを表示するには、データベースのアクティビティストリーム機能を使用することもできます。詳細については、「データベースアクティビティストリームを使用した Amazon Aurora のモニタリング」を参照してください。

監査ログファイルの行には、次のカンマ区切りの情報が指定された順序で含まれています。

フィールド	説明
timestamp	記録されたイベントの UNIX タイムスタンプ (マイクロ秒の精度)。

フィールド	説明
serverhost	イベントが記録されているインスタンスの名前。
username	ユーザーの接続されたユーザー名。
host	ユーザーの接続元のホスト。
connectionid	記録されたオペレーションの接続 ID 番号。
queryid	クエリ ID 番号。リレーショナルテーブルイベントと関連するクエリの検索に使用できます。TABLE イベントの場合、複数の行が追加されます。
オペレーション	記録されたアクションの種類。指定できる値は CONNECT、QUERY、READ、WRITE、CREATE、ALTER、RENAME、DROP です。
データベース	USE コマンドにより設定されたアクティブなデータベース。
オブジェクト	QUERY イベントの場合、この値は、データベースが実行したクエリを示します。TABLE イベントの場合、テーブル名を示します。
retcode	記録されたオペレーションのリターンコード。

Amazon Aurora MySQL でのレプリケーション

Aurora MySQL レプリケーション機能は、クラスターの高可用性とパフォーマンスにとって重要な要素です。Aurora では、最大 15 個の Aurora レプリカを使用して、簡単にクラスターの作成またはサイズ変更を行うことができます。

すべてのレプリカは同じ基盤となるデータから機能します。一部のデータベースインスタンスがオフラインになると、他のデータベースインスタンスがクエリの処理を続行します。あるいは、必要に応じてライターの機能を引き継ぐことができます。Aurora は複数のデータベースインスタンスに読み取り専用接続を自動的に分散させ、Aurora クラスターがクエリ負荷の重いワークロードをサポートできるようにします。

以下のトピックでは、Aurora MySQL レプリケーションのしくみと、レプリケーション設定を最大限の可用性とパフォーマンスを得るために調整する方法について説明します。

トピック

- Aurora レプリカの使用
- Amazon Aurora MySQL のレプリケーションオプション
- Amazon Aurora MySQL レプリケーションのパフォーマンスに関する考慮事項
- ダウンタイムのない再起動 (ZDR) (Amazon Aurora MySQL 用)
- Aurora MySQL でのレプリケーションフィルターの設定
- Amazon Aurora MySQL レプリケーションのモニタリング
- Amazon Aurora MySQL DB クラスターでのローカル書き込み転送の使用
- AWS リージョン 間での Amazon Aurora MySQL DB クラスターのレプリケーション
- Aurora と MySQL との間、または Aurora と別の Aurora DB クラスターとの間のレプリケーション (バイナリログレプリケーション)
- GTID ベースレプリケーションを使用する

Aurora レプリカの使用

Aurora レプリカは、Aurora DB クラスター内の独立したエンドポイントであり、読み取りオペレーションのスケーリングと可用性の向上に最適です。最大 15 個の Aurora レプリカを、AWS リージョン 内で DB クラスターがまたがるアベイラビリティーゾーン全体に分散できます。DB クラスターボリュームは DB クラスターのデータの複数のコピーで構成されますが、クラスターボリューム内の

データは、DB クラスター内のプライマリインスタンスと Aurora レプリカに対する単一の論理的なボリュームとして表されます。Aurora レプリカの詳細については、「Aurora レプリカ」を参照してください。

Aurora レプリカは、クラスターボリュームでの読み取りオペレーションに特化しているため、読み取りのスケーリングに最適です。書き込みオペレーションはプライマリインスタンスによって管理されます。クラスターボリュームは Aurora MySQL DB クラスター内のすべてのインスタンスで共有されるため、Aurora レプリカごとにデータのコピーをレプリケートする追加作業は必要ありません。対照的に、MySQL のリードレプリカは、単一スレッドで、出典 DB インスタンスからローカルデータストアへのすべての書き込みオペレーションを再生する必要があります。この制限により、大量の読み取りトラフィックをサポートする MySQL リードレプリカの能力に影響を与える可能性があります。

Aurora MySQL では、Aurora レプリカが削除されるとそのインスタンスエンドポイントは直ちに削除され、Aurora レプリカもリーダーエンドポイントから削除されます。削除中の Aurora レプリカで実行されているステートメントがある場合は、削除までに 3 分の猶予期間があります。既存のステートメントは、猶予期間中に適切に終了する場合があります。猶予期間が終了すると、Aurora レプリカはシャットダウンし、削除されます。

Important

Aurora MySQL の Aurora レプリカは常に、InnoDB テーブル上のオペレーションに、デフォルトのトランザクション分離レベル `REPEATABLE READ` を使用します。`SET TRANSACTION ISOLATION LEVEL` コマンドは、Aurora MySQL DB クラスターのプライマリインスタンスのトランザクションレベルを変更する場合にのみ使用できます。この制限により、Aurora レプリカ上でのユーザーレベルのロックを回避し、Aurora レプリカがレプリカの遅延を最小限に抑えたまま何千ものアクティブユーザーの接続のサポートをできるようにします。

Note

プライマリインスタンスで実行された DDL ステートメントにより、関連付けられた Aurora レプリカのデータベース接続が中断される場合があります。Aurora レプリカ接続でテーブルなどのデータベースオブジェクトを使用中である場合、そのオブジェクトが DDL ステートメントを使用してプライマリインスタンスで変更されると、Aurora レプリカ接続は中断されます。

> **Note**
>
> 中国 (寧夏) リージョンは、クロスリージョンリードレプリカをサポートしません。

Amazon Aurora MySQL のレプリケーションオプション

次のいずれのオプションの間でもレプリケーションを設定できます。

- 異なる AWS リージョン の 2 つの Aurora MySQL DB クラスター (Aurora MySQL DB クラスターのクロスリージョンリードレプリカを作成)。

  詳細については、「AWS リージョン 間での Amazon Aurora MySQL DB クラスターのレプリケーション」を参照してください。

- 同一の AWS リージョン の 2 つの Aurora MySQL DB クラスター (MySQL バイナリログ (binlog) のレプリケーションを使用)。

  詳細については、「Aurora と MySQL との間、または Aurora と別の Aurora DB クラスターとの間のレプリケーション (バイナリログレプリケーション)」を参照してください。

- 出典としての RDS for MySQL DB インスタンスと Aurora MySQL DB クラスター (RDS for MySQL DB インスタンスの Aurora リードレプリカを作成)。

  このアプローチを使用すると、Aurora への移行中に既存および進行中のデータ変更を Aurora MySQL に取り込むことができます。詳細については、「Aurora リードレプリカを使用した、RDS for MySQL DB インスタンスから Amazon Aurora MySQL DB クラスターへのデータの移行」を参照してください。

  また、このアプローチを使用して、データの読み取りクエリのスケーラビリティを向上させることもできます。これを行うには、読み取り専用 Aurora MySQL クラスター内の複数の DB インスタンスを使用してデータをクエリします。詳細については、「Amazon Aurora を使用した MySQL データベースの読み取りスケーリング」を参照してください。

- 1 つの AWS リージョン 内の Aurora MySQL DB クラスターと、異なるリージョンにある最大 5 つの Aurora 読み取り専用 Aurora MySQL DB クラスター (Aurora Global Database を作成)。

  Aurora Global Database を使用して、ワールドワイドなフットプリントを持つアプリケーションをサポートできます。プライマリ Aurora MySQL DB クラスターには、ライターインスタンスと最大 15 個の Aurora レプリカがあります。読み取り専用セカンダリ Aurora MySQL DB クラスター

は、それぞれ最大 16 個の Aurora レプリカで構成できます。詳細については、「Amazon Aurora Global Database の使用」を参照してください。

> **Note**
>
> また、Amazon Aurora DB クラスターのプライマリインスタンスを再起動すると、DB クラスター全体の一貫した読み取り/書き込みを保証するエントリポイントを再確立するために、DB クラスターの Aurora レプリカも自動的に再起動します。

Amazon Aurora MySQL レプリケーションのパフォーマンスに関する考慮事項

以下の機能を使用して、Aurora MySQL レプリケーションのパフォーマンスを微調整することができます。

レプリカログ圧縮機能は、レプリケーションメッセージのネットワーク帯域幅を自動的に縮小します。各メッセージはすべての Aurora レプリカに送信されるため、大規模なクラスターではメリットが大きくなります。この機能には、圧縮を実行する書き込みノードの CPU オーバーヘッドが含まれます。Aurora MySQL バージョン 2 とバージョン 3 では、常に有効になっています。

バイナリログフィルタ処理機能は、レプリケーションメッセージのネットワーク帯域幅を自動的に縮小します。レプリケーションメッセージに含まれるバイナリログ情報は、Aurora レプリカで使用されないため、これらのノードに送信されるメッセージからは除外されます。

Aurora MySQL バージョン 2 の場合、`aurora_enable_repl_bin_log_filtering` パラメーターを変更することにより、この機能を制御できます。このパラメータはデフォルトでオンになっています。この最適化は透過的であることを意図されているため、この設定をオフにできるのは、レプリケーションに関する問題の診断時またはトラブルシューティング時に限られる場合があります。この機能を使用できない古い Aurora MySQL クラスターなどの場合は、その動作に合わせてこの設定をオフにすることができます。

Aurora MySQL バージョン 3 の場合、バイナリログフィルタリングは常に有効です。

ダウンタイムのない再起動 (ZDR) (Amazon Aurora MySQL 用)

ダウンタイムのない再起動 (ZDR) 機能は、特定の種類の再起動中に DB インスタンスへのアクティブな接続の一部または全部を保持できます。ZDR は、Aurora がエラー条件 (レプリカが出典より遅すぎるなど) を解決するために自動的に実行する再起動に適用されます。

> **Important**
>
> ZDR メカニズムはベストエフォートベースで動作します。ZDR が適用される場合を決定する Aurora MySQL バージョン、インスタンスクラス、エラー条件、互換性のある SQL オペレーション、およびその他の要因は、いつでも変更される可能性があります。

Aurora MySQL 2.x の ZDR にはバージョン 2.10 以降が必要です。ZDR は Aurora MySQL 3.x のすべてのマイナーバージョンで利用可能です。Aurora MySQL バージョン 2 と 3 では、ZDR メカニズムはデフォルトでオンになっており、Aurora は `aurora_enable_zdr` パラメータを使用しません。

Aurora は、停止時間ゼロの再起動に関連するアクティビティに関するレポートを、[イベント] ページに表示します。Aurora は、ZDR メカニズムを使用して再起動を試みる際にイベントを記録します。このイベントは、Aurora が再起動を実行する理由を示します。その後、Aurora は再起動の完了時に別のイベントを記録します。この最終イベントは、再起動中にプロセスに要した時間、および保持またはドロップされた接続の数を報告します。データベースエラーログを参照して、再起動中に発生した処理の詳細を確認できます。

ZDR オペレーションが成功した後も接続はそのまま残りますが、一部の可変と機能は再初期化されます。次の種類の情報は、ダウンタイムのない再起動によって生じる再起動を通じては保持されません。

- グローバル可変 Aurora はセッション可変を復元しますが、再起動後のグローバル可変の復元は行いません。
- ステータス可変。特に、エンジンのステータスによって報告される稼働時間の値はリセットされます。
- `LAST_INSERT_ID`.
- テーブルのメモリ内 `auto_increment` 状態。メモリ内自動インクリメント状態が再初期化されます。自動インクリメント値の詳細については、MySQL リファレンスマニュアルを参照してください。
- `INFORMATION_SCHEMA` および `PERFORMANCE_SCHEMA` テーブルからの診断情報。この診断情報は、`SHOW PROFILE` や `SHOW PROFILES` などのコマンドの出力にも表示されます。

次の表では、クラスター内の DB インスタンスを再起動するときに Aurora が ZDR メカニズムを使用できるかどうかを決定するバージョン、インスタンスロール、およびその他の状況を示しています。

Aurora MySQL バージョン	ZDR はライターに適用されますか?	ZDR はリーダーに適用されますか？	ZDR は常に有効ですか？	注意
2.x、2.10.0 未満	いいえ	いいえ	該当なし	ZDR はこれらのバージョンでは利用できません。
2.10.0 ~ 2.11.0	はい	はい	はい	Aurora は、アクティブな接続で進行中のトランザクションをロールバックします。アプリケーションはトランザクションを再試行する必要があります。 Aurora は、TLS/SSL、一時テーブル、テーブルロック、またはユーザーロックを使用するすべての接続をキャンセルします。
2.11.1 以降	はい	はい	はい	Aurora は、アクティブな接続で進行中のトランザクションをロールバックします。アプリケーションはトランザクションを再試行する必要があります。 Aurora は、一時テーブル、テーブルロック、またはユーザーロックを使用するすべての接続をキャンセルします。
3.01 ~ 3.03	はい	はい	はい	Aurora は、アクティブな接続で進行中のトランザクションをロールバックします。アプリケーションはトランザクションを再試行する必要があります。 Aurora は、TLS/SSL、一時テーブル、テーブルロック、またはユーザーロックを使用するすべての接続をキャンセルします。
3.04 以上	はい	はい	はい	Aurora は、アクティブな接続で進行中のトランザクションをロールバックします。アプリ

Aurora MySQL バージョン	ZDR はライターに適用されますか?	ZDR はリーダーに適用されますか？	ZDR は常に有効ですか？	注意
				ケーションはトランザクションを再試行する必要があります。 Aurora は、一時テーブル、テーブルロック、またはユーザーロックを使用するすべての接続をキャンセルします。

Aurora MySQL でのレプリケーションフィルターの設定

レプリケーションフィルターを使用して、リードレプリカでレプリケートするデータベースとテーブルを指定できます。レプリケーションフィルターは、データベースとテーブルをレプリケーションに含めることも、レプリケーションから除外することもできます。

レプリケーションフィルターの使用例は以下のとおりです。

- リードレプリカのサイズを縮小します。レプリケーションフィルタリングを使用すると、リードレプリカで必要のないデータベースとテーブルを除外できます。
- セキュリティ上の理由から、データベースとテーブルをリードレプリカから除外するため。
- 異なるリードレプリカで、特定のユースケースごとにさまざまなデータベースとテーブルを複製するため。例えば、分析やシャーディングに特定のリードレプリカを使用できます。
- 異なる AWS リージョン にリードレプリカがある DB クラスターで、異なる AWS リージョン に異なるデータベースまたはテーブルを複製するには。
- インバウンドレプリケーショントポロジでレプリカとして設定されている Aurora MySQL DB クラスターでレプリケートするデータベースとテーブルを指定するには。この設定の詳細については、「Aurora と MySQL との間、または Aurora と別の Aurora DB クラスターとの間のレプリケーション (バイナリログレプリケーション)」を参照してください。

トピック

- Aurora MySQL のレプリケーションフィルターパラメータの設定
- Aurora MySQL のレプリケーションフィルターの制限

- Aurora MySQL のレプリケーションフィルターの例
- リードレプリカのレプリケーションフィルターを表示する

Aurora MySQL のレプリケーションフィルターパラメータの設定

レプリケーションフィルターを設定するには、次のパラメータを設定します。

- binlog-do-db - 指定されたバイナリログテーブルに変更を複製します。バイナリログソースクラスターに対してこのパラメータを設定すると、パラメータで指定されたバイナリログのみが複製されます。
- binlog-ignore-db - 指定されたバイナリログテーブルに変更を複製しません。バイナリログソースクラスターに対して binlog-do-db パラメータが設定されている場合、このパラメータは評価されません。
- replicate-do-db - 指定したデータベースに変更を複製します。バイナリログソースクラスターに対してこのパラメータを設定すると、パラメータで指定されたデータベースのみが複製されます。
- replicate-ignore-db - 指定したデータベースに変更を複製しないでください。バイナリログレプリカスクラスターに対して replicate-do-db パラメータが設定されている場合、このパラメータは評価されません。
- replicate-do-table -指定されたテーブルに変更を複製します。このパラメータをリードレプリカに設定した場合、パラメータで指定したテーブルのみが複製されます。また、replicate-do-db または replicate-ignore-db パラメータが設定されている場合は、指定されたテーブルを含むデータベースを、必ずバイナリログレプリカクラスターのレプリケーションに含めます。
- replicate-ignore-table - 指定したテーブルに変更を複製しないでください。バイナリログレプリカスクラスターに対して replicate-do-table パラメータが設定されている場合、このパラメータは評価されません。
- replicate-wild-do-table - 指定したデータベースおよびテーブル名のパターンに基づいてテーブルを複製します。% および _ ワイルドカードの文字がサポート対象となります。replicate-do-db または replicate-ignore-db パラメータが設定されている場合は、指定されたテーブルを含むデータベースを、必ずバイナリログレプリカクラスターのレプリケーションに含めます。
- replicate-wild-ignore-table - 指定したデータベースおよびテーブル名のパターンに基づいてテーブルを複製しないでください。% および _ ワイルドカードの文字がサポート対象となります。バイナリログレプリカスクラスターに対して replicate-do-table または replicate-wild-do-table パラメータが設定されている場合、このパラメータは評価されません。

パラメータは、記載されている順序に沿って評価されます。これらのパラメータの詳細な仕組みについては、MySQL のドキュメントを参照してください。

- 一般的な情報については、 Replica Server Options and Variables を参照してください。
- データベースレプリケーションのフィルターパラメータを評価する方法については、 Evaluation of Database-Level Replication and Binary Logging Options を参照してください。
- テーブルレプリケーションのフィルターパラメータを評価する方法については、Evaluation of Table-Level Replication Options を参照してください。

デフォルトでは、これらの各パラメータの値は空です。各バイナリログクラスターで、これらのパラメータを使用してレプリケーションフィルターを設定、変更、削除することができます。これらのパラメータの 1 つを設定する場合は、各フィルターを他のフィルターとコンマで区切ります。

% および _ パラメータで `replicate-wild-do-table` および `replicate-wild-ignore-table` ワイルドカードの文字を使用できます。% ワイルドカードは任意の文字数と一致し、_ ワイルドカードは 1 文字のみと一致します。

ソース DB インスタンスのバイナリログ形式は、データ変更のレコードを決定するため、レプリケーションでは重要です。`binlog_format` パラメータの設定により、レプリケーションが行ベースかステートメントベースかが決まります。詳細については、「Aurora MySQL バイナリログの設定」を参照してください。

> **Note**
>
> ソース DB インスタンスの `binlog_format` 設定に関係なく、すべてのデータ定義言語 (DDL) ステートメントはステートメントとして複製されます。

Aurora MySQL のレプリケーションフィルターの制限

Aurora MySQL のレプリケーションフィルターには、次の制限が適用されます。

- レプリケーションフィルターは、Aurora MySQL バージョン 3 でのみサポートされます。
- 各レプリケーションフィルターのパラメータには、2,000 文字といった制限があります。
- レプリケーションフィルターでは、カンマはサポートされていません。
- レプリケーションフィルタリングは、XA トランザクションをサポートしていません。

  詳細については、MySQL ドキュメントの「XA トランザクションの制限」を参照してください。

Aurora MySQL のレプリケーションフィルターの例

リードレプリカのレプリケーションフィルタリングを設定するには、リードレプリカに関連付けられている DB クラスターパラメータグループのレプリケーションフィルタリングパラメータを変更します。

> **Note**
>
> デフォルトの DB クラスターパラメータグループを変更することはできません。リードレプリカがデフォルトのパラメータグループを使用している場合は、新しいパラメータグループを作成してリードレプリカに関連付けます。DB クラスターパラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。

AWS Management Console、AWS CLI、または RDS API を使用して、DB クラスターパラメータグループのパラメータを設定できます。パラメータの設定の詳細については、「DB パラメータグループのパラメータの変更」を参照してください。DB クラスターパラメータグループのパラメータを設定すると、そのパラメータグループに関連付けられているすべての DB クラスターがパラメータ設定を使用します。DB クラスターパラメータグループでレプリケーションフィルタリングパラメータを設定する場合は、パラメータグループがリードレプリカクラスターにのみ関連付けられていることを確認してください。ソース DB インスタンスのレプリケーションフィルターのパラメータは空のままにします。

次の例では、AWS CLI を使用してパラメータを設定します。これらの例では、CLI コマンドが完了した直後にパラメータの変更が行われるように `ApplyMethod` を `immediate` に設定しています。リードレプリカの再起動後に保留中の変更を適用する場合は、`ApplyMethod` を `pending-reboot` に設定します。

以下の例では、レプリケーションフィルターを設定します。

- Including databases in replication
- Including tables in replication
- Including tables in replication with wildcard characters
- Excluding databases from replication
- Excluding tables from replication
- Excluding tables from replication using wildcard characters

Example レプリケーションにデータベースを含める

次の例では、レプリケーションに mydb1 データベースと mydb2 データベースが含まれています。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-do-
db,ParameterValue='mydb1,mydb2',ApplyMethod=immediate"
```

Windows の場合:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-do-
db,ParameterValue='mydb1,mydb2',ApplyMethod=immediate"
```

Example レプリケーションにテーブルを含める

次の例には、レプリケーションのデータベース table1 の table2 テーブルと mydb1 テーブルが含まれています。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-do-
table,ParameterValue='mydb1.table1,mydb1.table2',ApplyMethod=immediate"
```

Windows の場合:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-do-
table,ParameterValue='mydb1.table1,mydb1.table2',ApplyMethod=immediate"
```

Example ワイルドカードの文字を使用してレプリケーションにテーブルを含める

次の例には、レプリケーションのデータベース order の return および mydb で始まる名前のテーブルが含まれています。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-wild-do-table,ParameterValue='mydb.order
%,mydb.return%',ApplyMethod=immediate"
```

Windows の場合:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-wild-do-table,ParameterValue='mydb.order
%,mydb.return%',ApplyMethod=immediate"
```

Example レプリケーションからデータベースを除外する

次の例では、mydb5 データベースと mydb6 データベースをレプリケーションから除外しています。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-ignore-
db,ParameterValue='mydb5,mydb6',ApplyMethod=immediate"
```

Windows の場合:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-ignore-
db,ParameterValue='mydb5,mydb6,ApplyMethod=immediate"
```

Example レプリケーションからテーブルを除外する

次の例では、データベース mydb5 のテーブル table1 とデータベース mydb6 の table2 をレプリケーションから除外しています。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster-parameter-group \
```

```
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-ignore-
table,ParameterValue='mydb5.table1,mydb6.table2',ApplyMethod=immediate"
```

Windows の場合:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-ignore-
table,ParameterValue='mydb5.table1,mydb6.table2',ApplyMethod=immediate"
```

Example ワイルドカードの文字を使用したレプリケーションからテーブルを除外する

次の例では、データベース `order` の `return` および `mydb7` で始まる名前のテーブルをレプリケーションから除外しています。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-wild-ignore-table,ParameterValue='mydb7.order
%,mydb7.return%',ApplyMethod=immediate"
```

Windows の場合:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-wild-ignore-table,ParameterValue='mydb7.order
%,mydb7.return%',ApplyMethod=immediate"
```

リードレプリカのレプリケーションフィルターを表示する

リードレプリカのレプリケーションフィルターは、次の方法で表示できます。

- リードレプリカに関連付けられているパラメータグループのレプリケーションフィルタリングパラメータの設定を確認してください。

  手順については、「DB パラメータグループのパラメータ値を表示する」を参照してください。
- MySQL クライアントで、リードレプリカに接続し、`SHOW REPLICA STATUS` ステートメントを実行します。

出力の次のフィールドには、リードレプリカのレプリケーションフィルターが表示されます。

- Binlog_Do_DB
- Binlog_Ignore_DB
- Replicate_Do_DB
- Replicate_Ignore_DB
- Replicate_Do_Table
- Replicate_Ignore_Table
- Replicate_Wild_Do_Table
- Replicate_Wild_Ignore_Table

これらのフィールドの詳細については、MySQL のドキュメントの Checking Replication Status を参照してください。

Amazon Aurora MySQL レプリケーションのモニタリング

読み取りのスケーリングと高可用性は最短遅延時間に左右されます。Amazon CloudWatch AuroraReplicaLag メトリクスをモニタリングすることによって、Aurora レプリカが Aurora MySQL DB クラスターのプライマリインスタンスからどれくらい遅延しているかを確認できます。AuroraReplicaLag メトリクスは各 Aurora レプリカに記録されます。

プライマリ DB インスタンスは、AuroraReplicaLagMaximum メトリクスと AuroraReplicaLagMinimum Amazon CloudWatch メトリクスも記録します。AuroraReplicaLagMaximum メトリクスは、DB クラスターのプライマリ DB インスタンスと各 Aurora レプリカの間の最大遅延量を記録します。AuroraReplicaLagMinimum メトリクスは、DB クラスターのプライマリ DB インスタンスと各 Aurora レプリカの間の最小遅延量を記録します。

Aurora レプリカの遅延について最新の値が必要な場合は、Amazon CloudWatch で AuroraReplicaLag メトリクスを確認できます。Aurora レプリカの遅延は、information_schema.replica_host_status テーブル内にある Aurora MySQL DB クラスターの各 Aurora レプリカにも記録されます。このテーブルの詳細については、「information_schema.replica_host_status」を参照してください。

RDS インスタンスと CloudWatch メトリックスのモニタリングの詳細については、「Amazon Aurora クラスターでのメトリクスのモニタリング」を参照してください。

Amazon Aurora MySQL DB クラスターでのローカル書き込み転送の使用

ローカル (クラスター内) 書き込み転送により、アプリケーションは Aurora レプリカで直接、読み取り/書き込みトランザクションを発行できます。その後、これらのトランザクションはライター DB インスタンスに転送されてコミットされます。アプリケーションで書き込み後の読み取り一貫性 (トランザクション内の最新の書き込みを読み取る機能) が必要なときには、ローカル書き込み転送を使用できます。

リードレプリカは、ライターから非同期で更新を受け取ります。書き込み転送を行わないと、書き込み後の読み取り一貫性を必要とする読み取りをライター DB インスタンス上で処理する必要があります。または、複数のリードレプリカを活用してスケーラビリティを高めるために、複雑なカスタムアプリケーションロジックを開発する必要があります。アプリケーションは、トラフィックを正しいエンドポイントに送信するために、すべての読み取りトラフィックと書き込みトラフィックを完全に分割して、2 セットのデータベース接続を維持する必要があります。クエリがアプリケーション内の単一の論理セッション、つまりトランザクションの一部である場合、この開発オーバーヘッドはアプリケーションの設計を複雑にします。さらに、レプリケーションの遅延はリードレプリカによって異なる場合があるため、データベース内のすべてのインスタンスでグローバルな読み取りの一貫性を実現することは困難です。

書き込み転送により、これらのトランザクションを分割したり、ライターのみに送信したりする必要がなくなるため、アプリケーション開発が簡単になります。この新機能により、トランザクション内の最新の書き込みを読み取る必要があり、書き込み遅延の影響を受けないワークロードの読み取りスケールを簡単に実現できます。

ローカル書き込み転送は、Aurora グローバルデータベースのセカンダリ DB クラスターからプライマリ DB クラスターに書き込みを転送するグローバル書き込み転送とは異なります。Aurora グローバルデータベースの一部である DB クラスターでは、ローカル書き込み転送を使用できます。詳細については、「Amazon Aurora Global Database の書き込み転送を使用する」を参照してください。

ローカル書き込み転送には Aurora MySQL バージョン 3.04 以降が必要です。

トピック

- 書込み転送を使用したトランザクション
- 書き込み転送の設定パラメータ
- 書き込み転送のための Amazon CloudWatch メトリクスと Aurora MySQL ステータス変数
- 転送されたトランザクションとクエリの識別

ローカル書き込み転送の有効化

デフォルトでは、Aurora MySQL DB クラスターのローカル書き込み転送は有効になっていません。ローカル書き込み転送は、インスタンスレベルではなくクラスターレベルで有効にします。

> **Important**
>
> バイナリログギングを使用するクロスリージョンリードレプリカに対してローカル書き込み転送を有効にすることもできますが、書き込み操作はソース AWS リージョン に転送されません。これらは、binlog リードレプリカクラスターのライター DB インスタンスに転送されます。
> この方法は、セカンダリ AWS リージョン の binlog リードレプリカへの書き込みの用途がある場合にのみ使用してください。そうしないと、複製されたデータセットが互いに矛盾する「スプリットブレーン」シナリオになってしまう可能性があります。
> どうしても必要な場合を除いて、クロスリージョンのリードレプリカでのローカル書き込み転送ではなく、グローバルデータベースでのグローバル書き込み転送を使用することをお勧めします。詳細については、「Amazon Aurora Global Database の書き込み転送を使用する」を参照してください。

コンソール

DB クラスターを作成または変更するときに、AWS Management Console を使用して、[リードレプリカ書き込み転送] の [ローカル書き込み転送を有効にする] チェックボックスを選択します。

AWS CLI

AWS CLI で書き込み転送を有効にするには、`--enable-local-write-forwarding` オプションを使用します。このオプションは、`create-db-cluster` コマンドを使用して新しい DB クラスターを作成するときに機能します。`modify-db-cluster` コマンドを使用して、既存の DB クラスターを変更する場合にも機能します。これらの同じ CLI コマンドで `--no-enable-local-write-forwarding` オプションを使用することで、書き込み転送をオフにすることができます。

次の例では、書き込み転送を有効にした Aurora MySQL DB クラスターを作成します。

```
aws rds create-db-cluster \
  --db-cluster-identifier write-forwarding-test-cluster \
  --enable-local-write-forwarding \
  --engine aurora-mysql \
  --engine-version 8.0.mysql_aurora.3.04.0 \
  --master-username myuser \
  --master-user-password mypassword \
  --backup-retention 1
```

次に、書き込み転送を使用できるように、ライター DB インスタンスとリーダー DB インスタンスを作成します。詳細については、「Amazon Aurora DB クラスターの作成」を参照してください。

RDS API

Amazon RDS API を使用して書き込み転送を有効にするには、EnableLocalWriteForwarding パラメータを true に設定します。このパラメータは、CreateDBCluster オペレーションを使用して新しい DB クラスターを作成するときに機能します。この操作は、ModifyDBCluster オペレーションを使用して既存の DB クラスターを変更する場合にも機能します。EnableLocalWriteForwarding パラメータを false に設定することで、書き込み転送をオフにすることができます。

データベースセッションの書き込み転送を有効にする

aurora_replica_read_consistency パラメータは、書き込み転送を有効にする DB パラメータと DB クラスターパラメータです。読み取り整合性レベルには、EVENTUAL、SESSION、または GLOBAL を指定できます。整合性レベルの詳細については、書き込み転送の読み取り整合性 を参照してください。

このパラメータには、次の規則が適用されます。

- デフォルト値は null です。
- 書き込み転送は、aurora_replica_read_consistency を EVENTUAL、SESSION、または GLOBAL に設定した場合にのみ使用できます。このパラメータは、書き込み転送が有効な DB クラスターのリーダーインスタンスにのみ関係します。
- マルチステートメントトランザクション内で、このパラメータを設定したり (空の場合)、設定解除したり (既に設定されている場合) することはできません。そのようなトランザクション中に、ある有効な値から別の有効な値に変更できますが、このアクションはお勧めしません。

DB クラスターの書き込み転送が有効になっているかどうかの確認

DB クラスターで書き込み転送を使用できるかどうかを判断するには、クラスターの属性 `LocalWriteForwardingStatus` が `enabled` に設定されていることを確認します。

クラスターの詳細ページの [設定] タブの AWS Management Console に、[ローカルリードレプリカ書き込み転送] のステータスが [有効] と表示されます。

すべてのクラスターの書き込み転送設定のステータスを表示するには、次の AWS CLI コマンドを実行します。

Example

```
aws rds describe-db-clusters \
--query '*[].
{DBClusterIdentifier:DBClusterIdentifier,LocalWriteForwardingStatus:LocalWriteForwardingStatus}

[
    {
        "LocalWriteForwardingStatus": "enabled",
        "DBClusterIdentifier": "write-forwarding-test-cluster-1"
    },
    {
        "LocalWriteForwardingStatus": "disabled",
        "DBClusterIdentifier": "write-forwarding-test-cluster-2"
    },
    {
        "LocalWriteForwardingStatus": "requested",
        "DBClusterIdentifier": "test-global-cluster-2"
    },
    {
        "LocalWriteForwardingStatus": "null",
        "DBClusterIdentifier": "aurora-mysql-v2-cluster"
    }
]
```

DB クラスターは、`LocalWriteForwardingStatus` として以下の値を持ちます。

- `disabled` — 書き込み転送は無効です。
- `disabling` — 書き込み転送は無効化中です。
- `enabled` — 書き込み転送は有効です。
- `enabling` — 書き込み転送は有効化中です。

- null — この DB クラスターでは書き込み転送は使用できません。
- requested — 書き込み転送がリクエストされましたが、まだアクティブではありません。

書き込み転送とアプリケーションおよび SQL の互換性

書き込み転送では、次の種類の SQL ステートメントを使用できます。

- INSERT、DELETE、および UPDATE などのデータ操作言語 (DML) ステートメント。書き込み転送で使用できるこれらのステートメントのプロパティには、以下で説明するように、いくつかの制限があります。
- SELECT ... LOCK IN SHARE MODE と SELECT FOR UPDATE ステートメント。
- PREPARE と EXECUTE ステートメント。

書き込み転送機能を持つ DB クラスターでは、特定のステートメントの使用が許可されていないか、古い結果を生成する可能性があります。したがって、DB クラスターでは、EnableLocalWriteForwarding 設定はデフォルトで無効になっています。有効にする前に、アプリケーションコードがこれらの制限の影響を受けないことを確認してください。

書き込み転送で使用する SQL ステートメントには、次の制限が適用されます。場合によっては、書き込み転送が有効な DB クラスターでステートメントを使用できます。この方法は、aurora_replica_read_consistency 設定パラメータによってセッション内で書き込み転送が有効化されていない場合に機能します。書き込み転送のために許可されていないステートメントを使用しようとすると、次のようなエラーメッセージが表示されます。

```
ERROR 1235 (42000): This version of MySQL doesn't yet support 'operation with write
 forwarding'.
```

データ定義言語 (DDL)

ライター DB インスタンスに接続して DDL ステートメントを実行します。リーダー DB インスタンスからは実行できません。

テンポラリテーブルのデータを使用した永続テーブルの更新

書き込み転送が有効な DB クラスターでテンポラリテーブルを使用できます。ただし、ステートメントがテンポラリテーブルを参照している場合は、DML ステートメントを使用して永続テーブルを変更することはできません。例えば、テンポラリテーブルからデータを取る INSERT ... SELECT ステートメントを使用することはできません。

XA トランザクション

セッション内で書き込み転送が有効になっている場合、DB クラスターで次のステートメントを使用することはできません。これらのステートメントは、書き込み転送が有効になっていない DB クラスター、または aurora_replica_read_consistency 設定が空のセッションで使用できます。セッション内で書き込み転送を有効にする前に、コードでこれらのステートメントが使用されているかどうかを確認してください。

```
XA {START|BEGIN} xid [JOIN|RESUME]
XA END xid [SUSPEND [FOR MIGRATE]]
XA PREPARE xid
XA COMMIT xid [ONE PHASE]
XA ROLLBACK xid
XA RECOVER [CONVERT XID]
```

永続テーブルの LOAD ステートメント

書き込み転送が有効な DB クラスターでは、以下のステートメントを使用できません。

```
LOAD DATA INFILE 'data.txt' INTO TABLE t1;
LOAD XML LOCAL INFILE 'test.xml' INTO TABLE t1;
```

プラグインステートメント

書き込み転送が有効な DB クラスターでは、以下のステートメントを使用できません。

```
INSTALL PLUGIN example SONAME 'ha_example.so';
UNINSTALL PLUGIN example;
```

SAVEPOINT ステートメント

セッション内で書き込み転送が有効になっている場合、DB クラスターで次のステートメントを使用することはできません。これらのステートメントは、書き込み転送が有効になっていない DB クラスター、または aurora_replica_read_consistency 設定が空白のセッションで使用できます。セッション内で書き込み転送を有効にする前に、コードでこれらのステートメントが使用されているかどうかを確認してください。

```
SAVEPOINT t1_save;
ROLLBACK TO SAVEPOINT t1_save;
RELEASE SAVEPOINT t1_save;
```

書き込み転送の分離レベル

書き込み転送を使用するセッションでは、REPEATABLE READ 分離レベルのみを使用できます。Aurora レプリカで READ COMMITTED 分離レベルを使用することもできますが、その分離レベルは書き込み転送では機能しません。REPEATABLE READ および READ COMMITTED 分離レベルの詳細については、「Aurora MySQL の分離レベル」を参照してください。

書き込み転送の読み取り整合性

DB クラスターの読み取り整合性の程度を制御できます。読み取り整合性レベルによって、一部またはすべての変更がライターからレプリケートされるように、各読み取りオペレーションの前の DB クラスターの待機時間が決まります。読み取り整合性レベルを調整して、セッションから転送されたすべての書き込みオペレーションが、後続のクエリの前に DB クラスターに表示することができます。また、この設定を使用して、DB クラスターのクエリに、常にライターからの最新の更新が表示されるようにすることもできます。この設定は、他のセッションまたは他のクラスターによって送信されるクエリにも適用されます。アプリケーションでこの種類の動作を指定するには、aurora_replica_read_consistency DB パラメータまたは DB クラスターパラメータの値を選択します。

> **Important**
>
> 書き込みを転送するときには、必ず、aurora_replica_read_consistency DB パラメータまたは DB クラスターパラメータを設定してください。そうしないと、Aurora は書き込みを転送しません。デフォルトでは、このパラメータに空の値があるため、このパラメータを使用する場合は特定の値を選択してください。aurora_replica_read_consistency パラメータは、書き込み転送が有効な DB クラスターまたはインスタンスにのみ影響します。

整合性レベルを上げると、アプリケーションは、DB インスタンス間で変更が伝播されるのを待つ時間が長くなります。応答時間の短縮と、クエリを実行する前に他の DB インスタンスで行われた変更が完全に使用可能であることのバランスを選択できます。

aurora_replica_read_consistency パラメータには以下の値を指定できます。

- EVENTUAL — 同じセッションでの書き込み操作の結果は、ライター DB インスタンスで書き込み操作が実行されるまで表示されません。クエリは、更新された結果が使用可能になるのを待つことはありません。したがって、ステートメントのタイミングとレプリケーションの遅延の量によって

は、古いデータや更新されたデータが取得される可能性があります。これは、書き込み転送を使用しない Aurora MySQL DB クラスターの場合と同じ整合性です。

- SESSION — 書き込み転送を使用するすべてのクエリには、そのセッションで行われたすべての変更の結果が表示されます。トランザクションがコミットされているかどうかにかかわらず、変更が表示されます。必要に応じて、クエリは、転送された書き込み操作の結果がレプリケートされるまで待機します。
- GLOBAL — セッションには、DB クラスター内のすべてのセッションとインスタンスでコミットされたすべての変更が表示されます。各クエリは、セッション遅延の量に応じて変化する期間を待つことがあります。クエリは、クエリが開始された時点の、DB クラスターがライターからコミットされたすべてのデータを含む最新状態になったときに実行されます。

書き込み転送に関連する設定パラメータの詳細については、「書き込み転送の設定パラメータ」を参照してください。

> **Note**
>
> 跡えば次のように、aurora_replica_read_consistency をセッション変数として使用することもできます。
>
> ```
> mysql> set aurora_replica_read_consistency = 'session';
> ```

書き込み転送の使用例

この例は、INSERT ステートメントの後に SELECT ステートメントが実行される場合の aurora_replica_read_consistency パラメータの影響を示しています。aurora_replica_read_consistency の値とステートメントのタイミングによっては、結果が異なる場合があります。

一貫性を高めるには、SELECT ステートメントを発行する前にしばらくお待ちください。または Aurora は、結果のレプリケーションが完了するまで自動的に待機してから、SELECT 処理を続行することができます。

DB パラメータの設定の詳細については、「「パラメータグループを使用する」」を参照してください。

Example **aurora_replica_read_consistency** が **EVENTUAL** に設定された

INSERT ステートメントの直後に SELECT ステートメントを実行すると、新しい行が挿入される前の行数を示す COUNT(*) の値が返されます。しばらくしてから SELECT を再度実行すると、更新された行数が返されます。SELECT ステートメントは待機しません。

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)

mysql> insert into t1 values (6); select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.00 sec)
```

Example **aurora_replica_read_consistency** が **SESSION** に設定された

INSERT の直後の SELECT ステートメントは、INSERT ステートメントからの変更が反映されるまで待機します。後続の SELECT ステートメントは待機しません。

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.01 sec)
```

```
mysql> insert into t1 values (6); select count(*) from t1; select count(*) from t1;
Query OK, 1 row affected (0.08 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.37 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.00 sec)
```

読み取り整合性設定を SESSION に設定したまま、INSERT ステートメントの実行後に短い待機を行うと、次の SELECT ステートメントが実行されるまでに更新された行カウントが使用可能になります。

```
mysql> insert into t1 values (6); select sleep(2); select count(*) from t1;
Query OK, 1 row affected (0.07 sec)
+----------+
| sleep(2) |
+----------+
|        0 |
+----------+
1 row in set (2.01 sec)
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.00 sec)
```

Example **aurora_replica_read_consistency** が **GLOBAL** に設定された

各 SELECT ステートメントは、クエリを実行する前に、ステートメントの開始時刻の時点でのすべてのデータ変更が表示されるように待機します。各 SELECT ステートメントの待機時間は、レプリケーションラグの量によって異なります。

```
mysql> select count(*) from t1;
```

```
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.75 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.37 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.66 sec)
```

書き込み転送を使用したマルチパートステートメントの実行

DML ステートメントは、INSERT ... SELECT ステートメントや DELETE ... WHERE ステートメントなど、複数の部分から構成される場合があります。この場合、ステートメント全体がライター DB インスタンスに転送され、そこで実行されます。

書込み転送を使用したトランザクション

トランザクションアクセスモードが読み取り専用に設定されている場合、書き込み転送は使用されません。SET TRANSACTION ステートメントまたは START TRANSACTION ステートメントを使用して、トランザクションのアクセスモードを指定できます。transaction_read_only セッション変数の値を変更することで、トランザクションアクセスモードを指定することもできます。このセッション値は、書き込み転送が有効な DB クラスターに接続しているときにのみ変更できます。

長時間実行されるトランザクションがかなりの期間ステートメントを発行しない場合、アイドルタイムアウト期間を超える可能性があります。この期間のデフォルトは 1 分です。aurora_fwd_writer_idle_timeout パラメータを設定して、最大 1 日まで増やすことができます。アイドルタイムアウトを超えたトランザクションは、ライターインスタンスによって

キャンセルされます。次に送信するステートメントは、タイムアウトエラーを受け取ります。その後、Aurora はトランザクションをロールバックします。

このタイプのエラーは、書き込み転送が使用できなくなった場合に発生する可能性があります。例えば、DB クラスターを再起動した場合や、書き込み転送を無効にした場合、Aurora は書き込み転送を使用するすべてのトランザクションをキャンセルします。

ローカル書き込み転送を使用しているクラスター内のライターインスタンスが再起動されると、ローカル書き込み転送を使用しているリーダーインスタンス上のアクティブで転送されたトランザクションとクエリは自動的に閉じられます。ライターインスタンスが再び使用可能になったら、これらのトランザクションを再試行できます。

書き込み転送の設定パラメータ

Aurora DB パラメータグループには、書き込み転送機能の設定が含まれています。これらのパラメータの詳細を次の表にまとめ、表に続いて使用上の注意を記載してください。

パラメータ	スコープ	タイプ	デフォルト値	有効な値
aurora_fwd_writer_idle_timeout	クラスター	符号なし整数	60	1-86,400
aurora_fwd_writer_max_connections_pct	クラスター	符号なし長整数	10	0-90
aurora_replica_read_consistency	クラスターまたはインスタンス	列挙型	null	EVENTUAL, SESSION, GLOBAL

受信する書き込み要求を制御するには、以下の設定を使用してください。

- aurora_fwd_writer_idle_timeout — リーダーインスタンスから転送された接続で、ライター DB インスタンスがアクティビティを待ってから、接続を閉じるまでの秒数。この期間を超えてセッションがアイドル状態のままである場合、Aurora はセッションをキャンセルします。
- aurora_fwd_writer_max_connections_pct - リーダーから転送されたクエリを処理するためにライター DB インスタンスで使用できるデータベース接続の上限。これは、ライターの max_connections 設定のパーセンテージで表されます。例え

ば、max_connections が800 回、aurora_fwd_master_max_connections_pctまたはaurora_fwd_writer_max_connections_pctが 10 回 の場合、書き込みは最大 80 回 の同時転送セッションを許可します。これらの接続は、max_connections 設定によって管理される同じ接続プールから取得されます。

この設定は、書き込み転送が有効なライターにのみ適用されます。この値を小さくしても、既存の接続は影響を受けません。Aurora は、DB クラスターから新しい接続の作成を試みるときに、この新しい設定値を参照します。デフォルト値は 10 で、max_connections 値の 10% を表します。

> **Note**
>
> aurora_fwd_writer_idle_timeout と aurora_fwd_writer_max_connections_pct は DB クラスターのパラメータであるため、各クラスターのすべての DB インスタンスは、これらのパラメータに同じ値を持ちます。

aurora_replica_read_consistency の詳細については、「書き込み転送の読み取り整合性」を参照してください。

DB パラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。

書き込み転送のための Amazon CloudWatch メトリクスと Aurora MySQL ステータス変数

以下の Amazon CloudWatch メトリクスと Aurora MySQL ステータス変数は、1 つ以上の DB クラスターで書き込み転送を使用するときに適用されます。これらのメトリクスとステータス変数はすべて、ライター DB インスタンスで測定されます。

CloudWatch メトリクス	Aurora MySQL ステータス変数	単位	説明
ForwardingWriterDMLLatency	–	Milliseconds	書き込み DB インスタンスで転送された各 DML ステートメントを処理する平均時間。

CloudWatch メトリクス	Aurora MySQL ステータス変数	単位	説明
			DB クラスターが書き込みリクエストを転送する時間や、変更をライターに複製する時間は含まれません。
ForwardingWriterDMLThroughput	–	1 秒あたりのカウント数	この書き込み DB インスタンスによって 1 秒間に処理される、転送された DML ステートメントの数。
ForwardingWriterOpenSessions	Aurora_fwd_writer_open_sessions	Count (カウント)	書き込み DB インスタンスで転送されたセッションの数。
–	Aurora_fwd_writer_dml_stmt_count	Count (カウント)	この書き込み DB インスタンスに転送された DML ステートメントの合計数。
–	Aurora_fwd_writer_dml_stmt_duration	マイクロ秒	この書き込み DB インスタンスに転送された DML ステートメントの合計期間。
–	Aurora_fwd_writer_select_stmt_count	Count (カウント)	この書き込み DB インスタンスに転送された SELECT ステートメントの総数。

CloudWatch メトリクス	Aurora MySQL ステータス変数	単位	説明
–	`Aurora_fwd_writer_select_stmt_duration`	マイクロ秒	この書き込み DB インスタンスに転送された SELECT ステートメントの合計期間。

以下の CloudWatch メトリクスと Aurora MySQL ステータス変数は、書き込み転送が有効な DB クラスター内の各リーダー DB インスタンスで測定されます。

CloudWatch メトリクス	Aurora MySQL ステータス変数	単位	説明
`ForwardingReplicaDMLLatency`	–	ミリ秒	レプリカ上で転送された DML の平均応答時間。
`ForwardingReplicaDMLThroughput`	–	1 秒あたりのカウント数	1 秒あたりに処理された転送 DML ステートメントの数。
`ForwardingReplicaOpenSessions`	`Aurora_fwd_replica_open_sessions`	Count (カウント)	リーダー DB インスタンスで書き込み転送を使用しているセッションの数。
`ForwardingReplicaReadWaitLatency`	–	ミリ秒	リーダー DB インスタンス上の SELECT ステートメントがライターに追いつくのを待機する平均待機時間。 クエリを処理する前にリーダー DB インスタンスが待機する

CloudWatch メトリクス	Aurora MySQL ステータス変数	単位	説明
			程度は、aurora_replica_read_consistency 設定によって異なります。
ForwardingReplicaReadWaitThroughput	–	1 秒あたりのカウント数	書き込みを転送しているすべてのセッションで、1 秒間処理した SELECT ステートメントの総数。
ForwardingReplicaSelectLatency	–	ミリ秒	転送済み SELECT レイテンシー。モニタリング期間内に転送されたすべての SELECT ステートメントの平均値。
ForwardingReplicaSelectThroughput	–	1 秒あたりのカウント数	モニタリング期間内の 1 秒あたりの転送済み SELECT スループット。
–	Aurora_fwd_replica_dml_stmt_count	Count (カウント)	このリーダー DB インスタンスから転送された DML ステートメントの合計数。
–	Aurora_fwd_replica_dml_stmt_duration	マイクロ秒	このリーダー DB インスタンスから転送された DML ステートメントの合計期間。

CloudWatch メトリクス	Aurora MySQL ステータス変数	単位	説明
–	`Aurora_fwd_replica_errors_session_limit`	Count (カウント)	以下のエラー条件のいずれかが原因でプライマリクラスターが拒否したセッションの数。 • writer full • 転送中のステートメントが多すぎます。
–	`Aurora_fwd_replica_read_wait_count`	Count (カウント)	このリーダー DB インスタンスでの書き込み後の読み取り待機の合計数。
–	`Aurora_fwd_replica_read_wait_duration`	マイクロ秒	このリーダー DB インスタンスの読み取り整合性設定による待機時間の合計。
–	`Aurora_fwd_replica_select_stmt_count`	Count (カウント)	このリーダー DB インスタンスから転送された SELECT ステートメントの合計数。
–	`Aurora_fwd_replica_select_stmt_duration`	マイクロ秒	このリーダー DB インスタンスから転送された SELECT ステートメントの合計期間。

転送されたトランザクションとクエリの識別

information_schema.aurora_forwarding_processlist テーブルを使用して、転送されたトランザクションとクエリを識別できます。このテーブルの詳細については、「information_schema.aurora_forwarding_processlist」を参照してください。

次の例は、ライター DB インスタンス上のすべての転送された接続を示しています。

```
mysql> select * from information_schema.AURORA_FORWARDING_PROCESSLIST where
 IS_FORWARDED=1 order by REPLICA_SESSION_ID;

+-----+----------+--------------------+----------+---------+------+--------------
+---------------------------------------------+--------------+--------------------
+----------------------------------+----------------------+----------------+
| ID  | USER     | HOST               | DB       | COMMAND | TIME | STATE        |
 INFO                                      | IS_FORWARDED | REPLICA_SESSION_ID |
 REPLICA_INSTANCE_IDENTIFIER     | REPLICA_CLUSTER_NAME | REPLICA_REGION |
+-----+----------+--------------------+----------+---------+------+--------------
+---------------------------------------------+--------------+--------------------
+----------------------------------+----------------------+----------------+
| 648 | myuser   | IP_address:port1   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest58 SET k=k+1 WHERE id=4802579 |            1 |                637 | my-
db-cluster-instance-2        | my-db-cluster        | us-west-2      |
| 650 | myuser   | IP_address:port2   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest54 SET k=k+1 WHERE id=2503953 |            1 |                639 | my-
db-cluster-instance-2        | my-db-cluster        | us-west-2      |
+-----+----------+--------------------+----------+---------+------+--------------
+---------------------------------------------+--------------+--------------------
+----------------------------------+----------------------+----------------+
```

転送側のリーダー DB インスタンスで、SHOW PROCESSLIST を実行することにより、これらのライター DB 接続に関連するスレッドを確認できます。ライターの REPLICA_SESSION_ID の値、637 と 639 は、リーダーの Id 値と同じです。

```
mysql> select @@aurora_server_id;

+--------------------------------+
| @@aurora_server_id             |
+--------------------------------+
| my-db-cluster-instance-2       |
+--------------------------------+
1 row in set (0.00 sec)
```

```
mysql> show processlist;

+-----+----------+--------------------+----------+---------+------+--------------
+-------------------------------------------+
| Id  | User     | Host               | db       | Command | Time | State        | Info
                                        |
+-----+----------+--------------------+----------+---------+------+--------------
+-------------------------------------------+
| 637 | myuser   | IP_address:port1   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest12 SET k=k+1 WHERE id=4802579  |
| 639 | myuser   | IP_address:port2   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest61 SET k=k+1 WHERE id=2503953  |
+-----+----------+--------------------+----------+---------+------+--------------
+-------------------------------------------+
12 rows in set (0.00 sec)
```

AWS リージョン 間での Amazon Aurora MySQL DB クラスターのレプリケーション

Amazon Aurora MySQL DB クラスターを、出典 DB クラスターとは異なる AWS リージョン に、リードレプリカとして作成できます。このアプローチを使用すると、災害対策機能が向上し、ユーザーに近い AWS リージョン への読み取りオペレーションをスケールして、AWS リージョン 間の移行を容易にすることができます。

暗号化されている DB クラスターと暗号化されていない DB クラスターの両方のリードレプリカを作成できます。出典 DB クラスターが暗号化されている場合、リードレプリカを暗号化する必要があります。

出典 DB クラスターごとに、リードレプリカとすることができるクロスリージョン DB クラスターは最大 5 つです。

> **Note**
>
> クロスリージョンリードレプリカの代わりに、 Aurora Global Database を使用して、最小限のラグタイムで読み取り操作をスケールできます。Aurora Global Database では、1 つの AWS リージョン にプライマリ Aurora DB クラスターがあり、異なるリージョンに最大 5 つの読み取り専用セカンダリ DB クラスターがあります。各セカンダリ DB クラスターには、最大 16 個の (15 ではなく) Aurora レプリカを含めることができます。プライマリ DB クラスターからすべてのセカンダリへのレプリケーションは、データベースエンジンではなく Aurora ストレージレイヤーによって処理されるため、変更をレプリケートする際のラグタイムは非常に短く、通常は 1 秒未満となります。データベースエンジンをレプリケーションプロセスから除外するということは、データベースエンジンがワークロードの処理のみを実行することを意味します。また、Aurora MySQL の binlog (バイナリログ) のレプリケーションを設定または管理する必要もありません。詳細については、Amazon Aurora Global Database の使用 を参照してください。

別の AWS リージョン に Aurora MySQL DB クラスターのリードレプリカを作成する場合は、以下の点に注意してください。

- 出典 DB クラスターとクロスリージョンリードレプリカ DB クラスターのどちらにも、最大 15 個の Aurora レプリカを DB クラスターのプライマリインスタンスと同時に作成できます。この機能により、出典 AWS リージョン とレプリケーションターゲットの AWS リージョン の両方で、読み取りオペレーションをスケールできるようになります。

- クロスリージョンシナリオでは、AWS リージョン 間のネットワークチャネルが長くなるため、出典 DB クラスターとリードレプリカ間のラグタイムが長くなります。
- クロスリージョンレプリケーションから転送されたデータには、Amazon RDS のデータ転送料金が発生します。以下のクロスリージョンレプリケーションアクションでは、出典 AWS リージョンから転送されるデータに対して料金が発生します。
  - リードレプリカを作成すると、Amazon RDS によって出典クラスターのスナップショットが作成され、リードレプリカを保持する AWS リージョン に、そのスナップショットが転送されます。
  - 出典データベースのデータが変更されるたびに、Amazon RDS によって、出典リージョンからリードレプリカを保持する AWS リージョン にデータが転送されます。

  Amazon RDS データ転送料金の詳細については、「Amazon Aurora の料金」を参照してください。
- 同じ出典 DB クラスターを参照するリードレプリカに対して、複数の同時作成または削除アクションを実行できます。ただし、出典 DB クラスターごとに作成できるリードレプリカは 5 つまでです。
- レプリケーションを効率的に実行するには、各リードレプリカに、出典 DB クラスターと同程度のコンピューティングおよびストレージリソースが必要です。出典 DB クラスターを拡張した場合、リードレプリカも拡張する必要があります。

トピック

スタートする前に

クロスリージョンリードレプリカとなる Aurora MySQL DB クラスターを作成する前に、出典 Aurora MySQL DB クラスターでバイナリログを有効にする必要があります。Aurora MySQL のクロスリージョンレプリケーションは、MySQL バイナリレプリケーションを使用してクロスリージョンリードレプリカ DB クラスターでの変更を再生します。

Aurora MySQL DB クラスターでバイナリログ記録を有効にするには、出典 DB クラスターの binlog_format パラメータを更新します。binlog_format パラメータはクラスターレベルのパラメータであり、デフォルトクラスターパラメータグループにあります。DB クラスターがデフォルトクラスター DB パラメータグループを使用している場合、新しい DB クラスターパラメータグループを作成して binlog_format 設定を変更します。binlog_format を MIXED に設定することをお勧めします。ただし、特定バイナリログ形式が必要な場合は binlog_format を ROW または STATEMENT に設定する必要もあります。変更を適用するには、Aurora DB クラスターを再起動します。

Aurora MySQL でバイナリログ記録を使用する方法の詳細については、「Aurora と MySQL との間、または Aurora と別の Aurora DB クラスターとの間のレプリケーション (バイナリログレプリケーション)」を参照してください。Aurora MySQL 設定パラメータの変更の詳細については、「Amazon Aurora の DB クラスターパラメータと DB インスタンスパラメータ」および「「パラメータグループを使用する」」を参照してください。

クロスリージョンリードレプリカとなる Amazon Aurora MySQL DB クラスターの作成

AWS Management Console、AWS Command Line Interface (AWS CLI)、または Amazon RDS API を使用して、クロスリージョンリードレプリカである Aurora DB クラスターを作成できます。暗号化されている DB クラスターと暗号化されていない DB クラスターの両方からクロスリージョンリードレプリカを作成できます。

AWS Management Console を使用して Aurora MySQL のクロスリージョンリードレプリカを作成すると、Amazon RDS はターゲットの AWS リージョン に DB クラスターを作成した後、その DB クラスターのプライマリインスタンスとなる DB インスタンスを自動的に作成します。

AWS CLI または RDS API を使用してクロスリージョンリードレプリカを作成する場合は、まずターゲット AWS リージョン で DB クラスターを作成し、アクティブになるまで待ちます。アクティブになったら、その DB クラスターのプライマリインスタンスとなる DB インスタンスを作成します。

レプリケーションは、リードレプリカ DB クラスターのプライマリインスタンスが使用可能になるとスタートされます。

Aurora MySQL DB クラスターからクロスリージョンリードレプリカを作成するには、以下の手順に従います。これらの手順は、暗号化されている DB クラスターまたは暗号化されていない DB クラスターからリードレプリカを作成するために使用できます。

コンソール

AWS Management Console を使用してクロスリージョンリードレプリカとなる Aurora MySQL DB クラスターを作成するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. AWS Management Console の右上隅で、出典 DB クラスターをホストする AWS リージョン を選択します。
3. ナビゲーションペインで [データベース] を選択します。
4. クロスリージョンリードレプリカを作成する DB クラスターを選択します。
5. [Actions] (アクション) として、[Create cross-Region read replica] (クロスリージョンリードレプリカの作成) を選択します。
6. [クロスリージョンのリードレプリカの作成] ページで、以下の表に示すように、クロスリージョンリードレプリカ DB クラスターのオプション設定を選択します。

オプション	説明
送信先リージョン	新しいクロスリージョンリードレプリカ DB クラスターをホストする AWS リージョン を選択します。
送信先 DB サブネットグループ	クロスリージョンリードレプリカ DB クラスターで使用する DB サブネットグループを選択します。
パブリックアクセス可能	クロスリージョンリードレプリカ DB クラスターにパブリック IP アドレスを付与する場合は、[Yes (はい)] を選択します。それ以外の場合は、[No (いいえ)] を選択します。
暗号化	この DB クラスターの保存時に暗号化をオンにするには、「暗号化を有効にする」を選択します。詳細については、「Amazon Aurora リソースの暗号化」を参照してください。
AWS KMS key	[暗号化] が [暗号を有効化] に設定されている場合にのみ使用できます。この DB クラスターの暗号化に使用する AWS KMS key を選択します。詳細については、

オプション	説明
	「Amazon Aurora リソースの暗号化」を参照してください。
DB インスタンスクラス	DB クラスターのプライマリインスタンスに対する処理要件やメモリ要件を定義する DB インスタンスクラスを選択します。DB インスタンスクラスのオプションについては、「Aurora DB インスタンスクラス」を参照してください。
マルチ AZ 配置	[Yes] (はい) を選択して、フェイルオーバーをサポートするためにターゲット AWS リージョン 内の別のアベイラビリティーゾーンに新しい DB クラスターのリードレプリカを作成します。複数のアベイラビリティーゾーンの詳細については、「リージョンとアベイラビリティーゾーン」を参照してください。
リードレプリカのソース	クロスリージョンリードレプリカを作成する出典 DB クラスターを選択します。

オプション	説明
DB インスタンス識別子	クロスリージョンリードレプリカ DB クラスターのプライマリインスタンス名を入力します。この識別子は、新しい DB クラスターのプライマリインスタンスのエンドポイントアドレスで使用されます。 DB インスタンス識別子には次の制約があります。 • 1 ~ 63 文字の英数字またはハイフンを使用する必要があります。 • 1 字目は文字である必要があります。 • ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません。 • これは、AWS リージョン ごと、AWS アカウントごとにすべての DB インスタンスに対して一意にする必要があります。 クロスリージョンリードレプリカ DB クラスターは出典 DB クラスターのスナップショットから作成されるため、リードレプリカのマスターユーザー名およびマスターパスワードは出典 DB クラスターのマスターユーザー名およびマスターパスワードと同じになります。

オプション	説明
DB クラスター識別子	レプリカのターゲット AWS リージョン にあるアカウントに対して一意であるクロスリージョンリードレプリカ DB クラスターの名前を入力します。この識別子は、DB クラスターのクラスターエンドポイントアドレスで使用されます。クラスターエンドポイントの詳細については、「Amazon Aurora 接続管理」を参照してください。 DB クラスター識別子には以下の制約があります。 • 1 ~ 63 文字の英数字またはハイフンを使用する必要があります。 • 1 字目は文字である必要があります。 • ハイフンを、文字列の最後に使用したり、2 つ続けて使用したりすることはできません。 • これは AWS アカウント ごとの AWS リージョン ごとにすべての DB クラスターに対して一意にする必要があります。
優先度	新しい DB クラスターのプライマリインスタンスのフェイルオーバー優先度を選択します。この優先度により、プライマリインスタンスの障害からの復旧時に、Aurora レプリカを昇格する順序が決まります。値を選択しない場合、デフォルト値は tier-1 になります。詳細については、「Aurora DB クラスターの耐障害性」を参照してください。
データベースポート	データベースのアクセスに使用するために、アプリケーションやユーティリティのポートを指定します。Aurora DB クラスターのデフォルトの MySQL ポートは 3306 になります。会社のファイアウォールでは、このポートへの接続がブロックされます。会社のファイアウォールがデフォルトのポートをブロックする場合は、新しい DB クラスター用に別のポートを選択します。

オプション	説明
拡張モニタリング	「拡張モニタリングを有効にする」を選択して、DB クラスターが実行されているオペレーティングシステムのメトリクスの収集をリアルタイムでオンにします。詳細については、「拡張モニタリングを使用した OS メトリクスのモニタリング」を参照してください。
モニタリングロール	[拡張モニタリング] が [拡張モニタリングの有効化] に設定されている場合にのみ使用できます。Amazon CloudWatch Logs との通信を Amazon RDS に許可するために作成した IAM ロールを選択するか、[デフォルト] を選択して、RDS によって `rds-monitoring-role` という名前のロールが作成されるようにします。詳細については、「拡張モニタリングを使用した OS メトリクスのモニタリング」を参照してください。
詳細度	[拡張モニタリング] が [拡張モニタリングの有効化] に設定されている場合にのみ使用できます。DB クラスターのメトリクスを収集する間隔を秒単位で設定します。
マイナーバージョン自動アップグレード	この設定は Aurora MySQL DB クラスターには適用されません。 Aurora MySQL のエンジンに関する更新の詳細については、「Amazon Aurora MySQL のデータベースエンジンの更新」を参照してください。

7. [作成] を選択して、Aurora のクロスリージョンリードレプリカを作成します。

AWS CLI

CLI でクロスリージョンリードレプリカとなる Aurora MySQL DB クラスターを作成するには

1. リードレプリカ DB クラスターを作成する AWS リージョン で、AWS CLI create-db-cluster コマンドを呼び出します。--replication-source-identifier オプションを含め、リードレプリカを作成する出典 DB クラスターの Amazon リソースネーム (ARN) を指定します。

   --replication-source-identifier により識別される DB クラスターが暗号化されているクロスリージョンレプリケーションの場合、--kms-key-id オプションと --storage-encrypted オプションを指定します。

   > **Note**
   >
   > --storage-encrypted を指定して、--kms-key-id の値を指定することにより、暗号化されていない DB クラスターから暗号化されているリードレプリカへのクロスリージョンレプリケーションをセットアップできます。

   --master-username パラメータ --master-user-password とパラメータは指定できません。これらの値は、出典 DB クラスターから取得されます。

   次のコード例では、us-west-2 リージョンにある暗号化されていない DB クラスタースナップショットから us-east-1 リージョンにリードレプリカが作成されます。このコマンドは、us-east-1 リージョンで呼び出されます。この例では、マスターユーザーパスワードを生成して Secrets Manager で管理する --manage-master-user-password オプションを指定しています。詳細については、「Amazon Aurora および AWS Secrets Manager によるパスワード管理」を参照してください。または、--master-password オプションを使用して、自分でパスワードを指定して管理することもできます。

   Linux、macOS、Unix の場合:

   ```
   aws rds create-db-cluster \
     --db-cluster-identifier sample-replica-cluster \
     --engine aurora \
     --replication-source-identifier arn:aws:rds:us-
   west-2:123456789012:cluster:sample-master-cluster
   ```

   Windows の場合:

```
aws rds create-db-cluster ^
  --db-cluster-identifier sample-replica-cluster ^
  --engine aurora ^
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster
```

次のコード例では、us-west-2 リージョンにある暗号化されている DB クラススナップショットから us-east-1 リージョンにリードレプリカが作成されます。このコマンドは、us-east-1 リージョンで呼び出されます。

Linux、macOS、Unix の場合:

```
aws rds create-db-cluster \
  --db-cluster-identifier sample-replica-cluster \
  --engine aurora \
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster \
  --kms-key-id my-us-east-1-key \
  --storage-encrypted
```

Windows の場合:

```
aws rds create-db-cluster ^
  --db-cluster-identifier sample-replica-cluster ^
  --engine aurora ^
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster ^
  --kms-key-id my-us-east-1-key ^
  --storage-encrypted
```

`--source-region` GovCloud (米国東部) リージョンと AWS GovCloud (米国西部) リージョン間のクロスリージョンレプリケーションには、AWS で識別される DB クラスターが暗号化されている場合、`--replication-source-identifier` オプションが必要です。`--source-region` には、ソース DB クラスターの AWS リージョン を指定します。

`--source-region` を指定しない場合、`--pre-signed-url` の値を指定します。署名付きの URL は、ソースの AWS リージョン で呼び出される `create-db-cluster` コマンドに対する、署名バージョン 4 で署名されたリクエストを含む URL です。`pre-signed-url` オプショ

ンの詳細については、AWS CLI コマンドリファレンスの「create-db-cluster」を参照してください。

2. 以下の例に示すように、AWS CLI の describe-db-clusters コマンドを使用して、DB クラスターが使用できる状態であることを確認します。

```
aws rds describe-db-clusters --db-cluster-identifier sample-replica-cluster
```

describe-db-clusters の結果にステータス available と表示されたら、レプリケーションをスタートできるように DB クラスターのプライマリインスタンスを作成します。そのためには、以下の例に示すように AWS CLI create-db-instance コマンドを使用します。

Linux、macOS、Unix の場合:

```
aws rds create-db-instance \
  --db-cluster-identifier sample-replica-cluster \
  --db-instance-class db.r3.large \
  --db-instance-identifier sample-replica-instance \
  --engine aurora
```

Windows の場合:

```
aws rds create-db-instance ^
  --db-cluster-identifier sample-replica-cluster ^
  --db-instance-class db.r3.large ^
  --db-instance-identifier sample-replica-instance ^
  --engine aurora
```

DB インスタンスが作成されて使用可能になると、レプリケーションが始まります。DB インスタンスが使用可能かどうかを確認するには、AWS CLI の describe-db-instances コマンドを呼び出します。

RDS API

API を使用して、クロスリージョンリードレプリカとなる Aurora MySQL DB クラスターを作成するには

1. リードレプリカ DB クラスターを作成する AWS リージョン で、RDS API CreateDBCluster オペレーションを呼び出します。ReplicationSourceIdentifier パラメータを含め、リードレプリカを作成する出典 DB クラスターの Amazon リソースネーム (ARN) を指定します。

   ReplicationSourceIdentifier により識別される DB クラスターが暗号化されているクロスリージョンレプリケーションの場合、KmsKeyId パラメータを指定して、StorageEncrypted パラメータを true に設定します。

   > **Note**
   >
   > StorageEncrypted を **true** と指定し、KmsKeyId の値を指定することにより、暗号化されていない DB クラスターから暗号化されているリードレプリカへのクロスリージョンレプリケーションをセットアップできます。この場合、PreSignedUrl を指定する必要はありません。

   MasterUsername パラメータと MasterUserPassword パラメータは、出典 DB クラスターから取得されるため、これらの値を追加する必要はありません。

   次のコード例では、us-west-2 リージョンにある暗号化されていない DB クラスタースナップショットから us-east-1 リージョンにリードレプリカが作成されます。このアクションは、us-east-1 リージョンで呼び出されます。

```
https://rds.us-east-1.amazonaws.com/
  ?Action=CreateDBCluster
  &ReplicationSourceIdentifier=arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster
  &DBClusterIdentifier=sample-replica-cluster
  &Engine=aurora
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T001547Z
```

```
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=a04c831a0b54b5e4cd236a90dcb9f5fab7185eb3b72b5ebe9a70a4e95790c8b7
```

次のコード例では、us-west-2 リージョンにある暗号化されている DB クラススナップショットから us-east-1 リージョンにリードレプリカが作成されます。このアクションは、us-east-1 リージョンで呼び出されます。

```
https://rds.us-east-1.amazonaws.com/
  ?Action=CreateDBCluster
  &KmsKeyId=my-us-east-1-key
  &StorageEncrypted=true
  &PreSignedUrl=https%253A%252F%252Frds.us-west-2.amazonaws.com%252F
         %253FAction%253DCreateDBCluster
         %2526DestinationRegion%253Dus-east-1
         %2526KmsKeyId%253Dmy-us-east-1-key
         %2526ReplicationSourceIdentifier%253Darn%25253Aaws%25253Ards%25253Aus-
west-2%25253A123456789012%25253Acluster%25253Asample-master-cluster
         %2526SignatureMethod%253DHmacSHA256
         %2526SignatureVersion%253D4
         %2526Version%253D2014-10-31
         %2526X-Amz-Algorithm%253DAWS4-HMAC-SHA256
         %2526X-Amz-Credential%253DAKIADQKE4SARGYLE%252F20161117%252Fus-
west-2%252Frds%252Faws4_request
         %2526X-Amz-Date%253D20161117T215409Z
         %2526X-Amz-Expires%253D3600
         %2526X-Amz-SignedHeaders%253Dcontent-type%253Bhost%253Buser-agent%253Bx-
amz-content-sha256%253Bx-amz-date
         %2526X-Amz-Signature
%253D255a0f17b4e717d3b67fad163c3ec26573b882c03a65523522cf890a67fca613
  &ReplicationSourceIdentifier=arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster
  &DBClusterIdentifier=sample-replica-cluster
  &Engine=aurora
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T001547Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=a04c831a0b54b5e4cd236a90dcb9f5fab7185eb3b72b5ebe9a70a4e95790c8b7
```

AWS GovCloud (米国東部) リージョンと AWS GovCloud (米国西部) リージョン間のクロスリージョンレプリケーションで、ReplicationSourceIdentifier で識別される DB クラスターが暗号化されている場合、PreSignedUrl パラメータも指定します。署名付き URL は、レプリケートする暗号化された DB クラスターを含む出典 AWS リージョン で実行可能な CreateDBCluster API オペレーションの有効なリクエストである必要があります。KMS キー識別子は、リードレプリカを暗号化するために使用され、送信先 AWS リージョン で有効な KMS キーである必要があります。署名付き URL を手動ではなく自動的に生成するには、--source-region オプションを使用して AWS CLI の create-db-cluster コマンドを使用します。

2. 次の例に示すように、RDS API DescribeDBClusters オペレーションを使用して、DB クラスターが使用可能になっていることを確認します。

```
https://rds.us-east-1.amazonaws.com/
  ?Action=DescribeDBClusters
  &DBClusterIdentifier=sample-replica-cluster
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T002223Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=84c2e4f8fba7c577ac5d820711e34c6e45ffcd35be8a6b7c50f329a74f35f426
```

DescribeDBClusters の結果にステータス available と表示されたら、レプリケーションをスタートできるように DB クラスターのプライマリインスタンスを作成します。そのためには、次の例に示すように RDS API CreateDBInstance アクションを使用します。

```
https://rds.us-east-1.amazonaws.com/
  ?Action=CreateDBInstance
  &DBClusterIdentifier=sample-replica-cluster
  &DBInstanceClass=db.r3.large
  &DBInstanceIdentifier=sample-replica-instance
  &Engine=aurora
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T003808Z
```

```
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=125fe575959f5bbcebd53f2365f907179757a08b5d7a16a378dfa59387f58cdb
```

DB インスタンスが作成されて使用可能になると、レプリケーションが始まります。DB インスタンスが使用可能かどうかを確認するには、AWS CLI の DescribeDBInstances コマンドを呼び出します。

Amazon Aurora MySQL クロスリージョンレプリカの表示

Amazon Aurora MySQL DB クラスターのクロスリージョンレプリケーションの関係を表示するには、describe-db-clusters AWS CLI コマンドまたは DescribeDBClusters RDS API オペレーションを呼び出します。レスポンスでは、`ReadReplicaIdentifiers` フィールドを参照して、クロスリージョンリードレプリカ DB クラスターの識別子を確認します。レプリケーションソースであるソース DB クラスターの ARN の `ReplicationSourceIdentifier` 要素を参照してください。

リードレプリカを DB クラスターに昇格させる

Aurora MySQL リードレプリカをスタンドアロンの DB クラスターに昇格できます。Aurora MySQL リードレプリカを昇格させると、DB インスタンスの再起動後に利用可能になります。

通常、出典 DB クラスターに障害が発生した場合のデータリカバリースキームとして、Aurora MySQL リードレプリカをスタンドアロン DB クラスターに昇格させます。

これを行うには、初期にリードレプリカを作成し、次に出典 DB クラスターで障害をモニタリングします。障害が発生した場合、以下の作業を行います。

1. リードレプリカを昇格させます。
2. 昇格された DB クラスターにデータベーストラフィックを向けます。
3. 昇格された DB クラスターを出典として使用して置き換え用のリードレプリカを作成します。

リードレプリカを昇格させると、リードレプリカはスタンドアロン Aurora DB クラスターになります。リードレプリカのサイズによっては、昇格プロセスが完了するまで数分以上かかる場合があります。リードレプリカを新しい DB クラスターに昇格させると、他の DB クラスターと同等になります。例えば、そのリードレプリカを作成して、ポイントインタイム復元オペレーションを実行できます。また、その DB クラスターの Aurora レプリカを作成することもできます。

昇格された DB クラスターはリードレプリカではなくなったため、レプリケーションターゲットとしては使用できません。

以下のステップは、DB クラスターにリードレプリカを昇格させる一般的なプロセスを示しています。

1. リードレプリカ出典 DB クラスターへのトランザクションの書き込みを停止し、すべての更新がリードレプリカに加えられるまで待ちます。データベース更新は、出典 DB クラスターで行われた後にリードレプリカで行われるため、このレプリケーションラグは大きく変動する場合があります。`ReplicaLag` メトリクスを使用して、リードレプリカにすべての更新がいつ加えられたかを確認できます。`ReplicaLag` メトリクスは、出典 DB インスタンスからのリードレプリカ DB インスタンスのラグの時間を記録します。`ReplicaLag` メトリクスが `0` に達すると、リードレプリカが出典 DB インスタンスに追いついています。
2. Amazon RDS コンソールの [Promote] (昇格) オプション、AWS CLI コマンド promote-read-replica-db-cluster、または PromoteReadReplicaDBCluster Amazon RDS API オペレーションを使用して、リードレプリカを昇格させます。

   Aurora MySQL DB インスタンスを選択してリードレプリカを昇格させます。リードレプリカが昇格されると、Aurora MySQL DB クラスターがスタンドアロン DB クラスターに昇格します。フェイルオーバー優先度が最も高い DB インスタンスが DB クラスターのプライマリ DB インスタンスに昇格されます。他の DB インスタンスは Aurora レプリカになります。

   > **Note**
   >
   > 昇格プロセスの完了までには数分かかります。リードレプリカを昇格させると、レプリケーションが停止され、DB インスタンスが再起動されます。再起動が完了すると、リードレプリカは新しい DB クラスターとして使用可能になります。

コンソール

Aurora MySQL リードレプリカを DB クラスターに昇格するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. コンソールで、[Instances (インスタンス)] を選択します。

   [Instance] ペインが表示されます。
3. [Instances (インスタンス)] ペインで、昇格させるリードレプリカを選択します。

   リードレプリカは、Aurora MySQL DB インスタンスとして表示されます。

4. [アクション] で [リードレプリカの昇格] を選択します。
5. 確認ページで、[リードレプリカの昇格] を選択します。

AWS CLI

リードレプリカを DB クラスターに昇格させるには、AWS CLI promote-read-replica-db-cluster コマンドを使用します。

Example

Linux、macOS、Unix の場合:

```
aws rds promote-read-replica-db-cluster \
    --db-cluster-identifier mydbcluster
```

Windows の場合:

```
aws rds promote-read-replica-db-cluster ^
    --db-cluster-identifier mydbcluster
```

RDS API

リードレプリカを DB クラスターに昇格させるには、PromoteReadReplicaDBCluster を呼び出します。

Amazon Aurora MySQL クロスリージョンレプリカのトラブルシューティング

Amazon Aurora クロスリージョンリードレプリカを作成するときに発生する可能性がある一般的なエラーメッセージの一覧と、示されたエラーの解決策を次に示します。

出典クラスター [DB クラスター ARN] でバイナリログが有効になっていません

この問題を解決するには、出典 DB クラスターでバイナリログ記録を有効にします。詳細については、「スタートする前に」を参照してください。

出典クラスター [DB クラスター ARN] に、書き込みで同期されているクラスターパラメータグループがありません

このエラーは、binlog_format DB クラスターパラメータを更新しても、DB クラスターのプライマリインスタンスを再起動しなかった場合に発生します。DB クラスターのプライマリインスタンス (つまり、書き込み) を再起動し、再試行します。

出典クラスター [DB クラスター ARN] は、既にこのリージョンにリードレプリカを持っています

任意の AWS リージョン の出典 DB クラスターごとに、リードレプリカとすることができるクロスリージョン DB クラスターを最大 5 つまで用意できます。特定の AWS リージョン の DB クラスターに最大数のリードレプリカが既に存在する場合、そのリージョンで新しいクロスリージョン DB クラスターを作成する前に、既存のリードレプリカのいずれかを削除する必要があります。

DB クラスター [DB クラスター ARN] は、クロスリージョンレプリケーションをサポートするために、データベースエンジンのアップグレードを必要とする

この問題を解決するには、出典 DB クラスターのすべてのインスタンスのデータベースエンジンバージョンを最新のデータベースエンジンバージョンにアップグレードした後、クロスリージョンリードレプリカ DB を再作成してください。

Aurora と MySQL との間、または Aurora と別の Aurora DB クラスターとの間のレプリケーション (バイナリログレプリケーション)

Amazon Aurora MySQL は MySQL と互換性があるため、MySQL データベースと Amazon Aurora MySQL DB クラスターとの間のレプリケーションを設定できます。このタイプのレプリケーションでは、MySQL バイナリログレプリケーションが使用され、バイナリログレプリケーションとも呼ばれます。Aurora でバイナリログレプリケーションを使用する場合は、MySQL データベースで MySQL バージョン 5.5 以降を実行することをお勧めします。Aurora MySQL DB クラスターがレプリケーション出典またはレプリカである場合は、レプリケーションを設定できます。Amazon RDS MySQL DB インスタンス、Amazon RDS の外部の MySQL データベース、または別の Aurora MySQL DB クラスターを使用してレプリケートできます。

> **Note**
>
> 特定のタイプの Aurora クラスター間では、バイナリログレプリケーションを使用できません。特に、バイナリログレプリケーションは、Aurora Serverless v1 クラスターでは使用できません。`SHOW MASTER STATUS` と `SHOW SLAVE STATUS` (Aurora MySQL バージョン 2)、 または `SHOW REPLICA STATUS` (Aurora MySQL バージョン 3) ステートメントが出力を返さない場合は、使用しているクラスターがバイナリログレプリケーションをサポートしていることを確認してください。
>
> Aurora MySQL バージョン 3 では、バイナリログレプリケーションは、`mysql` システムデータベースに複製されません。Aurora MySQL バージョン 3 のバイナリログレプリケーションでは、パスワードとアカウントが複製されません。そのため、`CREATE USER`、`GRANT`、`REVOKE` などのデータ制御言語 (DCL) ステートメントは複製されません。

また、別の AWS リージョン で、RDS for MySQL DB インスタンスまたは Aurora MySQL DB クラスターに、レプリケートすることもできます。AWS リージョン 間のレプリケーションを実行する場合は、DB クラスターと DB インスタンスがパブリックにアクセス可能であることを確認してください。Aurora MySQL DB クラスターが VPC のプライベートサブネットにある場合は、AWS リージョン 間で VPC ピアリングを使用します。詳細については、「VPC 内の DB クラスターに別の VPC 内の EC2 インスタンスからアクセスする」を参照してください。

Aurora MySQL DB クラスターと別の AWS リージョン の Aurora MySQL DB クラスター間のレプリケーションを設定する場合、Aurora MySQL DB クラスターをソース DB クラスターとは別の AWS リージョン のリードレプリカとして作成できます。詳細については、「AWS リージョン 間での Amazon Aurora MySQL DB クラスターのレプリケーション」を参照してください。

Aurora MySQL 2 および 3 では、レプリケーションにグローバルトランザクション識別子 (GTID) を使用する外部のソースまたはターゲットと Aurora MySQL との間でレプリケートできます。Aurora MySQL DB クラスターの GTID 関連のパラメータ内に、外部データベースの GTID ステータスと互換性がある設定が含まれていることを確認してください。これを行う方法については、「GTID ベースレプリケーションを使用する」を参照してください。Aurora MySQL バージョン 3.01 以降では、GTID を使用しない出典からレプリケートされたトランザクションに GTID を割り当てる方法を選択できます。その設定を制御するストアドプロシージャの詳細については、mysql.rds_assign_gtids_to_anonymous_transactions (Aurora MySQL バージョン 3) を参照してください。

> **Warning**
>
> Aurora MySQL と MySQL との間で複製する場合は、必ず InnoDB テーブルのみを使用してください。MyISAM テーブルをレプリケートする場合は、これらのテーブルを以下のコマンドを使用して InnoDB に変換してから、レプリケーションを設定できます。
>
> ```
> alter table <schema>.<table_name> engine=innodb, algorithm=copy;
> ```

MySQL または別の Aurora DB クラスターとのレプリケーションの設定

Aurora MySQL と MySQL との間でレプリケーションを設定するには、次のステップを使用します。各ステップについては、このトピックで詳しく説明します。

1. レプリケーション出典のバイナリログ記録を有効にする
2. レプリケーション出典のバイナリログを不要になるまで保持する

3. レプリケーションソースのスナップショットまたはダンプを作成する

4. レプリカターゲットにスナップショットまたはダンプをロードする

5. レプリケーションソースでレプリケーションユーザーを作成する

6. レプリカターゲットでレプリケーションを有効にする

7. レプリカをモニタリングする

1. レプリケーション出典のバイナリログ記録を有効にする

以下のデータベースエンジンのレプリケーション出典で、バイナリログを有効にする手順を確認します。

データベースエンジン	手順
Aurora MySQL	Aurora MySQL DB クラスターのバイナリログ記録を有効にするには `binlog_format` DB クラスターパラメータを `ROW`、`STATEMENT` 、または `MIXED` に設定します。特定のバイナリログ形式の必要性がない限り、`MIXED` をお勧めします。(デフォルト値は`OFF`です。) `binlog_format` パラメータを変更するには、カスタム DB クラスターパラメータグループを作成し、そのカスタムパラメータグループを DB クラスターに関連付けます。デフォルト DB クラスターパラメータグループのパラメータは変更できません。 `binlog_format` パラメータを `OFF` から別の値に変更した場合、変更を有効にするには、Aurora DB クラスターを再起動する必要があります。 詳細については、Amazon Aurora の DB クラスターパラメータと DB インスタンスパラメータおよび「パラメータグループを使用する」 を参照してください。
RDS for MySQL	Amazon RDS DB インスタンスのバイナリログ記録を有効にするには Amazon RDS DB インスタンスでは、バイナリログ記録を直接有効にすることはできませんが、以下のいずれかの操作により有効にすることができます。 • DB インスタンスの自動バックアップを有効にする。自動バックアップの有効化は、DB インスタンス作成時、または既存の DB インスタンスを変更することで可

データベースエンジン	手順
	能です。詳細については、Amazon RDS ユーザーガイドの「DB インスタンスの作成」を参照してください。 • DB インスタンスのリードレプリカを作成します。詳細については、Amazon RDS ユーザーガイドの「リードレプリカの使用」を参照してください。

データベースエンジン	手順
MySQL (外部)	暗号化レプリケーションを設定するには Aurora MySQL バージョン 2 を使用してデータを安全に複製するには、暗号化されたレプリケーションを使用できます。 **Note** 暗号化レプリケーションを使う必要がない場合、このステップをスキップできます。 暗号化レプリケーションを使用するための前提条件は次のとおりです。 • Secure Sockets Layer (SSL) は、外部の MySQL マスターデータベースで有効になっている必要があります。 • クライアントのキーとクライアントの証明書が Aurora MySQL DB クラスター用に準備されている必要があります。 暗号化のレプリケーション中、Aurora MySQL DB クラスターはクライアントとして MySQL データベースサーバーに動作します。Aurora MySQL 用の証明書およびキーは、.pem 形式のファイルにあります。 1. 暗号化レプリケーションのための準備があることを確認してください。 • 外部の MySQL マスターデータベースに有効になった SSL がなく、またクライアントキーおよびクライアント証明書が準備されていない場合、MySQL データベースサーバーで SSL を有効にし、必要なクライアントキーおよびクライアント証明書を生成します。 • SSL が外部マスターで有効になっている場合は、Aurora MySQL DB クラスターにクライアントキーおよび証明書を提供します。これらがない場合は、Aurora MySQL DB クラスター用に新しいキーと証明書を生成します。クライアント証明書に署名するには、外部の MySQL 出典データベースで SSL の設定に使用した認証局キーが必要です。

データベースエンジン	手順

詳細については、MySQL ドキュメントの「Creating SSL Certificates and Keys Using openssl」を参照してください。

認証局証明書、クライアントキーおよびクライアント証明書が必要となります。

2. SSL を使用して、マスターユーザーとして Aurora MySQL DB クラスターに接続します。

   SSL で Aurora MySQL DB クラスターに接続する詳細については、「Aurora MySQL DB クラスターでの TLS の使用」を参照してください。

3. `mysql.rds_import_binlog_ssl_material` ストアドプロシージャを実行して、Aurora MySQL DB クラスターに SSL 情報をインポートします。

   `ssl_material_value` パラメータには、正しい JSON ペイロードで Aurora MySQL DB クラスター用の .pem 形式から情報を挿入します。

   次の例では、SSL 情報を Aurora MySQL DB クラスターにインポートします。.pem 形式ファイルでは、通常の場合、コード本文に例に示されるコード本文より長くなっています。

```
call mysql.rds_import_binlog_ssl_material(
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcj
qP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96
xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/
i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz22
1CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICA
TE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcj
qP3maAhDFcvBS7O6V
```

データベースエンジン	手順

```
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96
xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/
i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz22
1CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE
 KEY-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pc
jqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSF
JuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJ
tjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMu
cxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

詳細については、mysql.rds_import_binlog_ssl_materialおよびAurora MySQL DB クラスターでの TLS の使用を参照してください。

> **Note**
>
> 手順を実行したあと、シークレットはファイルに保存されます。ファイルを後で消去するには、mysql.rds_remove_binlog_ssl_material ストアドプロシージャを実行できます。

外部 MySQL データベースのバイナリログ記録を有効にするには

1. コマンドラインシェルから、mysql サービスを停止します。

```
sudo service mysqld stop
```

2. `my.cnf` ファイルを編集します (このファイルは通常 `/etc` にあります)。

データベースエンジン	手順

```
sudo vi /etc/my.cnf
```

log_bin オプションと server_id オプションを [mysqld] に追加します。log_bin オプションは、バイナリログファイルのファイル名識別子を提供します。server_id オプションは、出典とレプリカの関係のサーバーに一意の識別子を提供します。

暗号化レプリケーションが求められない場合、バイナリログが有効で、SSL は無効な状態で外部 MySQL データベースがスタートされるようにします。

非暗号化データ用の /etc/my.cnf ファイルの関連するエントリを以下に示します。

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1
```

暗号化レプリケーションが求められる場合、外部 MySQL データベースが SSL およびバイナリログ有効化された状態でスタートすることを確認します。

/etc/my.cnf ファイルのエントリには、MySQL データベースサーバーの .pem ファイルの場所が含まれます。

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1

# Setup SSL.
ssl-ca=/home/sslcerts/ca.pem
ssl-cert=/home/sslcerts/server-cert.pem
ssl-key=/home/sslcerts/server-key.pem
```

データベースエンジン	手順

また、MySQL DB インスタンスの sql_mode オプションを 0 に設定するか、my.cnf ファイルから除外する必要があります。

外部の MySQL データベースに接続するときに、外部の MySQL データベースのバイナリログの場所を記録します。

```
mysql> SHOW MASTER STATUS;
```

出力は次のようになります。

```
+------------------+----------+--------------+------------------
+-------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
 Executed_Gtid_Set |
+------------------+----------+--------------+------------------
+-------------------+
| mysql-bin.000031 |      107 |              |                  |
               |
+------------------+----------+--------------+------------------
+-------------------+
1 row in set (0.00 sec)
```

詳細については、MySQL ドキュメントの Setting the replication source configuration を参照してください。

3. mysql サービスをスタートします。

```
sudo service mysqld start
```

2. レプリケーション出典のバイナリログを不要になるまで保持する

MySQL バイナリログのレプリケーションを使用する場合、Amazon RDS はレプリケーションプロセスを管理しません。したがって、レプリケーション出典のバイナリログファイルは、変更がレプリカ

に適用されるまで保持する必要があります。このメンテナンスによって、障害発生時にソースデータベースを復元しやすくなります。

次のステップに従って、データベースエンジンのバイナリログを保持します。

データベースエンジン	手順
Aurora MySQL	Aurora MySQL DB クラスターのバイナリログを保持するには Aurora MySQL DB クラスターのバイナリログファイルにはアクセスできません。そのため、確実に変更がレプリカに適用されてから、レプリケーション出典のバイナリログファイルが Amazon RDS によって削除されるように、バイナリログファイルの保持期間は十分に長く設定する必要があります。Aurora MySQL DB クラスターのバイナリログファイルは最大 90 日間、保持できます。 MySQL データベースまたは RDS for MySQL DB インスタンスをレプリカとしてレプリケーションを設定する場合、レプリカを作成するデータベースが巨大なときには、レプリカへのデータベースの初期のコピーが完了し、レプリカラグが 0 に達するまで、バイナリログファイルが保持されるように、保持期間を長く設定してください。 バイナリログの保持期間を設定するには、「mysql.rds_set_configuration」の手順を使用して、DB クラスターのバイナリログファイルの保持時間数に合わせて、`'binlog retention hours'` の設定パラメータを指定します。Aurora MySQL バージョン 2 以降、およびバージョン 3 の最大値は 2160 (90 日) です。 以下の例では、バイナリログファイルの保持期間を 6 日に設定しています。 `CALL mysql.rds_set_configuration('binlog retention hours', 144);` レプリケーションが開始された後、レプリカに対して `SHOW SLAVE STATUS` (Aurora MySQL バージョン 2) または `SHOW REPLICA STATUS` (Aurora MySQL バージョン 3) コマンドを実行し、`Seconds behind master` フィールドを調べることで、変更がレプリカに適用されたことを確認できます。`Seconds behind master` フィールドが 0 の場合、レプリカラグはありません。レプリカラグがないときは、`binlog retention hours` 設定パラメータをより短い期間に設定することで、バイナリログファイルの保持期間を短くします。

データベースエンジン	手順
	この設定を指定しない場合、Aurora MySQL のデフォルト値は 24 (1 日) です。 `'binlog retention hours'` に最大値より大きい値を指定すると、Aurora MySQL は最大値を使用します。
RDS for MySQL	Amazon RDS DB インスタンスのバイナリログを保持するには 前の行で説明したように、Amazon RDS DB インスタンスのバイナリログファイルを保持するには、保持期間を Aurora MySQL DB クラスターと同様に設定します。 DB インスタンスのリードレプリカを作成しても、Amazon RDS DB インスタンスのバイナリログファイルを保持できます。このリードレプリカはバイナリログファイルの保持専用に一時的に作成されます。リードレプリカが作成されたら、リードレプリカに対して mysql.rds_stop_replication プロシージャを呼び出します。レプリケーションの停止中に Amazon RDS がレプリケーション出典のバイナリログファイルを削除することはありません。永続レプリカとのレプリケーションを設定した後、レプリケーション出典と永続レプリカ間のレプリカラグ (`Seconds behind master` フィールド) が 0 に達したときに、リードレプリカを削除できます。
MySQL (外部)	外部 MySQL データベースのバイナリログを有効にするには 外部 MySQL データベースのバイナリログファイルは Amazon RDS によって管理されていないため、手動で削除されるまでは保持されます。 レプリケーションが開始された後、レプリカに対して `SHOW SLAVE STATUS` (Aurora MySQL バージョン 2) または `SHOW REPLICA STATUS` (Aurora MySQL バージョン 3) コマンドを実行し、`Seconds behind master` フィールドを調べることで、変更がレプリカに適用されたことを確認できます。`Seconds behind master` フィールドが 0 の場合、レプリカラグはありません。レプリカラグがないときは、古いバイナリログファイルを削除できます。

3. レプリケーションソースのスナップショットまたはダンプを作成する

レプリケーションソースのスナップショットまたはダンプを使用して、データのベースラインコピーをレプリカにロードし、その時点からレプリケーションを開始します。

次のステップに従って、データベースエンジンのレプリケーションソースのスナップショットまたはダンプを作成します。

データベースエンジン	手順

Aurora MySQL	Aurora MySQL DB クラスターのスナップショットを作成するには
	1. Amazon Aurora DB クラスターのスナップショットを作成します。詳細については、「DB クラスタースナップショットの作成」を参照してください。
	2. 先ほど作成した DB クラスターのスナップショットから復元することで、新しい Aurora DB クラスターを作成します。復元された DB クラスターの DB パラメータグループは、元の DB クラスターと同一のものを維持してください。これを実行することで、DB クラスターのコピーでも確実にバイナリログ作成が有効になります。詳細については、「DB クラスターのスナップショットからの復元」を参照してください。
	3. コンソールで、[データベース] を選択し、復元された Aurora DB クラスターのプライマリインスタンス (書き込み) を選択してその詳細を表示します。[最近のイベント] までスクロールします。binlog ファイル名と場所を含むイベントメッセージが表示されます。イベントメッセージの形式は以下のとおりです。
	`Binlog position from crash recovery is binlog-file-name  binlog-position`
	レプリケーションをスタートしたときの binlog ファイルの名前と場所の値を保存します。
	AWS CLI から describe-events コマンドを呼び出して、binlog ファイルの名前と場所を取得することもできます。次に `describe-events` コマンドの例とサンプル出力を示します。
	`PROMPT> aws rds describe-events`
	`{ "Events": [ {`

データベースエンジン	手順

```
            "EventCategories": [],
            "SourceType": "db-instance",
            "SourceArn": "arn:aws:rds:us-west-2:123456789012:
db:sample-restored-instance",
            "Date": "2016-10-28T19:43:46.862Z",
            "Message": "Binlog position from crash recovery is mysql-
bin-changelog.000003 4278",
            "SourceIdentifier": "sample-restored-instance"
        }
    ]
}
```

binlog ファイルの名前と位置は、MySQL エラーログで、MySQL バイナリログファイルの最後の場所を調べることでも確認できます。

4. レプリカターゲットが別の AWS アカウント、外部 MySQL データベース、または RDS for MySQL DB インスタンスによって所有される Aurora DB クラスターである場合、Amazon Aurora DB クラスターのスナップショットからデータをロードすることはできません。代わりに、MySQL クライアントを使用して DB クラスターに接続し、`mysqldump` コマンドを発行することで、Aurora DB クラスターのダンプを作成します。作成した Aurora DB クラスターのコピーに対して、必ず `mysqldump` コマンドを実行してください。次に例を示します。

```
PROMPT> mysqldump --databases <database_name> --single-transaction
--order-by-primary -r backup.sql -u <local_user> -p
```

5. 新しく作成した Aurora DB クラスターからデータのダンプを作成し終わったら、その DB クラスターはもう不要なため削除します。

データベースエンジン	手順
RDS for MySQL	Amazon RDS DB インスタンスのスナップショットを作成するには Amazon RDS DB インスタンスのリードレプリカを作成します。詳細については、Amazon Relational Database Service ユーザーガイドの「リードレプリカの作成」を参照してください。 1. リードレプリカに接続し、mysql.rds_stop_replication プロシージャを実行することでレプリケーションを停止します。 2. リードレプリカが 停止 している間に、リードレプリカに接続して、SHOW SLAVE STATUS (Aurora MySQL バージョン 2) または SHOW REPLICA STATUS (Aurora MySQL バージョン 3) コマンドを実行します。Relay_Master_Log_File フィールドから現在のバイナリログファイルの名前を、Exec_Master_Log_Pos フィールドからそのログファイルの場所を取得します。レプリケーションをスタートするときのために、これらの値を保存します。 3. リードレプリカが [Stopped] の状態のままで、リードレプリカの DB スナップショットを作成します。詳細については、Amazon Relational Database Service ユーザーガイドの「DB スナップショットの作成」を参照してください。 4. リードレプリカを削除します。

データベースエンジン	手順

MySQL (外部)	外部 MySQL データベースのダンプを作成するには
	1. ダンプを作成する前に、ダンプのバイナリログの場所がソースインスタンスのデータで更新されていることを確認する必要があります。そのためには、まず以下のコマンドを使用して、インスタンスへの書き込みオペレーションを停止する必要があります。
	`mysql> FLUSH TABLES WITH READ LOCK;`
	2. 以下の mysqldump コマンドを使用して、MySQL データベースのダンプを作成します。
	`PROMPT> sudo mysqldump --databases <database_name> --master-data=2 --single-transaction \ --order-by-primary -r backup.sql -u <local_user> -p`
	3. ダンプを作成した後、次のコマンドを実行して、MySQL データベースのテーブルのロックを解除します。
	`mysql> UNLOCK TABLES;`

4. レプリカターゲットにスナップショットまたはダンプをロードする

Amazon RDS の外部 MySQL データベースのダンプからデータをロードする場合、ダンプファイルのコピー先となる EC2 インスタンスを作成してから、その EC2 インスタンスから DB クラスターまたは DB インスタンスにデータをロードします。この方法では、ダンプファイルを EC2 インスタンスにコピーする前に圧縮して、Amazon RDS へのデータのコピーに関連するネットワークコストを削減できます。また、ダンプファイルを暗号化して、ネットワーク経由で転送されるデータを保護することもできます。

次のステップに従って、レプリケーションソースのスナップショットまたはダンプをデータベースエンジンのレプリカターゲットにロードします。

データベースエンジン	手順

Aurora MySQL	Aurora MySQL DB クラスターにスナップショットまたはダンプをロードするには
	• レプリケーション出典のスナップショットが DB クラスターのスナップショットである場合は、DB クラスターのスナップショットから復元することで、レプリカターゲットとして新しい Aurora MySQL DB クラスターを作成できます。詳細については、「DB クラスターのスナップショットからの復元」を参照してください。
	• レプリケーション出典のスナップショットが DB スナップショットである場合は、DB スナップショットから新しい Aurora MySQL DB クラスターにデータを移行できます。詳細については、「Amazon Aurora MySQL DB クラスターへのデータの移行」を参照してください。
	• レプリケーションソースのデータが mysqldump コマンドからの出力である場合は、以下のステップを実行します。
	1. mysqldump コマンドの出力をレプリケーション出典から、Aurora MySQL DB クラスターにも接続できる場所にコピーします。
	2. mysql コマンドを使用して Aurora MySQL DB クラスターに接続します。次に例を示します。
	`PROMPT> mysql -h <host_name> -port=3306 -u <db_master_user> -p`
	3. mysql プロンプトで、データベースダンプファイルの名前を渡して source コマンドを実行することで、Aurora MySQL DB クラスターにデータをロードします。例えば、以下のようになります。
	`mysql> source backup.sql;`

RDS for MySQL	Amazon RDS DB インスタンスにダンプをロードするには 1. mysqldump コマンドの出力をレプリケーション出典から、MySQL DB インスタンスにも接続できる場所にコピーします。 2. mysql コマンドを使用して MySQL DB インスタンスに接続します。次に例を示します。

データベースエンジン	手順
	``` PROMPT> mysql -h <host_name> -port=3306 -u <db_master_user> -p ``` 3. mysql プロンプトで、データベースダンプファイルの名前を渡して source コマンドを実行することで、MySQL DB インスタンスにデータをロードします。例えば、以下のようになります。 ``` mysql> source backup.sql; ```
MySQL (外部)	外部 MySQL データベースにダンプをロードするには 外部 MySQL データベースに DB スナップショットまたは DB クラスターのスナップショットをロードすることはできません。代わりに、mysqldump コマンドの出力を使用する必要があります。 1. mysqldump コマンドの出力をレプリケーション出典から、MySQL データベースにも接続できる場所にコピーします。 2. mysql コマンドを使用して MySQL データベースに接続します。次に例を示します。 ``` PROMPT> mysql -h <host_name> -port=3306 -u <db_master_user> -p ``` 3. mysql プロンプトで、データベースダンプファイルの名前を渡して source コマンドを実行することで、MySQL データベースにデータをロードします。次に例を示します。 ``` mysql> source backup.sql; ```

5. レプリケーションソースでレプリケーションユーザーを作成する

レプリケーション専用のユーザー ID をソースに作成します。次の例は、RDS for MySQL または外部の MySQL ソースデータベース用です。

```
mysql> CREATE USER 'repl_user'@'domain_name' IDENTIFIED BY 'password';
```

Aurora MySQL ソースデータベースの場合、skip_name_resolveDB クラスターパラメータは 1 (ON) に設定され、変更できないため、ドメイン名の代わりにホストの IP アドレスを使用する必要があります。詳細については、MySQL ドキュメントの「skip_name_resolve」を参照してください。

```
mysql> CREATE USER 'repl_user'@'IP_address' IDENTIFIED BY 'password';
```

ユーザーには REPLICATION CLIENT および REPLICATION SLAVE 権限が必要です。ユーザーのこれらの権限を付与します。

暗号化レプリケーションを使用する必要がある場合は、レプリケーションのユーザーに対して SSL 接続を要求します。例えば、以下のいずれかのステートメントを使用して、ユーザーアカウント repl_user に SSL 接続を要求できます。

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'IP_address';
```

```
GRANT USAGE ON *.* TO 'repl_user'@'IP_address' REQUIRE SSL;
```

> **Note**
>
> REQUIRE SSL が含まれていない場合には、レプリケーション接続がメッセージの表示なしで非暗号化接続に戻る場合があります。

6. レプリカターゲットでレプリケーションを有効にする

レプリケーションを有効化する前に、MySQL DB インスタンス レプリカターゲットの Aurora MySQL DB クラスターまたは RDS のスナップショットを手動で作成することをお勧めします。問題が発生し、DB クラスターまたは DB インスタンスレプリカターゲットとのレプリケーションを再開する必要がある場合は、このスナップショットから DB クラスターまたは DB インスタンスを復元できます。そのために、再びレプリカターゲットにデータをインポートする必要はありません。

次のステップに従って、データベースエンジンでレプリケーションを有効にします。

データベースエンジン	手順
Aurora MySQL	Aurora MySQL DB クラスターからのレプリケーションを有効にするには 1. 複製の開始点を見つけます。バイナリログのファイル名と位置が必要です。 DB クラスターのレプリカターゲットが以下から作成された場合: • DB クラスタースナップショット — 3. レプリケーションソースのスナップショットまたはダンプを作成する に示すように、復元した DB クラスターの最近のイベントからバイナリログファイルの名前と位置を取得します。 • DB snapshot – レプリケーションソースのスナップショットを作成した際、SHOW SLAVE STATUS (Aurora MySQL バージョン 2) または SHOW REPLICA STATUS (Aurora MySQL バージョン 3) コマンドから、バイナリログファイルの名前と位置を取得しました。 2. DB クラスターに接続し、前の手順からのバイナリログファイルの名前と場所を使用して次のプロシージャを呼び出し、レプリケーションソースとのレプリケーションを開始します。 • mysql.rds_set_external_source (Aurora MySQL バージョン 3) • mysql.rds_set_external_master (Aurora MySQL バージョン 2) • mysql.rds_start_replication (すべてのバージョン) 次の例は、Aurora MySQL バージョン 3 の場合です。 `CALL mysql.rds_set_external_source ('mydbinstance.123456789012 .us-east-1.rds.amazonaws.com', 3306, 'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0); CALL mysql.rds_start_replication;` SSL 暗号化を使用するには、最終値を 0 ではなく 1 に設定します。
RDS for MySQL	Amazon RDS DB インスタンスからのレプリケーションを有効にするには 1. DB インスタンスレプリカターゲットが DB スナップショットから作成された場合は、レプリケーションにはまずバイナリログファイルの名前と場所が必要になり

データベースエンジン	手順
	ます。レプリケーションソースのスナップショットを作成した際、SHOW SLAVE STATUS (Aurora MySQL バージョン 2) または SHOW REPLICA STATUS (Aurora MySQL バージョン 3) コマンドから、これらの値を取得しました。 2. DB インスタンスに接続し、mysql.rds_set_external_master (Aurora MySQL バージョン 2) または mysql.rds_set_external_source (Aurora MySQL バージョン 3) と mysql.rds_start_replication プロシージャを呼び出して、レプリケーションソースとのレプリケーションを開始します。前のステップで取得したバイナリログファイルの名前と場所を使用します。次に例を示します。 `CALL mysql.rds_set_external_master ('mydbcluster.cluster-12345 6789012.us-east-1.rds.amazonaws.com', 3306, 'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0); CALL mysql.rds_start_replication;` SSL 暗号化を使用するには、最終値を 0 ではなく 1 に設定します。

データベースエンジン	手順

MySQL (外部)	外部 MySQL データベースからのレプリケーションを有効にするには
	1. レプリケーションにまず必要になるバイナリログファイルの名前と場所を取得します。レプリケーションソースのスナップショットを作成した際、SHOW SLAVE STATUS (Aurora MySQL バージョン 2) または SHOW REPLICA STATUS (Aurora MySQL バージョン 3) コマンドから、これらの値を取得しました。外部の MySQL レプリカターゲットへのデータロードに mysqldump オプションを指定して実行した --master-data=2 コマンドの出力を使用した場合は、その出力にバイナリログファイルの名前と場所が含まれています。次に例を示します。
	`-- -- Position to start replication or point-in-time recovery from -- -- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin-changelog.000031', MASTER_LOG_POS=107;`
	2. 外部の MySQL レプリカターゲットに接続し、CHANGE MASTER TO と START SLAVE (Aurora MySQL バージョン 2) または START REPLICA (Aurora MySQL バージョン 3) を発行して、前のステップで確認したバイナリログファイル名と場所を使用してレプリケーションソースとのレプリケーションを開始します。次に例を示します。
	`CHANGE MASTER TO MASTER_HOST = 'mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com', MASTER_PORT = 3306, MASTER_USER = 'repl_user', MASTER_PASSWORD = 'password', MASTER_LOG_FILE = 'mysql-bin-changelog.000031', MASTER_LOG_POS = 107; -- And one of these statements depending on your engine version: START SLAVE; -- Aurora MySQL version 2 START REPLICA; -- Aurora MySQL version 3`

レプリケーションが失敗した場合、レプリカにおいて意図しない I/O が大幅に増加することで、パフォーマンスが低下する可能性があります。レプリケーションが失敗するか、不要になった場合は、mysql.rds_reset_external_master (Aurora MySQL バージョン 2) または mysql.rds_reset_external_source (Aurora MySQL バージョン 3) ストアドプロシージャを実行して、レプリケーション設定を削除できます。

リードレプリカへのレプリケーションを停止する場所の設定

Aurora MySQL バージョン 3.04 以降では、mysql.rds_start_replication_until (Aurora MySQL バージョン 3) ストアドプロシージャを使用してレプリケーションを開始してバイナリログファイルの指定した位置で停止できます。

リードレプリカへのレプリケーションをスタートして指定の位置でレプリケーションを停止するには

1. MySQL クライアントを使用して、マスターユーザーとしてレプリカ Aurora MySQL DB クラスターに接続します。
2. mysql.rds_start_replication_until (Aurora MySQL バージョン 3) ストアドプロシージャを実行します。

   次の例では、レプリケーションをスタートし、120 バイナリログファイルの場所 mysql-bin-changelog.000777 に達するまで変更をレプリケートします。災害対策シナリオでは、場所 120 は災害発生直前の時点として想定されます。

   ```
   call mysql.rds_start_replication_until(
     'mysql-bin-changelog.000777',
     120);
   ```

停止ポイントに達すると、レプリケーションは自動的に停止します。RDS イベントとして、Replication has been stopped since the replica reached the stop point specified by the rds_start_replication_until stored procedure が生成されます。

GTID ベースのレプリケーションを使用する場合は、mysql.rds_start_replication_until_gtid (Aurora MySQL バージョン 3) ストアドプロシージャの代わりに、mysql.rds_start_replication_until (Aurora MySQL バージョン 3) ストアドプロシージャを実行します。GTID ベースのレプリケーションの詳細については、「GTID ベースレプリケーションを使用する」を参照してください。

7. レプリカをモニタリングする

Aurora MySQL DB クラスターと MySQL との間のレプリケーションを設定する場合、Aurora MySQL DB クラスターがレプリカターゲットであれば、そのクラスターのフェイルオーバーイベントをモニタリングする必要があります。フェイルオーバーが発生すると、レプリカターゲットである DB クラスターが、新しいホスト上に別のネットワークアドレスで再作成されます。フェイルオーバーイベントをモニタリングする方法については、「Amazon RDS イベント通知の操作」を参照してください。

また、レプリカターゲットがどれほどレプリケーションソースに遅れをとっているかをモニタリングするには、レプリカターゲットに接続して、`SHOW SLAVE STATUS` (Aurora MySQL バージョン 2) または `SHOW REPLICA STATUS` (Aurora MySQL バージョン 3) コマンドを実行します。このコマンドの出力の `Seconds Behind Master` フィールドに、マスターからレプリカターゲットへのコピーの進行状況が示されます。

レプリケーション出典とレプリケーションターゲット間でのパスワードの同期

SQL ステートメントを使用してレプリケーション出典のユーザーアカウントとパスワードを変更すると、それらの変更はレプリケーションターゲットに自動的にレプリケートされます。

AWS Management Console、AWS CLI、または RDS API を使用してレプリケーション出典のマスターパスワードを変更した場合、それらの変更はレプリケーションターゲットに自動的にレプリケートされません。出典システムとターゲットシステム間でマスターユーザーとマスターパスワードを同期する場合は、レプリケーションターゲットに対して同様の変更を自分で行う必要があります。

Aurora と MySQL との間、または Aurora と別の Aurora DB クラスターとの間のレプリケーションの停止

MySQL DB インスタンス、外部 MySQL データベース、または別の Aurora DB クラスターでのバイナリログのレプリケーションを停止するには、このトピックの後で詳しく説明するステップに従ってください。

1. レプリカターゲットでバイナリログのレプリケーションを停止する

2. レプリケーション出典のバイナリログ記録を無効にする

1. レプリカターゲットでバイナリログのレプリケーションを停止する

データベースエンジンのバイナリログレプリケーションを停止するには、以下の説明を参照してください。

データベースエンジン	手順
Aurora MySQL	Aurora MySQL DB クラスターレプリカターゲットでのバイナリログのレプリケーションを停止するには レプリカターゲットである Aurora DB クラスターに接続し、mysql.rds_stop_replication プロシージャを呼び出します。
RDS for MySQL	Amazon RDS DB インスタンスのバイナリログのレプリケーションを停止するには レプリカターゲットである RDS DB インスタンスに接続し、mysql.rds_stop_replication プロシージャを呼び出します。
MySQL (外部)	外部 MySQL データベースのバイナリログのレプリケーションを停止するには MySQL データベースに接続し、STOP SLAVE (バージョン 5.7) またはSTOP REPLICA (バージョン 8.0) コマンドを実行します。

2. レプリケーション出典のバイナリログ記録を無効にする

次の表の説明に従って、データベースエンジンのレプリケーションソースで、バイナリログを無効にします。

データベースエンジン	手順
Aurora MySQL	Amazon Aurora DB クラスターのバイナリログ記録を無効にするには 1. レプリケーションソースである Aurora DB クラスターに接続します。 2. mysql.rds_set_configuration プロシージャを使用して、次の例に示すように値 NULL で設定パラメータ binlog retention hours を指定します。 `CALL mysql.rds_set_configuration('binlog retention hours', NULL);`

データベースエンジン	手順
	Note `binlog retention hours` には、値 `0` は使用できません。 3. レプリケーション出典で `binlog_format` パラメータを `OFF` に設定します。`binlog_format` パラメータは、DB クラスターに関連付けられているカスタム DB クラスターパラメータグループにあります。 `binlog_format` パラメータ値を変更した後、変更を有効にするために DB クラスターを再起動します。 詳細については、Amazon Aurora の DB クラスターパラメータと DB インスタンスパラメータおよびDB パラメータグループのパラメータの変更を参照してください。
RDS for MySQL	Amazon RDS DB インスタンスのバイナリログ記録を無効にするには Amazon RDS DB インスタンスでは、バイナリログ記録を直接無効にすることはできませんが、以下のいずれかの操作により有効にすることができます。 1. DB インスタンスの自動バックアップを無効にする。既存の DB インスタンスを変更して Backup Retention Period (バックアップ保持期間) を 0 に設定することで、自動バックアップを無効にできます。詳細については、Amazon Relational Database Service ユーザーガイドの「Amazon RDS DB インスタンスの変更」および「バックアップの操作」を参照してください。 2. DB インスタンスのすべてのリードレプリカを削除します。詳細については、Amazon Relational Database Service ユーザーガイドの「MariaDB、MySQL、PostgreSQL DB インスタンスのリードレプリカの使用」を参照してください。

データベースエンジン	手順
MySQL (外部)	外部 MySQL データベースのバイナリログ記録を無効にするには MySQL データベースに接続して、`STOP REPLICATION` コマンドを呼び出します。 1. コマンドラインシェルから、`mysqld` サービスを停止します。 ``` sudo service mysqld stop ``` 2. `my.cnf` ファイルを編集します (このファイルは通常 `/etc` にあります)。 ``` sudo vi /etc/my.cnf ``` `log_bin` セクションから `server_id` および `[mysqld]` オプションを削除します。 詳細については、MySQL ドキュメントの Setting the replication source configuration を参照してください。 3. mysql サービスをスタートします。 ``` sudo service mysqld start ```

Amazon Aurora を使用した MySQL データベースの読み取りスケーリング

MySQL DB インスタンスで Amazon Aurora を使用することで、Amazon Aurora の読み取りスケーリング機能を活用して MySQL DB インスタンスの読み取りワークロードを拡張できます。Aurora を使用して MySQL DB インスタンスの読み取りを拡張するには、Amazon Aurora MySQL DB クラスターを作成し、MySQL DB インスタンスのリードレプリカに指定します。これは、RDS for MySQL DB インスタンス、または Amazon RDS の外部で実行されている MySQL データベースに適用されます。

Amazon Aurora DB クラスターの作成については、「Amazon Aurora DB クラスターの作成」を参照してください。

MySQL DB インスタンスと Amazon Aurora DB クラスターの間でレプリケーションを設定するときは、以下のガイドラインに従ってください。

- Amazon Aurora DB クラスターを参照するときは、Amazon Aurora MySQL DB クラスターのエンドポイントアドレスを使用します。フェイルオーバーが発生すると、Aurora MySQL DB クラスターのプライマリインスタンスに昇格された Aurora レプリカで、引き続きこの DB クラスターのエンドポイントアドレスが使用されます。
- ライターインスタンスのバイナリログが Aurora レプリカに適用されたことを確認するまで、これらのバイナリログを保持します。このメンテナンスによって、障害発生時にライターインスタンスを復元できます。

Important

自己管理型レプリケーションを使用する場合、ユーザー自身で発生する可能性のあるすべてのレプリケーションの問題をモニタリングし、解決する必要があります。詳細については、「リードレプリカ間の遅延の診断と解決」を参照してください。

Note

Amazon Aurora MySQL DB クラスターでレプリケーションを開始するために必要なアクセス許可は制限されており、Amazon RDS マスターユーザーは使用できません。このため、Aurora MySQL DB クラスターと MySQL DB インスタンスの間でレプリケーションを設定するには、mysql.rds_set_external_master (Aurora MySQL バージョン 2) または mysql.rds_set_external_source (Aurora MySQL バージョン 3) と mysql.rds_start_replication プロシージャを使用する必要があります。

外部のソースインスタンスと Aurora MySQL DB クラスターの間でレプリケーションを開始する

1. 出典 MySQL DB インスタンスを読み取り専用にします。

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SET GLOBAL read_only = ON;
```

2. 出典 MySQL DB インスタンスで SHOW MASTER STATUS コマンドを実行して、binlog の場所を特定します。以下の例のような出力を受け取ります。

```
File                          Position
-------------------------------------
 mysql-bin-changelog.000031      107
-------------------------------------
```

3. mysqldump を使用して、外部の MySQL DB インスタンスから Amazon Aurora MySQL DB クラスターにデータベースをコピーします。大規模なデータベースの場合、Amazon Relational Database Service ユーザーガイドの「ダウンタイムを短縮して MySQL または MariaDB DB インスタンスにデータをインポートする」の手順を使用することが必要になる場合があります。

   Linux、macOS、Unix の場合:

```
mysqldump \
    --databases <database_name> \
    --single-transaction \
    --compress \
    --order-by-primary \
    -u local_user \
    -p local_password | mysql \
        --host aurora_cluster_endpoint_address \
        --port 3306 \
        -u RDS_user_name \
        -p RDS_password
```

   Windows の場合:

```
mysqldump ^
    --databases <database_name> ^
    --single-transaction ^
    --compress ^
    --order-by-primary ^
    -u local_user ^
    -p local_password | mysql ^
        --host aurora_cluster_endpoint_address ^
        --port 3306 ^
        -u RDS_user_name ^
        -p RDS_password
```

> **Note**
>
> -p オプションと入力するパスワードの間にスペースがないことを確認します。

--host コマンドで、--user (-u)、--port、-p、mysql オプションを使用して、Aurora DB クラスターに接続するためのホスト名、ユーザー名、ポート、パスワードを指定します。このホスト名は、Amazon Aurora DB クラスターのエンドポイントの DNS 名 (例えば mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com) です。エンドポイントの値は、Amazon RDS マネジメントコンソールでクラスターの詳細を確認できます。

4. もう一度出典 MySQL DB インスタンスを書き込み可能にします。

```
mysql> SET GLOBAL read_only = OFF;
mysql> UNLOCK TABLES;
```

レプリケーションで使用するバックアップの作成の詳細については、MySQL ドキュメントの Backing up a source or replica by making it read only を参照してください。

5. Amazon RDS マネジメントコンソールで、出典 MySQL データベースをホストするサーバーの IP アドレスを、Amazon Aurora DB クラスターの VPC セキュリティグループに追加します。VPC セキュリティグループの変更方法の詳細については、Amazon Virtual Private Cloud ユーザーガイドの「VPC のセキュリティグループ」を参照してください。

出典 MySQL インスタンスと通信できるようにするために、Amazon Aurora DB クラスターの IP アドレスからの接続を許可するようにローカルネットワークを設定することも必要になる場合があります。Amazon Aurora DB クラスターの IP アドレスを確認するには、host コマンドを使用します。

```
host aurora_endpoint_address
```

このホスト名は、Amazon Aurora DB クラスターのエンドポイントからの DNS 名です。

6. 選択したクライアントを使用して、外部の MySQL インスタンスに接続し、レプリケーションに使用される MySQL ユーザーを作成します。このアカウントはレプリケーション専用に使用され、セキュリティを強化するためにドメインに制限する必要があります。次に例を示します。

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
```

7. 外部の MySQL インスタンスについて、REPLICATION CLIENT と REPLICATION SLAVE の特権をレプリケーションユーザーに付与します。例えば、すべてのデータベースに対する REPLICATION CLIENT および REPLICATION SLAVE 権限を "repl_user" ユーザーに付与するには、以下のコマンドを実行します。

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
    IDENTIFIED BY 'password';
```

8. レプリケーションを設定する前に、Aurora MySQL DB クラスターの手動スナップショットをリードレプリカに指定します。DB クラスターをリードレプリカとしてレプリケーションを再構築する必要がある場合は、このスナップショットから Aurora MySQL DB クラスターを復元でき、MySQL DB インスタンスから新しい Aurora MySQL DB クラスターにデータをインポートする必要はありません。

9. Amazon Aurora DB クラスターをレプリカとして指定します。Amazon Aurora DB クラスターにマスターユーザーとして接続し、mysql.rds_set_external_master (Aurora MySQL バージョン 2) または mysql.rds_set_external_source (Aurora MySQL バージョン 3) と mysql.rds_start_replication プロシージャを使用して、ソース MySQL データベースをレプリケーションマスターとして指定します。

   ステップ 2 で特定したマスターログファイル名とマスターログの場所を使用します。次に例を示します。

```
For Aurora MySQL version 2:
CALL mysql.rds_set_external_master ('mymasterserver.mydomain.com', 3306,
    'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0);

For Aurora MySQL version 3:
CALL mysql.rds_set_external_source ('mymasterserver.mydomain.com', 3306,
    'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0);
```

10. Amazon Aurora DB クラスターで、mysql.rds_start_replication プロシージャを呼び出してレプリケーションを開始します。

```
CALL mysql.rds_start_replication;
```

出典 MySQL DB インスタンスと Amazon Aurora DB クラスター間のレプリケーションが確立されると、Aurora レプリカを Amazon Aurora DB クラスターに追加できます。その後で、Aurora レプリカ

に接続してデータの読み取りを拡張できます。Aurora レプリカの作成については、「DB クラスターに Aurora レプリカを追加する」を参照してください。

バイナリログのレプリケーションの最適化

次に、Aurora MySQL でバイナリログのレプリケーションのパフォーマンスを最適化し、関連する問題のトラブルシューティングを行う方法について説明します。

> **Tip**
>
> この説明は、MySQL バイナリログのレプリケーションメカニズムとその仕組みに精通していることを前提としています。背景情報については、MySQL ドキュメントの「レプリケーション実装ガイド」を参照してください。

マルチスレッドバイナリログレプリケーション

マルチスレッドのバイナリログレプリケーションでは、SQL スレッドはリレーログからイベントを読み取り、SQL ワーカースレッドが適用されるようにキューに入れます。SQL ワーカースレッドは、コーディネータスレッドによって管理されます。バイナリログイベントは、可能な場合はパラレルに適用されます。

マルチスレッドバイナリログレプリケーションは、Aurora MySQL バージョン 3 および Aurora MySQL バージョン 2.12.1 以降でサポートされています。

Aurora MySQL DB インスタンスがバイナリログレプリケーションを使用するように構成されている場合、レプリカインスタンスはデフォルトで 3.04 より前の Aurora MySQL バージョンに対してシングルスレッドレプリケーションを使用します。マルチスレッドレプリケーションを有効にするには、カスタムパラメータグループの `replica_parallel_workers` パラメータを 0 より大きい値に設定します。

Aurora MySQL バージョン 3.04 以降では、レプリケーションはデフォルトでマルチスレッド化され、`replica_parallel_workers` は `4` に設定されています。このパラメータはカスタムパラメータグループで変更できます。

以下の構成オプションを使用して、マルチスレッドレプリケーションを微調整することができます。使用量に関する情報については、MySQL リファレンスマニュアルのレプリケーションとバイナリログのオプションと可変を参照してください。

最適な構成は、いくつかの要因によって異なります。例えば、バイナリログレプリケーションのパフォーマンスは、データベースワークロードの特性と、レプリカが実行されている DB インスタンス

クラスの影響を受けます。したがって、新しいパラメータ設定を本番インスタンスに適用する前に、これらの構成パラメータに対するすべての変更を徹底的にテストすることをお勧めします。

- binlog_group_commit_sync_delay
- binlog_group_commit_sync_no_delay_count
- binlog_transaction_dependency_history_size
- binlog_transaction_dependency_tracking
- replica_preserve_commit_order
- replica_parallel_type
- replica_parallel_workers

Aurora MySQL バージョン 3.06 以降では、複数のセカンダリインデックスを持つ大きなテーブルのトランザクションをレプリケートするときにバイナリログレプリカのパフォーマンスを向上させることができます。この機能により、バイナリログレプリカにセカンダリインデックスの変更を並列で適用するスレッドプールが導入されます。この機能は aurora_binlog_replication_sec_index_parallel_workers DB クラスターパラメータによって制御されます。これにより、セカンダリインデックスの変更を適用できる並列スレッドの総数が制御されます。パラメータは、デフォルトで 0 (無効) に設定されています。この機能を有効にしてもインスタンスを再起動する必要はありません。この機能を有効にするには、進行中のレプリケーションを停止し、必要な数の並列ワーカースレッドを設定してから、レプリケーションを再開します。

また、このパラメータをグローバル変数として使用することもできます。ここで、*n* は並列ワーカースレッドの数です。

```
SET global aurora_binlog_replication_sec_index_parallel_workers=n;
```

バイナリログレプリケーションの最適化 (Aurora MySQL 2.10 以降)

Aurora MySQL 2.10 以降では、Aurora は、バイナリログのレプリケーションにバイナリログ I/O キャッシュと呼ばれる最適化を自動的に適用します。最後にコミットされたバイナリログイベントをキャッシュすることにより、この最適化は、バイナリログ出典インスタンスでのフォアグラウンドトランザクションへの影響を制限しながら、バイナリログのダンプスレッドのパフォーマンスを向上するように設計されています。

> **Note**
>
> この機能に使用されるこのメモリは、MySQL binlog_cache 設定とは無関係です。
> この機能は、db.t2 および db.t3 インスタンスクラスを使用する Aurora DB インスタンスには適用されません。

この最適化を有効にするために、設定パラメータを調整する必要はありません。特に、以前の Aurora MySQL バージョンで設定パラメータ aurora_binlog_replication_max_yield_seconds をゼロ以外の値に調整する場合は、Aurora MySQL 2.10 以降ではゼロに戻します。

ステータス可変 aurora_binlog_io_cache_reads および aurora_binlog_io_cache_read_requests は Aurora MySQL 2.10 以降で使用できます。これらのステータス可変は、バイナリログ I/O キャッシュからデータが読み込まれる頻度をモニタリングするのに役立ちます。

- aurora_binlog_io_cache_read_requests はキャッシュからのバイナリログ I/O 読み取りリクエストの数を示します。
- aurora_binlog_io_cache_reads はキャッシュから情報を取得するバイナリログ I/O 読み取り数を示します。

次の SQL クエリは、キャッシュされた情報を利用するバイナリログ読み取りリクエストの割合を計算します。この場合、比率が 100 に近づくほど、より良好であることを意味します。

```
mysql> SELECT
  (SELECT VARIABLE_VALUE FROM INFORMATION_SCHEMA.GLOBAL_STATUS
    WHERE VARIABLE_NAME='aurora_binlog_io_cache_reads')
  / (SELECT VARIABLE_VALUE FROM INFORMATION_SCHEMA.GLOBAL_STATUS
    WHERE VARIABLE_NAME='aurora_binlog_io_cache_read_requests')
  * 100
  as binlog_io_cache_hit_ratio;
+---------------------------+
| binlog_io_cache_hit_ratio |
+---------------------------+
|         99.99847949080622 |
+---------------------------+
```

バイナリログ I/O キャッシュ機能には、バイナリログのダンプスレッドに関連する新しいメトリクスも含まれています。ダンプスレッドは、新しいバイナリログレプリカがバイナリログ出典インスタンスに接続したときに作成されるスレッドです。

ダンプスレッドメトリクスは、60 秒ごとにプレフィックス `[Dump thread metrics]` を伴ってデータベースログに出力されます。メトリクスには、`Secondary_id`、`Secondary_uuid`、バイナリログのファイル名、各レプリカが読み込んでいる位置など、各バイナリログのレプリカの情報が含まれます。メトリクスには、レプリケーション出典とレプリカの間の距離をバイト単位で表す `Bytes_behind_primary` も含まれます。このメトリクスは、レプリカ I/O スレッドのラグを測定します。この数値は、バイナリログのレプリカの `seconds_behind_master` メトリクスによって表されるレプリカ SQL 適用元スレッドのラグとは異なります。距離が減少するか増加するかを確認することで、バイナリログレプリカが出典に追いついているのか、遅れているのかを判断できます。

バイナリログレプリケーションの最適化 (Aurora MySQL バージョン 2 ~ 2.09)

Aurora MySQL のバイナリログのレプリケーションを最適化するには、以下のクラスターレベルの最適化パラメータを調整します。これらのパラメータは、バイナリログの出典インスタンスのレイテンシーとレプリケーションラグの間の適切なバランスを特定するのに役立ちます。

- `aurora_binlog_use_large_read_buffer`
- `aurora_binlog_read_buffer_size`
- `aurora_binlog_replication_max_yield_seconds`

> **Note**
>
> MySQL 5.7 互換クラスターの場合、Aurora MySQL バージョン 2 から 2.09.* でこれらのパラメータを使用できます。Aurora MySQL 2.10.0 以降では、これらのパラメータはバイナリログ I/O キャッシュの最適化によって置き換えられ、それらを使用する必要はありません。

トピック

- 大きな読み取りバッファと最大収量の最適化の概要
- 関連パラメータ
- バイナリログレプリケーションの最大収率メカニズムの有効化
- バイナリログのレプリケーションを無効化する最大収率の最適化
- 大きい読み取りバッファを無効化する

大きな読み取りバッファと最大収量の最適化の概要

クラスターが多数のトランザクションを処理している間、バイナリログのダンプスレッドが Aurora クラスターボリュームにアクセスすると、バイナリログレプリケーションのパフォーマンスが低下することがあります。パラメータ aurora_binlog_use_large_read_buffer、aurora_binlog_replication_max_yield_seconds を使用すると、このタイプの競合を最小限に抑えることができます。

例えばaurora_binlog_replication_max_yield_seconds が 0 より大きい値に設定され、ダンプスレッドの現在のバイナリログファイルがアクティブな状況にあるとします。この場合、バイナリログのダンプスレッドは、トランザクションによって現在のバイナリログファイルがいっぱいになるまで、指定された秒数待ちます。この待機期間により、各バイナリログイベントを個別にレプリケートすることによって発生する可能性のある競合が回避されます。ただし、それによりバイナリログレプリカのレプリカラグが増加します。これらのレプリカは、aurora_binlog_replication_max_yield_seconds 設定と同じ秒数だけ出典より遅れる可能性があります。

現在のバイナリログファイルとは、スレッドダンプがレプリケーションを実行するために現在読み込んでいるバイナリログファイルのことです。バイナリログファイルが更新中または受信トランザクションによる更新のために開かれているとき、バイナリログファイルはアクティブであると見なします。Aurora MySQL がアクティブなバイナリログファイルでいっぱいになると、MySQL が新しいバイナリログファイルを作成して切り替えます。古いバイナリログファイルは非アクティブになります。受信トランザクションによって更新されることはありません。

> **Note**
>
> これらのパラメータを調整する前に、トランザクションのレイテンシーとスループットを長期間測定してください。バイナリログレプリケーションのパフォーマンスは安定しており、競合が時折発生する場合でも低レイテンシーとなります。

`aurora_binlog_use_large_read_buffer`

このパラメータが 1 に設定されている場合、パラメータ aurora_binlog_read_buffer_size および aurora_binlog_replication_max_yield_seconds の設定に基づいて、Aurora MySQL がバイナリログのレプリケーションを最適化します。aurora_binlog_use_large_read_buffer が 0 の場合、Aurora MySQL はパラメータ aurora_binlog_read_buffer_size および aurora_binlog_replication_max_yield_seconds の値を無視します。

`aurora_binlog_read_buffer_size`

読み取りバッファが大きいバイナリログのスレッドダンプは、I/O ごとにより多くのイベントを読み取ることで、読み取り I/O オペレーションの数を最小化しています。読み取りバッファのサイズは、`aurora_binlog_read_buffer_size` パラメータで設定します。読み取りバッファが大きいと、大量のバイナリログデータを生成するワークロードのバイナリログの競合を減らすことができます。

> **Note**
>
> このパラメータは、クラスターにも設定 `aurora_binlog_use_large_read_buffer=1` がある場合にのみ有効です。
> 読み取りバッファサイズを拡張しても、バイナリログのレプリケーションのパフォーマンスに影響はありません。バイナリログのダンプスレッドは、読み取りバッファを満たすトランザクションの更新を待ちません。

`aurora_binlog_replication_max_yield_seconds`

ワークロードに必要なトランザクションのレイテンシーが低く、レプリケーションラグを許容できる場合は、`aurora_binlog_replication_max_yield_seconds` パラメータを増やすことができます。このパラメータにより、クラスター内のバイナリログレプリケーションの最大収率のプロパティが制御されます。

> **Note**
>
> このパラメータは、クラスターにも設定 `aurora_binlog_use_large_read_buffer=1` がある場合にのみ有効です。

Aurora MySQL は、`aurora_binlog_replication_max_yield_seconds` パラメータ値への変更をただちに認識します。DB インスタンスを再起動する必要はありません。ただし、この設定を有効にすると、現在のバイナリログファイルが最大サイズの 128 MB に達し、新しいファイルにローテーションされた場合にのみ、ダンプスレッドの生成がスタートされます。

関連パラメータ

次の DB クラスターパラメータを使用して、バイナリログの最適化を有効にします。

パラメータ	デフォルト値	有効な値	説明
aurora_binlog_use_large_read_buffer	1	0、1	レプリケーション改善の機能を有効化するスイッチ。値が 1 のとき、バイナリログのダンプスレッドによって、バイナリログのレプリケーションに aurora_binlog_read_buffer_size が使用されます。それ以外の場合は、デフォルトのバッファサイズ (8K) が使用されます。Aurora MySQL バージョン 3 では使用されません。
aurora_binlog_read_buffer_size	5242880	8192-536870912	パラメータ aurora_binlog_use_large_read_buffer が 1 に設定されている場合に、バイナリログのダンプスレッドで使用される読み取りバッファサイズ。Aurora MySQL バージョン 3 では使用されません。
aurora_binlog_repl	0	0 ~ 36000	Aurora MySQL バージョン 2.07.* の場

パラメータ	デフォルト値	有効な値	説明
ication_max_yield_seconds			合、受け入れられる最大値は 45 です。2.09 以降のバージョンでは、より高い値に調整できます。 バージョン 2 の場合、このパラメータは、パラメータ aurora_binlog_use_large_read_buffer が 1 に設定されている場合にのみ機能します。

バイナリログレプリケーションの最大収率メカニズムの有効化

次のように、バイナリログレプリケーションの最大収率の最適化を有効にすることができます。これにより、バイナリログの出典インスタンスでのトランザクションのレイテンシーが最小限に抑えられます。ただし、レプリケーションラグが大きくなる場合があります。

Aurora MySQL クラスターのバイナリログ最大収率の最適化を有効にするには

1. DB クラスターパラメータグループを作成または編集するには、以下のパラメータ設定を使用します。
   - aurora_binlog_use_large_read_buffer: ON または 1 値で有効化します。
   - aurora_binlog_replication_max_yield_seconds: 0 より大きい値を指定します。
2. DB クラスターのパラメータグループを、バイナリログの出典として機能する Aurora MySQL クラスターに関連付けます。そのためには、「「パラメータグループを使用する」」の手順に従います。
3. パラメータの変更が有効なっていることを確認します。これを行うには、バイナリログの出典インスタンスで以下のクエリを実行します。

```
SELECT @@aurora_binlog_use_large_read_buffer,
 @@aurora_binlog_replication_max_yield_seconds;
```

出力は次のようになります。

```
+---------------------------------------
+-----------------------------------------------+
| @@aurora_binlog_use_large_read_buffer |
 @@aurora_binlog_replication_max_yield_seconds |
+---------------------------------------
+-----------------------------------------------+
|                                     1 |
  45 |
+---------------------------------------
+-----------------------------------------------+
```

バイナリログのレプリケーションを無効化する最大収率の最適化

以下のように、バイナリログのレプリケーションの最大収率の最適化を無効化することができます。これにより、レプリケーションラグが最小限に抑えられます。ただし、バイナリログの出典インスタンスでのトランザクションのレイテンシーが高くなることがあります。

Aurora MySQL クラスターの最大収率の最適化を無効化するには

1. Aurora MySQL クラスターに関連付けられている DB クラスターのパラメータグループで、aurora_binlog_replication_max_yield_seconds が 0 に設定されていることを確認します。パラメータグループを使用して設定パラメータの設定の詳細については、「「パラメータグループを使用する」」を参照してください。
2. パラメータの変更が有効なっていることを確認します。これを行うには、バイナリログの出典インスタンスで以下のクエリを実行します。

```
SELECT @@aurora_binlog_replication_max_yield_seconds;
```

出力は次のようになります。

```
+-----------------------------------------------+
| @@aurora_binlog_replication_max_yield_seconds |
+-----------------------------------------------+
```

```
|                                                 0 |
+---------------------------------------------------+
```

大きい読み取りバッファを無効化する

以下のように、大きい読み取りバッファの機能全体を無効化することができます。

Aurora MySQL クラスターで大きいバイナリログの読み取りバッファを無効化するには

1. `aurora_binlog_use_large_read_buffer` を `OFF` または `0` にリセットします。

   Aurora MySQL クラスターに関連付けられている DB クラスターのパラメータグループで、`aurora_binlog_use_large_read_buffer` が `0` に設定されていることを確認します。パラメータグループを使用して設定パラメータの設定の詳細については、「「パラメータグループを使用する」」を参照してください。
2. バイナリログの出典インスタンスで、以下のクエリを実行します。

   ```
   SELECT @@ aurora_binlog_use_large_read_buffer;
   ```

   出力は次のようになります。

   ```
   +---------------------------------------+
   | @@aurora_binlog_use_large_read_buffer |
   +---------------------------------------+
   |                                     0 |
   +---------------------------------------+
   ```

拡張バイナリログ記録の設定

拡張バイナリログを使用すると、バイナリログを有効にすることによるコンピューティングパフォーマンスのオーバーヘッドを、場合によっては最大 50% まで削減できます。拡張バイナリログを使用すると、このオーバーヘッドを約 13% まで削減できます。オーバーヘッドを減らすために、拡張バイナリログはバイナリログとトランザクションログをストレージに並行に書き込みます。これにより、トランザクションコミット時に書き込まれるデータが最小限に抑えられます。

また、拡張バイナリログを使用すると、コミュニティ MySQL バイナリログと比較して、再起動およびフェイルオーバー後のデータベースの回復時間が最大 99% 向上します。拡張バイナリログは既存

のバイナリログベースのワークロードと互換性があり、コミュニティ MySQL バイナリログと同じ方法で操作できます。

拡張バイナリログは Aurora MySQL 3.03.1 バージョン以降で利用できます。

トピック

- 拡張バイナリログパラメータの設定
- その他の関連パラメータ
- 拡張バイナリログとコミュニティ MySQL バイナリログの違い
- 拡張バイナリログの Amazon CloudWatch メトリクス
- 拡張バイナリログの制限

拡張バイナリログパラメータの設定

拡張バイナリログパラメータをオン/オフにすることで、コミュニティ MySQL バイナリログと拡張バイナリログを切り替えることができます。既存のバイナリログコンシューマーは、バイナリログファイルシーケンスにギャップがなく、引き続きバイナリログファイルを読み込んで使用できます。

拡張バイナリログを無効化するには

パラメータ	デフォルト値	説明
`binlog_format`	–	`binlog_format` パラメータを、選択したバイナリログ形式に設定して、拡張バイナリログをオンにします。`binlog_format parameter` がオフに設定されていないことを確認してください。詳細については、「Aurora MySQL バイナリログの設定」を参照してください。
`aurora_enhanced_binlog`	`0`	Aurora MySQL クラスターに関連付けられている DB クラスターのパラメータグループで、このパラメータ

パラメータ	デフォルト値	説明
		の値を 1 に設定します。このパラメータの値を変更する場合、DBClusterParameterGroupStatus 値が pending-reboot と表示されている状態でライターインスタンスを再起動する必要があります。
binlog_backup	1	拡張バイナリログをオンにするには、このパラメータをオフにしてください。そのためには、このパラメータの値を 0 に設定します。
binlog_replication_globaldb	1	拡張バイナリログをオンにするには、このパラメータをオフにしてください。そのためには、このパラメータの値を 0 に設定します。

Important

binlog_backup と binlog_replication_globaldb パラメータは、拡張バイナリログを使用する場合にのみオフにできます。

拡張バイナリログをオフにするには

パラメータ	説明
aurora_enhanced_binlog	Aurora MySQL クラスターに関連付けられている DB クラスターのパラメータグループで、このパラメータの値を 0 に設定します。このパラメータの値を変更する際は必ず、DBCluster

パラメータ	説明
	ParameterGroupStatus 値が pending-reboot と表示されている状態でライターインスタンスを再起動する必要があります。
binlog_backup	拡張バイナリログをオフにするには、このパラメータをオンにしてください。そのためには、このパラメータの値を 1 に設定します。
binlog_replication_globaldb	拡張バイナリログをオフにするには、このパラメータをオンにしてください。そのためには、このパラメータの値を 1 に設定します。

拡張バイナリログがオンになっているかどうかを確認するには、MySQL クライアントで次のコマンドを実行します。

```
mysql>show status like 'aurora_enhanced_binlog';

+-------------------------+--------+
| Variable_name           | Value  |
+-------------------------+--------+
| aurora_enhanced_binlog | ACTIVE |
+-------------------------+--------+
1 row in set (0.00 sec)
```

拡張バイナリログがオンになっている場合、aurora_enhanced_binlog の出力は ACTIVE と表示されます。

その他の関連パラメータ

拡張バイナリログを有効にすると、次のパラメータが影響を受けます。

- max_binlog_size パラメータは表示されますが、変更できません。デフォルト値 134217728 は、拡張バイナリログがオンになったときに 268435456 に自動調整されます。
- コミュニティ MySQL バイナリログとは異なり、拡張バイナリログがオンになっていても、binlog_checksum は動的パラメータとして動作しません。このパラメータへの変更を有効

にするには、ApplyMethod が immediate の場合にも DB クラスターを手動で再起動する必要があります。

- binlog_order_commits パラメータに設定した値は、拡張バイナリログがオンになっているときのコミットの順序には影響しません。コミットは常に順序付けられ、それ以上パフォーマンスへの影響はありません。

拡張バイナリログとコミュニティ MySQL バイナリログの違い

拡張バイナリログは、コミュニティ MySQL バイナリログと比較したとき、クローン、バックアップ、Aurora グローバルデータベースとのインタラクションが異なります。拡張バイナリログを使用する前に、次の違いを理解しておくことをお勧めします。

- ソース DB クラスターの拡張バイナリログファイルは、クローンされた DB クラスターでは使用できません。
- 拡張バイナリファイルは Aurora バックアップには含まれていません。そのため、DB クラスターに保持期間が設定されていても、DB クラスターを復元した後は、ソース DB クラスターの拡張バイナリログファイルを使用できません。
- Aurora グローバルデータベースで使用する場合、プライマリ DB クラスターの拡張バイナリログファイルはセカンダリリージョンの DB クラスターに複製されません。

例

次の例は、拡張バイナリログとコミュニティ MySQL バイナリログの違いを示しています。

復元またはクローンされた DB クラスター上

拡張バイナリログがオンになっている場合、復元またはクローンされた DB クラスターでは過去のバイナリログファイルを使用できません。復元またはクローン操作の後、バイナリログがオンになっている場合、新しい DB クラスターは 1 (mysql-bin-changelog.000001) から始まる独自のバイナリログファイルのシーケンスの書き込みを開始します。

復元またはクローン操作後に拡張バイナリログを有効にするには、復元またはクローンされた DB クラスターに必要な DB クラスターパラメーターを設定します。詳細については、「拡張バイナリログパラメータの設定」を参照してください。

Example 拡張バイナリログがオンになっているときに実行されるクローンまたは復元操作

ソース DB クラスター:

```
mysql> show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        |
| mysql-bin-changelog.000003 |       156 | No        |
| mysql-bin-changelog.000004 |       156 | No        | --> Enhanced Binlog turned on
| mysql-bin-changelog.000005 |       156 | No        | --> Enhanced Binlog turned on
| mysql-bin-changelog.000006 |       156 | No        | --> Enhanced Binlog turned on
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

復元またはクローンされた DB クラスターでは、拡張バイナリログがオンになっていても、バイナリログファイルはバックアップされません。バイナリログデータの不連続性を避けるため、拡張バイナリログを有効にする前に書き込まれたバイナリログファイルも使用できません。

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        | --> New sequence of Binlog files
+----------------------------+-----------+-----------+
1 row in set (0.00 sec)
```

Example 拡張バイナリログがオフになっているときに実行されるクローンまたは復元操作

ソース DB クラスター:

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        | --> Enhanced Binlog enabled
```

```
| mysql-bin-changelog.000003 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

復元またはクローンされた DB クラスターでは、拡張バイナリログをオフにした後に書き込まれたバイナリログファイルを使用できます。

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
1 row in set (0.00 sec)
```

Amazon Aurora Global Database で

Amazon Aurora Global Database では、プライマリ DB クラスターのバイナリログデータはセカンダリ DB クラスターに複製レプリケートされません。クロスリージョンフェイルオーバープロセスの後、新たに昇格されたプライマリ DB クラスターでバイナリログデータを使用できなくなります。バイナリログがオンになっている場合、新たに昇格された DB クラスターは 1 (mysql-bin-changelog.000001) から始まる独自のバイナリログファイルのシーケンスを開始します。

フェイルオーバー後に拡張バイナリログを有効にするには、セカンダリ DB クラスターで必要な DB クラスターパラメータを設定する必要があります。詳細については、「拡張バイナリログパラメータの設定」を参照してください。

Example 拡張バイナリログがオンになっている場合、グローバルデータベースのフェイルオーバー操作が実行されます。

古いプライマリ DB クラスター (フェイルオーバー前):

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        |
| mysql-bin-changelog.000003 |       156 | No        |
| mysql-bin-changelog.000004 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000005 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000006 |       156 | No        | --> Enhanced Binlog enabled
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

新しいプライマリ DB クラスター (フェイルオーバー後):

拡張バイナリログがオンになっている場合、バイナリログファイルはセカンダリリージョンに複製されません。バイナリログデータの不連続性を避けるため、拡張バイナリログを有効にする前に書き込まれたバイナリログファイルは使用できません。

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        | --> Fresh sequence of Binlog
 files
+----------------------------+-----------+-----------+
1 row in set (0.00 sec)
```

Example 拡張バイナリログがオフになっている場合、グローバルデータベースのフェイルオーバー操作が実行されます。

ソース DB クラスター:

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
```

```
| Log_name                    | File_size | Encrypted |
+-----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000003 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+-----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

復元またはクローンされた DB クラスター:

拡張バイナリログをオフにした後に書き込まれたバイナリログファイルは複製され、新たに格された DB クラスターで使用できます。

```
mysql>show binary logs;

+-----------------------------+-----------+-----------+
| Log_name                    | File_size | Encrypted |
+-----------------------------+-----------+-----------+
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+-----------------------------+-----------+-----------+
3 rows in set (0.00 sec)
```

拡張バイナリログの Amazon CloudWatch メトリクス

以下の Amazon CloudWatch メトリクスは、拡張バイナリログがオンになっている場合にのみ公開されます。

CloudWatch メトリクス	説明	単位
ChangeLogBytesUsed	拡張バイナリログで使用されるストレージ容量。	バイト

CloudWatch メトリクス	説明	単位
ChangeLogReadIOPs	5 分以内の、拡張バイナリログで実行される読み取り I/O オペレーションの数。	5 分あたりのカウント
ChangeLogWriteIOPs	5 分以内の、拡張バイナリログで実行される書き込みディスク I/O オペレーションの数。	5 分あたりのカウント

拡張バイナリログの制限

拡張バイナリログがオンになっている場合、Amazon Aurora DB クラスターには次の制限が適用されます。

- 拡張バイナリログは Aurora MySQL 3.03.1 バージョン以降でのみサポートされています。
- プライマリ DB クラスターに書き込まれた拡張バイナリログファイルは、クローンまたは復元された DB クラスターにはコピーされません。
- Amazon Aurora Global Database で使用する場合、プライマリ DB クラスターの拡張バイナリログファイルはセカンダリ DB クラスターに複製されません。そのため、フェイルオーバープロセス後、過去のバイナリログデータは新しいプライマリ DB クラスターで使用できなくなります。
- 以下のバイナリログ設定パラメータは無視されます。
  - `binlog_group_commit_sync_delay`
  - `binlog_group_commit_sync_no_delay_count`
  - `binlog_max_flush_queue_time`
- データベース内の破損したテーブルを削除したり、名前を変更したりすることはできません。これらのテーブルを削除するには、AWS Support にお問い合わせください。
- 拡張バイナリログがオンになっている場合、バイナリログ I/O キャッシュは無効になります。詳細については、「バイナリログのレプリケーションの最適化」を参照してください。

> **Note**
>
> 拡張バイナリログでは、バイナリログ I/O キャッシュと同様の読み取りパフォーマンスと、向上した書き込みパフォーマンスが提供されます。

- バックトラック機能はサポートされていません。次の条件では、拡張バイナリログを DB クラスターで有効にできません。
  - バックトラック機能が現在有効になっている DB クラスター。
  - バックトラック機能が以前に有効になっていたが、現在は無効化されていない DB クラスター。
  - ソース DB クラスターまたはバックトラック機能が有効になっているスナップショットから復元された DB クラスター。

GTID ベースレプリケーションを使用する

以下では、Aurora MySQL クラスターと外部ソースとの間において、バイナリログ (binlog) レプリケーションでグローバルトランザクション ID (GTID) を使用する方法について説明します。

> **Note**
>
> Aurora では、この機能は、外部 MySQL データベースとの間で binlog レプリケーションを使用する Aurora MySQL クラスターでのみ使用できます。もう一方のデータベースは、Amazon RDS MySQL インスタンス、オンプレミス MySQL データベース、または別の AWS リージョン にある Aurora DB クラスターです。その種類のレプリケーションを設定する方法については、「Aurora と MySQL との間、または Aurora と別の Aurora DB クラスターとの間のレプリケーション (バイナリログレプリケーション)」を参照してください。

binlog レプリケーションを使用する際に MySQL での GTID ベースのレプリケーションに慣れていない場合は、MySQL ドキュメントの「Replication with global transaction identifiers」を参照してください。

GTID ベースのレプリケーションは、Aurora MySQL バージョン 2 および 3 でサポートされています。

トピック

- グローバルトランザクション ID (GTID) の概要
- GTID ベースレプリケーションのパラメータ
- Aurora MySQL クラスターの GTID ベースレプリケーションの設定
- Aurora MySQL DB クラスターの GTID ベースレプリケーションを無効にする

グローバルトランザクション ID (GTID) の概要

グローバルトランザクション ID (GTID) はコミットされた MySQL トランザクションに対して生成される一意の ID です。GTID を使用することで、簡単に binlog をレプリケーションおよびトラブルシューティングできるようになります。

> **Note**
>
> Aurora がクラスター内の DB インスタンス間でデータを同期する場合、そのレプリケーションメカニズムにバイナリログ (binlog) は含まれません。Aurora MySQL では、GTID ベースのレプリケーションは、binlog レプリケーションも使用して外部の MySQL 互換データベースから Aurora MySQL DB クラスター間でレプリケートする場合にのみ適用されます。

MySQL では、binlog レプリケーションに 2 種類のトランザクションを使用します。

- GTID トランザクション - GTID によって識別されるトランザクション。
- 匿名トランザクション - GTID が割り当てられていないトランザクション。

レプリケーション設定では、GTID はすべての DB インスタンスで一意です。GTID を使用すると、ログファイルの位置を参照する必要がないため、GTID はレプリケーション設定を簡素化します。GTID はまた、レプリケートされたトランザクションを追跡し、出典インスタンスとレプリカが一致しているかどうかの判断を容易にします。

外部の MySQL 互換データベースから Aurora クラスターにレプリケートする場合は通常、GTID ベースのレプリケーションを Aurora と共に使用します。このレプリケーション設定は、オンプレミスまたは Amazon RDS データベースから Aurora MySQL への移行の一環として行うことができます。外部データベースで既に GTID が使用されている場合に、Aurora クラスターに対して GTID ベースのレプリケーションを有効にすると、レプリケーションプロセスが簡単になります。

Aurora MySQL クラスター用に GTID ベースのレプリケーションを設定するには、まず DB クラスターパラメータグループの関連設定パラメータを設定します。その後、そのパラメータグループとクラスターを関連付けます。

GTID ベースレプリケーションのパラメータ

以下のパラメータを使用して、GTID ベースレプリケーションを設定します。

Parameter	有効な値	説明
gtid_mode	OFF, OFF_PERMISSIVE , ON_PERMISSIVE , ON	OFF は新しいトランザクションが匿名トランザクション (つまり GTID を持たない) であることを指定し、トランザクションは匿名でレプリケートされる必要があります。 OFF_PERMISSIVE は新しいトランザクションが匿名トランザクションであることを指定しますが、すべてのトランザクションをレプリケートできます。 ON_PERMISSIVE は新しいトランザクションが GTID トランザクションであることを指定しますが、すべてのトランザクションをレプリケートできます。 ON は新しいトランザクションが GTID トランザクションであることを指定し、トランザクションは複製される GTID トランザクションでなければなりません。
enforce_gtid_consistency	OFF, ON, WARN	OFF はトランザクションが GTID の整合性に違反することを許可します。 ON はトランザクションが GTID の整合性に違反することを防ぎます。 WARN は、トランザクションが GTID の整合性に違反することを許可しますが、違反が発生すると警告を生成します。

> **Note**
>
> AWS Management Console では、gtid_mode パラメータは gtid-mode のように表示されます。

GTID ベースのレプリケーションでは、Aurora MySQL DB クラスターの DB クラスターパラメータグループでこれらの設定を使用します。

- ON と ON_PERMISSIVE は、Aurora MySQL クラスターからの送信レプリケーションにのみ適用されます。いずれの値でも、Aurora DB クラスターは、外部データベースにレプリケートされるトランザクションに GTID を使用します。ON の場合は、外部データベースも GTID ベースのレプリケーションを使用する必要があります。ON_PERMISSIVE の場合、GTID ベースのレプリケーションは、外部データベースでオプションになります。
- OFF_PERMISSIVE が設定された場合、これは、Aurora DB クラスターが、外部データベースからの受信レプリケーションを受け入れることができることを意味します。これは、外部データベースで GTID ベースのレプリケーションが使用されているかどうかにかかわらず実行することができます。
- OFF が設定された場合、これは、Aurora DB クラスターが、GTID ベースのレプリケーションを使用しない外部データベースからの受信レプリケーションのみを受け入れることができることを意味します。

> **Tip**
>
> 受信レプリケーションは、Aurora MySQL クラスターの最も一般的な binlog レプリケーションのシナリオです。受信レプリケーションでは、GTID モードを OFF_PERMISSIVE に設定することをお勧めします。この設定では、レプリケーション出典の GTID 設定に関係なく、外部データベースからの受信レプリケーションが可能になります。

パラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。

Aurora MySQL クラスターの GTID ベースレプリケーションの設定

GTID ベースのレプリケーションが Aurora MySQL DB クラスターで有効になっている場合、GTID 設定は、インバウンドとアウトバウンドの binlog レプリケーションのいずれにも適用されます。

Aurora MySQL クラスターの GTID ベースのレプリケーションを有効にするには

1. DB クラスターパラメータグループを作成または編集するには、以下のパラメータ設定を使用します。

   - gtid_mode - ON または ON_PERMISSIVE

- `enforce_gtid_consistency` – `ON`

2. DB クラスターパラメータグループを Aurora MySQL クラスターに関連付けます。そのためには、「「パラメータグループを使用する」」の手順に従います。
3. (オプション) GTID を含まないトランザクションに GTID を割り当てる方法を指定します。これを行うには、mysql.rds_assign_gtids_to_anonymous_transactions (Aurora MySQL バージョン 3) でストアドプロシージャを呼び出します。

Aurora MySQL DB クラスターの GTID ベースレプリケーションを無効にする

Aurora MySQL DB クラスターに対する GTID ベースのレプリケーションは、無効にすることができます。これを行うと、Aurora クラスターは、GTID ベースのレプリケーションを使用する外部データベースとのインバウンドまたはアウトバウンドの binlog レプリケーションを実行できなくなります。

> **Note**
>
> 次の手順で、リードレプリカは、外部データベースとの間で binlog レプリケーションを伴う Aurora 設定のレプリケーションターゲットを意味します。読み取り専用の Aurora レプリカ DB インスタンスを意味するものではありません。例えば、Aurora クラスターが外部出典からの受信レプリケーションを受け入れる場合、Aurora プライマリインスタンスは binlog レプリケーションのリードレプリカとして機能します。

このセクションで示されているストアドプロシージャの詳細については、「Aurora MySQL ストアドプロシージャ」を参照してください。

Aurora MySQL DB クラスターの GTID ベースレプリケーションを無効にするには

1. Aurora レプリカで、次の手順を実行します。

   バージョン 3 の場合

   ```
   CALL mysql.rds_set_source_auto_position(0);
   ```

   バージョン 2 の場合

   ```
   CALL mysql.rds_set_master_auto_position(0);
   ```

2. gtid_mode を ON_PERMISSIVE にリセットします。

   a. Aurora MySQL クラスターに関連付けられている DB クラスターパラメータグループで、gtid_mode が ON_PERMISSIVE に設定されていることを確認します。

      パラメータグループを使用して設定パラメータの設定の詳細については、「「パラメータグループを使用する」」を参照してください。

   b. Aurora MySQL DB クラスターを再起動します。

3. gtid_mode を OFF_PERMISSIVE にリセットします。

   a. Aurora MySQL クラスターに関連付けられている DB クラスターパラメータグループで、gtid_mode が OFF_PERMISSIVE に設定されていることを確認します。

   b. Aurora MySQL DB クラスターを再起動します。

4. すべての GTID トランザクションが Aurora プライマリインスタンスに適用されるまで待ちます。適用されたことを確認するには、次の手順を実行します。

   a. Aurora プライマリインスタンスで、SHOW MASTER STATUS コマンドを実行します。

      出力は、次のようになります。

```
File                          Position
-----------------------------------
mysql-bin-changelog.000031      107
-----------------------------------
```

      出力のファイルと位置に注意してください。

   b. リードレプリカごとに、前のステップで得たソースインスタンスのファイルと位置の情報を使用して、次のクエリを実行します。

      バージョン 3 の場合

```
SELECT SOURCE_POS_WAIT('file', position);
```

      バージョン 2 の場合

```
SELECT MASTER_POS_WAIT('file', position);
```

例えば、ファイル名が mysql-bin-changelog.000031 で、場所が 107 である場合は、次のステートメントを実行します。

バージョン 3 の場合

```
SELECT SOURCE_POS_WAIT('mysql-bin-changelog.000031', 107);
```

バージョン 2 の場合

```
SELECT MASTER_POS_WAIT('mysql-bin-changelog.000031', 107);
```

5. GTID パラメータをリセットして、GTID ベースのレプリケーションを無効にします。

   a. Aurora MySQL クラスターに関連付けられた DB クラスターパラメータグループに次のパラメータが設定されていることを確認します。

      - gtid_mode – OFF
      - enforce_gtid_consistency – OFF

   b. Aurora MySQL DB クラスターを再起動します。

Amazon Aurora MySQL と他の AWS のサービスの統合

Amazon Aurora MySQL を他の AWS のサービスと統合することで、Aurora MySQL DB クラスターを拡張して AWS クラウドの追加機能を使用できるようになります。Aurora MySQL DB クラスターでは、AWS のサービスを使用して以下のことができます。

- ネイティブ関数 AWS Lambda または `lambda_sync` を使用して、`lambda_async` 関数を同期または非同期に呼び出します。詳細については、「Amazon Aurora MySQL DB クラスターからの Lambda 関数の呼び出し」を参照してください。
- `LOAD DATA FROM S3` コマンドまたは `LOAD XML FROM S3` コマンドを使用して、Amazon Simple Storage Service (Amazon S3) バケットに格納されているテキストファイルや XML ファイルのデータを DB クラスター内にロードする。詳細については、「Amazon S3 バケットのテキストファイルから Amazon Aurora MySQL DB クラスターへのデータのロード」を参照してください。
- `SELECT INTO OUTFILE S3` コマンドを使用して DB クラスターから Amazon S3 バケット内のテキストファイルにデータを保存する。詳細については、「Amazon Aurora MySQL DB クラスターから Amazon S3 バケット内のテキストファイルへのデータの保存」を参照してください。
- アプリケーションの Auto Scaling で Aurora レプリカを自動的に追加または削除します。詳細については、「Aurora レプリカでの Amazon Aurora Auto Scaling の使用」を参照してください。
- Amazon Comprehend で感情分析を実行するか、SageMaker で多種多様な機械学習アルゴリズムを実行する。詳細については、「Amazon Aurora 機械学習の使用」を参照してください。

Aurora は、AWS Identity and Access Management (IAM) を使用して他の AWS のサービスに確実にアクセスできるようにします。他の AWS のサービスにアクセスする権限を付与するには、必要なアクセス許可を持つ IAM ロールを作成し、そのロールを DB クラスターに関連付けます。Aurora MySQL DB クラスターから他の AWS のサービスへのアクセスを許可する方法の詳細と手順については、「ユーザーの代わりに Amazon Aurora MySQL が他の AWS のサービスにアクセスすることを許可する」を参照してください。

ユーザーの代わりに Amazon Aurora MySQL が他の AWS のサービスにアクセスすることを許可する

Aurora MySQL DB クラスターに、ユーザーに代わって他のサービスにアクセスさせるには、AWS Identity and Access Management (IAM) ロールを作成および設定します。このロールは、DB クラスターのデータベースユーザーが他の AWS のサービスにアクセスする権限を付与します。詳細については、「AWS のサービスにアクセスするための IAM ロールの設定」を参照してください。

ターゲット AWS サービスへのアウトバウンド接続を許可するように、Aurora DB クラスターを設定する必要もあります。詳細については、「Amazon Aurora MySQL から他の AWS のサービスへのネットワーク通信の有効化」を参照してください。

これにより、データベースユーザーは AWS の他のサービスを使用して以下のアクションを実行できるようになります。

- ネイティブ関数 AWS Lambda または `lambda_sync` を使用して、`lambda_async` 関数を同期または非同期に呼び出します。または、AWS Lambda プロシージャを使用して `mysql.lambda_async` 関数を非同期に呼び出します。詳細については、「Aurora MySQL ネイティブ関数を使用した Lambda 関数の呼び出し」を参照してください。
- `LOAD DATA FROM S3` ステートメントまたは `LOAD XML FROM S3` ステートメントを使用して、Amazon S3 バケットに保存されているテキストファイルや XML ファイルのデータを DB クラスター内にロードする。詳細については、「Amazon S3 バケットのテキストファイルから Amazon Aurora MySQL DB クラスターへのデータのロード」を参照してください。
- `SELECT INTO OUTFILE S3` ステートメントを使用して、DB クラスターのデータを Amazon S3 バケットに保存されているテキストファイル内に保存する。詳細については、「Amazon Aurora MySQL DB クラスターから Amazon S3 バケット内のテキストファイルへのデータの保存」を参照してください。
- ログデータを Amazon CloudWatch Logs MySQL にエクスポートします。詳細については、「Amazon CloudWatch Logs への Amazon Aurora MySQL ログの発行」を参照してください。
- アプリケーションの Auto Scaling で Aurora レプリカを自動的に追加または削除します。詳細については、「Aurora レプリカでの Amazon Aurora Auto Scaling の使用」を参照してください。

AWS のサービスにアクセスするための IAM ロールの設定

Aurora DB クラスターから別の AWS サービスにアクセスすることを許可するには、以下のことを行います。

1. AWS のサービスにアクセス権限を付与する IAM ポリシーを作成します。詳細については、以下を参照してください。
   - Amazon S3 リソースにアクセスするための IAM ポリシーの作成
   - AWS Lambda リソースにアクセスするための IAM ポリシーの作成
   - CloudWatch Logs リソースにアクセスするための IAM ポリシーの作成
   - AWS KMS リソースにアクセスするための IAM ポリシーの作成

2. IAM ロールを作成し、作成したポリシーをアタッチします。詳細については、「Amazon Aurora が AWS のサービスにアクセスすることを許可する IAM ロールの作成」を参照してください。
3. その IAM ロールを Aurora DB クラスターに関連付けます。詳細については、「IAM ロールと Amazon Aurora MySQL DB クラスターの関連付け」を参照してください。

Amazon S3 リソースにアクセスするための IAM ポリシーの作成

Aurora では、Amazon S3 のリソースにアクセスして Aurora DB クラスターにデータをロードしたり、Aurora DB クラスターのデータを保存したりできます。ただし、初期に IAM ポリシーを作成してバケットおよびオブジェクトのアクセス許可を付与し、Aurora から Amazon S3 にアクセスできるようにする必要があります。

次の表は、ユーザー名で Amazon S3 バケットにアクセスできる Aurora の機能と、各機能に必要な最小限のバケットとオブジェクトのアクセス許可の一覧です。

機能	Bucket permissions (バケットのアクセス許可)	オブジェクトのアクセス許可
`LOAD DATA FROM S3`	`ListBucket`	`GetObject` `GetObjectVersion`
`LOAD XML FROM S3`	`ListBucket`	`GetObject` `GetObjectVersion`

`SELECT INTO OUTFILE S3`	`ListBucket`	`AbortMultipartUpload`
		`DeleteObject`
		`GetObject`
		`ListMultipartUploadParts`
		`PutObject`

次のポリシーは、ユーザー名で Amazon S3 バケットにアクセスするために Aurora で必要となるアクセス許可を追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAuroraToExampleBucket",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:AbortMultipartUpload",
                "s3:ListBucket",
                "s3:DeleteObject",
                "s3:GetObjectVersion",
                "s3:ListMultipartUploadParts"
            ],
            "Resource": [
                "arn:aws:s3:::example-bucket/*",
                "arn:aws:s3:::example-bucket"
            ]
        }
    ]
}
```

> **Note**
>
> Resource 値の両方のエントリを含んでいることを確認します。Aurora には、バケット自体とバケット内のすべてのオブジェクトの両方に対するアクセス許可が必要です。
> ユースケースによっては、ポリシーのサンプルにすべてのアクセス許可を追加する必要がない場合があります。また、その他のアクセス許可が必要になる可能性もあります。例えば、Amazon S3 バケットが暗号化されている場合には、 kms:Decrypt アクセス許可を追加する必要があります。

以下のステップを使用して、ユーザーの代わりに Aurora から Amazon S3 バケットにアクセスするために必要な最低のアクセス権限を提供する IAM ポリシーを作成できます。Aurora からすべての Amazon S3 バケットへのアクセスを許可するには、以下のステップをスキップし、独自のポリシーを作成する代わりに定義済みの IAM ポリシーである AmazonS3ReadOnlyAccess または AmazonS3FullAccess を使用できます。

Amazon S3 リソースへのアクセスを許可する IAM ポリシーを作成するには

1. IAM マネジメントコンソールを開きます。
2. ナビゲーションペインで、[ポリシー] を選択します。
3. [ポリシーの作成] を選択します。
4. [Visual editor] タブで、[Choose a service] を選択し、[S3] を選択します。
5. [アクション] で [すべて展開] を選択してから、IAM ポリシーに必要なバケットへのアクセス許可とオブジェクトへのアクセス許可を選択します。

   オブジェクトへのアクセス許可は、Amazon S3 のオブジェクトオペレーションのアクセス許可であり、バケット自体ではなくバケット内のオブジェクトに付与する必要があります。Amazon S3 におけるオブジェクトオペレーションのアクセス許可の詳細については、「オブジェクトオペレーションに対するアクセス許可」を参照してください。
6. [リソース] を選択し、[バケット] に [ARN の追加] を選択します。
7. [ARN の追加] ダイアログボックスで、リソースの詳細を指定し、[追加] を選択します。

   アクセスを許可する Amazon S3 バケットを指定します。例えば、`example-bucket` という名前の Amazon S3 バケットへのアクセスを Aurora に許可するには、Amazon リソースネーム (ARN) の値を `arn:aws:s3:::example-bucket` に設定します。
8. [オブジェクト] リソースがリストされた場合は、[オブジェクト] に対して [ARN の追加] を選択します。
9. [Add ARN(s)] ダイアログボックスで、リソースの詳細を指定します。

   Amazon S3 バケットの場合は、アクセスを許可する Amazon S3 バケットを指定します。オブジェクトの場合は、[Any] を選択してバケット内の任意のオブジェクトにアクセス許可を付与できます。

   > **Note**
   >
   > Aurora から Amazon S3 バケット内の特定のファイルやフォルダにのみアクセスすることを許可するには、[Amazon リソースネーム (ARN)] により具体的な ARN 値を設定することができます。Amazon S3 のアクセスポリシーの定義方法については、「Amazon S3 リソースへのアクセス許可の管理」を参照してください。
10. (オプション) バケットの [ARN の追加] を選択して、別の Amazon S3 バケットをポリシーに追加し、そのバケットに対して前のステップを繰り返します。

> **Note**
>
> このステップを繰り返して、対応するバケットのアクセス許可ステートメントを、Aurora からアクセスする各 Amazon S3 バケットのポリシーに追加できます。オプションで、Amazon S3 内のすべてのバケットとオブジェクトへのアクセスを許可できます。

11. [ポリシーの確認] を選択します。
12. [名前] に、IAM ポリシーの名前 (例: `AllowAuroraToExampleBucket`) を設定します。IAM ロールを作成して Aurora DB クラスターに関連付ける際に、この名前を使用します。オプションで [Description] 値を追加することもできます。
13. [Create policy] を選択します。
14. 「Amazon Aurora が AWS のサービスにアクセスすることを許可する IAM ロールの作成」の各ステップを実行します。

AWS Lambda リソースにアクセスするための IAM ポリシーの作成

ユーザーの代わりに Aurora から AWS Lambda 関数を呼び出すために必要な最低のアクセス許可を付与する、IAM ポリシーを作成できます。

次のポリシーは、Aurora がユーザーに代わって AWS Lambda 関数を呼び出すために必要なアクセス許可を追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAuroraToExampleFunction",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource":
 "arn:aws:lambda:<region>:<123456789012>:function:<example_function>"
        }
    ]
}
```

以下のステップを使用して、ユーザーの代わりに Aurora から AWS Lambda 関数を呼び出すために必要な、最低限のアクセス許可を付与する IAM ポリシーを作成できます。Aurora からすべての

AWS Lambda 関数を呼び出すことを許可するには、以下のステップをスキップして、独自のポリシーを作成する代わりに定義済みの `AWSLambdaRole` ポリシーを使用できます。

AWS Lambda 関数への呼び出しを許可する IAM ポリシーを作成するには

1. IAM コンソールを開きます。
2. ナビゲーションペインで、[ポリシー] を選択します。
3. [ポリシーの作成] を選択します。
4. [ビジュアルエディタ] タブで、[サービスの選択] を選択し、[Lambda] を選択します。
5. [アクション] の [すべて展開] を選択し、IAM ポリシーに必要な AWS Lambda アクセス許可を選択します。

   `InvokeFunction` が選択されていることを確認します。これは、Amazon Aurora から AWS Lambda 関数を呼び出すために必要な最低限のアクセス許可です。
6. [リソース] を選択し、[関数] に対して [ARN の追加] を選択します。
7. [Add ARN(s)] ダイアログボックスで、リソースの詳細を指定します。

   アクセスを許可する Lambda 関数を指定します。例えば、Aurora から `example_function` という名前の Lambda 関数にアクセスすることを許可するには、ARN 値として `arn:aws:lambda:::function:example_function` を設定します。

   AWS Lambda のアクセスポリシーを定義する方法の詳細については、「AWS Lambda に対する認証とアクセス制御」を参照してください。
8. オプションで、[さらにアクセス許可を追加する] を選択して、ポリシーに別の AWS Lambda 関数を追加し、その関数に対して前のステップを繰り返します。

   > **Note**
   >
   > このステップを繰り返して、対応する関数のアクセス許可ステートメントを、Aurora からアクセスする各 AWS Lambda 関数のポリシーに追加できます。
9. [ポリシーの確認] を選択します。
10. [名前] に、IAM ポリシーの名前 (`AllowAuroraToExampleFunction` など) を設定します。IAM ロールを作成して Aurora DB クラスターに関連付ける際に、この名前を使用します。オプションで [Description] 値を追加することもできます。
11. [Create policy] を選択します。

12. 「Amazon Aurora が AWS のサービスにアクセスすることを許可する IAM ロールの作成」の各ステップを実行します。

CloudWatch Logs リソースにアクセスするための IAM ポリシーの作成

Aurora は CloudWatch Logs にアクセスして Aurora DB クラスターから監査ログデータをエクスポートできます。ただし、初期に IAM ポリシーを作成してロググループおよびログストリーミングのアクセス許可を付与し、Aurora から CloudWatch Logs にアクセスできるようにする必要があります。

以下のポリシーは、ユーザー名で Amazon CloudWatch Logs にアクセスするために Aurora が要求する権限および、ロググループを作成してデータをエクスポートするための最小限の権限を追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnableCreationAndManagementOfRDSCloudwatchLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:GetLogEvents",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/rds/*:log-stream:*"
        },
        {
            "Sid": "EnableCreationAndManagementOfRDSCloudwatchLogGroupsAndStreams",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutRetentionPolicy",
                "logs:CreateLogGroup"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/rds/*"
        }
    ]
}
```

ポリシーの ARN を変更すると、特定の AWS リージョンおよびアカウントへのアクセスを制限できます。

以下のステップを使用して、ユーザーの代わりに Aurora から CloudWatch Logs にアクセスするために必要な最低のアクセス権限を提供する IAM ポリシーを作成できます。Aurora に CloudWatch Logs へのフルアクセスを付与するには、このステップをスキップして、独自のポリシーを作成する代わりに、定義済みの `CloudWatchLogsFullAccess` IAM ポリシーを使用します。詳細については、Amazon CloudWatch ユーザーガイドの「CloudWatch Logs でアイデンティティベースのポリシー (IAM ポリシー) を使用する」を参照してください。

CloudWatch Logs リソースへのアクセスを許可する IAM ポリシーを作成するには

1. IAM コンソールを開きます。
2. ナビゲーションペインで、[ポリシー] を選択します。
3. [ポリシーの作成] を選択します。
4. [ビジュアルエディタ] タブで、[サービスの選択] を選択し、[CloudWatch Logs] を選択します。
5. [アクション] で、右側にある [すべて展開] を選択し、IAM ポリシーに必要な Amazon CloudWatch Logs アクセス許可を選択します。

   次のアクセス許可が選択されていることを確認します。

   - `CreateLogGroup`
   - `CreateLogStream`
   - `DescribeLogStreams`
   - `GetLogEvents`
   - `PutLogEvents`
   - `PutRetentionPolicy`
6. [リソース] を選択し、[log-group] に対して [ARN の追加] を選択します。
7. [ARN の追加] ダイアログボックスで、以下の値を入力します。

   - リージョン - AWS リージョンまたは *
   - アカウント - アカウント番号または *
   - ロググループ名 - `/aws/rds/*`
8. [ARN の追加] ダイアログボックスで、[追加] を選択します。
9. [log-stream] に [ARN の追加] を選択します。
10. [ARN の追加] ダイアログボックスで、以下の値を入力します。

    - リージョン - AWS リージョンまたは *

- アカウント - アカウント番号または *
- ロググループ名]/aws/rds/* -
- ログストリーミング名 - *

11. [ARN の追加] ダイアログボックスで、[追加] を選択します。
12. [ポリシーの確認] を選択します。
13. [名前] に、IAM ポリシーの名前 (AmazonRDSCloudWatchLogs など) を設定します。IAM ロールを作成して Aurora DB クラスターに関連付ける際に、この名前を使用します。オプションで [Description] 値を追加することもできます。
14. [Create policy] を選択します。
15. 「Amazon Aurora が AWS のサービスにアクセスすることを許可する IAM ロールの作成」の各ステップを実行します。

AWS KMS リソースにアクセスするための IAM ポリシーの作成

Aurora は、データベースバックアップの暗号化に使用された AWS KMS keys にアクセスできます。ただし、初期に IAM ポリシーを作成してアクセス許可を付与し、Aurora が KMS キーにアクセスできるようにする必要があります。

次のポリシーでは、ユーザーの代わりに KMS キーにアクセスするために Aurora で必要となるアクセス許可が追加されています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:<region>:<123456789012>:key/<key-ID>"
        }
    ]
}
```

次のステップを使用して、Aurora がユーザーの代わりに KMS キーにアクセスするために必要な最小限のアクセス許可を付与する IAM ポリシーを作成できます。

KMS キーへのアクセスを許可する IAM ポリシーを作成するには

1. [IAM コンソール] を開きます。
2. ナビゲーションペインで、[ポリシー] を選択します。
3. [Create policy] (ポリシーの作成) を選択します。
4. [ビジュアルエディタ] タブで、[サービスの選択] を選択し、[KMS] を選択します。
5. [アクション] で、[書き込み]、[復号] の順に選択します。
6. [リソース]、[ARN の追加] の順に選択します。
7. [ARN の追加] ダイアログボックスで、以下の値を入力します。
   - [リージョン] - AWS リージョンを (us-west-2 のように) 入力します。
   - [アカウント] - ユーザーアカウント番号を入力します。
   - [ログストリーミング名] - KMS キー識別子を入力します。
8. [ARN の追加] ダイアログボックスで、[追加] を選択します。
9. [ポリシーの確認] を選択します。
10. [名前] に、IAM ポリシーの名前 (AmazonRDSKMSKey など) を設定します。IAM ロールを作成して Aurora DB クラスターに関連付ける際に、この名前を使用します。オプションで [Description] 値を追加することもできます。
11. [Create policy] (ポリシーを作成) を選択します。
12. 「Amazon Aurora が AWS のサービスにアクセスすることを許可する IAM ロールの作成」の各ステップを実行します。

Amazon Aurora が AWS のサービスにアクセスすることを許可する IAM ロールの作成

Aurora から AWS リソースへのアクセスを許可する IAM ポリシーを作成したら、次に IAM ロールを作成して、この新しい IAM ロールに IAM ポリシーをアタッチする必要があります。

ユーザーに代わって Amazon RDS クラスターが他の AWS のサービスと通信することを許可する IAM ロールを作成するには、以下のステップを実行します。

Amazon RDS が AWS のサービスにアクセスすることを許可する IAM ロールを作成するには

1. IAM コンソールを開きます。
2. ナビゲーションペインで [Roles (ロール)] を選択します。
3. [Create role] を選択します。

4. [AWS のサービス] で [RDS] を選択します。
5. [ユースケースの選択] で [RDS - データベースへのロールの追加] を選択します。
6. [Next] を選択します。
7. [アクセス許可ポリシーのアタッチ] ページで、[検索] フィールドにポリシーの名前を入力します。
8. ポリシーがリストに表示されたら、次のセクションの説明のいずれかを使用して前に定義したポリシーを選択します。
   - Amazon S3 リソースにアクセスするための IAM ポリシーの作成
   - AWS Lambda リソースにアクセスするための IAM ポリシーの作成
   - CloudWatch Logs リソースにアクセスするための IAM ポリシーの作成
   - AWS KMS リソースにアクセスするための IAM ポリシーの作成
9. [Next] を選択します。
10. [ロール名] に、IAM ロールの名前 (`RDSLoadFromS3` など) を入力します。オプションで [Description] 値を追加することもできます。
11. [Create Role] を選択します。
12. 「IAM ロールと Amazon Aurora MySQL DB クラスターの関連付け」の各ステップを実行します。

IAM ロールと Amazon Aurora MySQL DB クラスターの関連付け

Amazon Aurora DB クラスター内のデータベースユーザーから他の AWS のサービスにアクセスすることを許可するには、Amazon Aurora が AWS のサービスにアクセスすることを許可する IAM ロールの作成 で作成した IAM ロールを、その DB クラスターに関連付けます。サービスを直接関連付けることで、AWS に新しい IAM ロールを作成させることもできます。

> **Note**
>
> IAM ロールを Aurora Serverless v1 DB クラスターに関連付けることはできません。詳細については、「Amazon Aurora Serverless v1 の使用」を参照してください。
> IAM ロールを Aurora Serverless v2 DB クラスターに関連付けることはでます。

IAM ロールを DB クラスターに関連付けるには、2 つのことを行います。

1. RDS コンソール、add-role-to-db-cluster AWS CLI コマンド、または AddRoleToDBCluster RDS API オペレーションを使用して、DB クラスターの関連付けられたロールのリストにロールを追加します。

   Aurora DB クラスターごとに最大 5 つの IAM ロールを追加できます。

2. 関連する AWS のサービスのクラスターレベルのパラメータを、関連付けられた IAM ロールの ARN に設定します。

   次の表では、AWS の他のサービスにアクセスするために使用する IAM ロールのクラスターレベルのパラメータ名について説明します。

クラスターレベルのパラメータ	説明	
aws_default_lambda_role	DB クラスターから Lambda 関数を呼び出すときに使用します。	
aws_default_logs_role	DB クラスターからのログデータを Amazon CloudWatch Logs にエクスポートするために、このパラメータを使用する必要はなくなりました。Aurora MySQL は、必要なアクセス許可のためにサービスにリンクされたロールを使用するようになりました。サービスにリンクされたロールの詳細については、「Amazon Aurora のサービスにリンクされたロールの使用」を参照してください。	
aws_default_s3_role	DB クラスターから LOAD DATA FROM S3 ステートメント、LOAD XML FROM S3 ステートメント、または SELECT INTO OUTFILE S3 ステートメントを呼び出すときに使用します。 Aurora MySQL バージョン 2 では、該当するステートメントの aurora_load_from_s3_role または aurora_select_into_s3_role に IAM ロールが指定されていな	

クラスターレベルのパラメータ	説明	
	い場合に、このパラメータで指定された IAM ロールが使用されます。 Aurora MySQL バージョン 3 では、このパラメータに指定した IAM ロールが常に使用されます。	
aurora_load_from_s3_role	DB クラスターから LOAD DATA FROM S3 ステートメントまたは LOAD XML FROM S3 ステートメントを呼び出すときに使用します。このパラメータに IAM ロールが指定されていない場合は、aws_default_s3_role に指定した IAM ロールが使用されます。 Aurora MySQL バージョン 3 では、このパラメータは使用できません。	
aurora_select_into_s3_role	DB クラスターから SELECT INTO OUTFILE S3 ステートメントを呼び出すときに使用します。このパラメータに IAM ロールが指定されていない場合は、aws_default_s3_role に指定した IAM ロールが使用されます。 Aurora MySQL バージョン 3 では、このパラメータは使用できません。	

ユーザーに代わって Amazon RDS クラスターが他の AWS のサービスと通信することを許可する IAM ロールを関連付けるには、以下のステップを実行します。

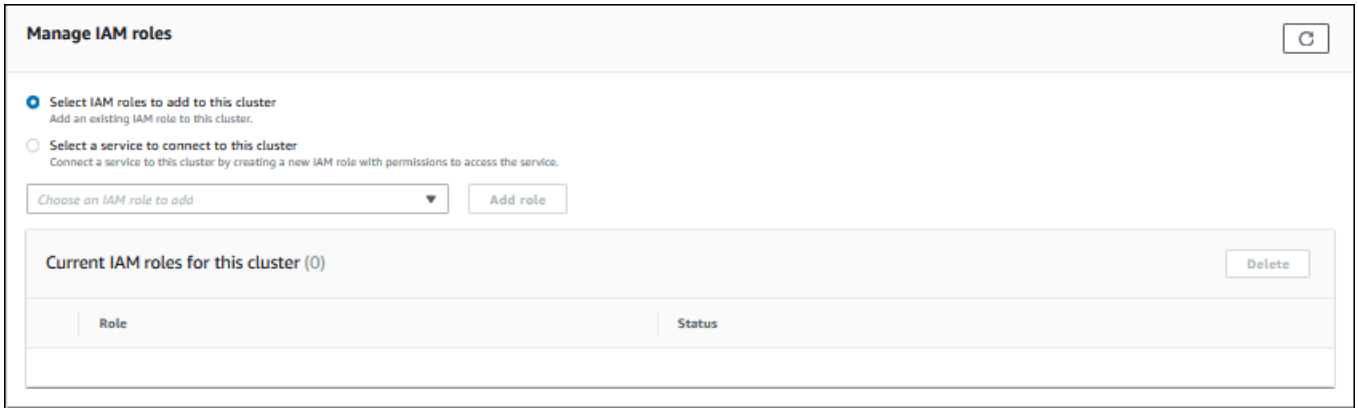
コンソール

コンソールを使用して IAM ロールを Aurora DB クラスターに関連付けるには

1. RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. [データベース] をクリックします。

3. IAM ロールを関連付ける Aurora DB クラスターの名前を選択して、詳細を表示します。
4. [Connectivity & security] (接続とセキュリティ) タブの [Manage IAM roles] (IAM ロールの管理) セクションで、次のいずれかを実行します。

   - このクラスターに追加する IAM ロールを選択してください (デフォルト)
   - このクラスターに接続するサービスを選択してください



5. 既存の IAM ロールを使用するには、メニューからロールを選択し、[Add role] (ロールの追加) を選択します。

   ロールの追加に成功すると、そのステータスは `Pending`、`Available` の順に表示されます。
6. サービスに直接接続するには、以下を実行します。

   a. [Select a service to connect to this cluster] (このクラスターに接続するサービスを選択する) を選択します。

   b. メニューからサービスを選択し、[Connect service] (サービスに接続する) を選択します。

   c. [Connect cluster to ***`Service Name`***] (クラスターをサービス名に接続する) で、サービスへの接続に使用する Amazon リソースネーム (ARN) を入力し、[Connect service] (サービスに接続する) を選択します。

   AWS は、サービスに接続するための新しい IAM ロールを作成します。そのステータスは `Pending`、次に `Available` と表示されます。
7. (オプション) DB クラスターへの IAM ロールの関連付けを中止し、関連するアクセス許可を削除するには、ロールの [Delete] (削除) を選択します。

関連する IAM ロールにクラスターレベルのパラメータを設定するには

1. RDS コンソールで、ナビゲーションペインの [パラメータグループ] を選択します。
2. カスタム DB パラメータグループをすでに使用している場合は、DB クラスターの新しいパラメータグループを作成する代わりに、そのグループを選択して使用できます。DB クラスターのデフォルトのパラメータグループを使用している場合は、以下のステップに従って、DB クラスターの新しいパラメータグループを作成します。

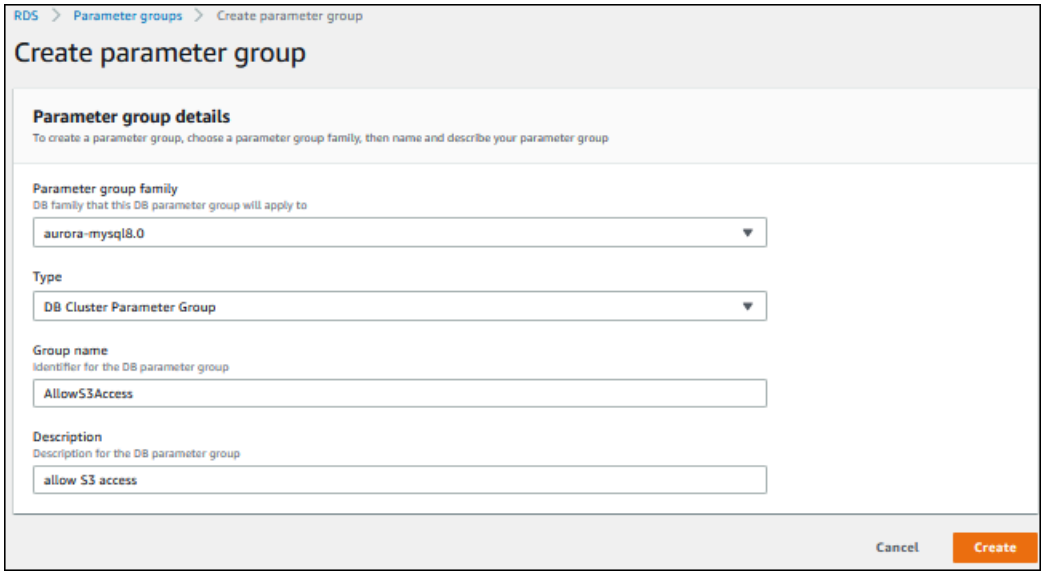
   a. [パラメータグループの作成]を選択します。

   b. [パラメータグループファミリー] で、Aurora MySQL 8.0 互換の DB クラスターには `aurora-mysql8.0` を選択し、Aurora MySQL 5.7 互換の DB クラスターには `aurora-mysql5.7` を選択します。

   c. [タイプ] で、[DB クラスターのパラメータグループ] を選択します。

   d. [グループ名] に、DB クラスターの新しいパラメータグループの名前を入力します。

   e. [説明] に、DB クラスターの新しいパラメータグループの説明を入力します。



   f. [作成] を選択します。

3. [パラメータグループ] ページで、DB クラスターパラメータグループを選択して [パラメータグループアクション]、[編集] の順に選択します。
4. 適切なクラスターレベルのパラメータを、関連する IAM ロールの ARN 値に設定します。

   例えば、`aws_default_s3_role` パラメータを `arn:aws:iam::123456789012:role/AllowS3Access` に設定します。

5. [変更の保存] をクリックします。

6. DB クラスターの DB クラスターパラメータグループを変更するには、次のステップをすべて行います。

   a. [データベース] を選択後、Aurora DB クラスターを選択します。

   b. [Modify] (変更) を選択します。

   c. [データベースの選択肢] までスクロールし、[DB クラスターのパラメータグループ] を、DB クラスターのパラメータグループに設定します。

   d. [続行] を選択します。

   e. 変更を確認し、[すぐに適用] を選択します。

   f. [クラスタークラスターの変更] を選択します。

   g. [データベース] を選択後、DB クラスターのプライマリインスタンスを選択します。

   h. [アクション] で、[再起動] を選択します。

      インスタンスが再起動すると、IAM ロールが DB クラスターに関連付けられます。

      クラスターのパラメータグループの詳細については、「Aurora MySQL 設定パラメータ」を参照してください。

CLI

AWS CLI を使用して IAM ロールを DB クラスターに関連付けるには

1. 以下に示すように、`add-role-to-db-cluster` から AWS CLI コマンドを呼び出して、IAM ロールの ARN を DB クラスターに追加します。

```
PROMPT> aws rds add-role-to-db-cluster --db-cluster-identifier my-cluster --role-
arn arn:aws:iam::123456789012:role/AllowAuroraS3Role
PROMPT> aws rds add-role-to-db-cluster --db-cluster-identifier my-cluster --role-
arn arn:aws:iam::123456789012:role/AllowAuroraLambdaRole
```

2. DB クラスターのデフォルトのパラメータグループを使用している場合は、DB クラスターの新しいパラメータグループを作成します。カスタム DB パラメータグループをすでに使用している場合は、DB クラスターの新しいパラメータグループを作成する代わりに、そのグループを使用できます。

   DB クラスターの新しいパラメータグループを作成するには、以下に示すように、`create-db-cluster-parameter-group` から AWS CLI コマンドを呼び出します。

```
PROMPT> aws rds create-db-cluster-parameter-group  --db-cluster-parameter-group-
name AllowAWSAccess \
     --db-parameter-group-family aurora5.7 --description "Allow access to Amazon S3
 and AWS Lambda"
```

Aurora MySQL 5.7 互換 DB クラスターの場合は、`aurora-mysql5.7` に `--db-parameter-group-family` を指定します。Aurora MySQL 8.0 互換 DB クラスターの場合は、`--db-parameter-group-family` に `aurora-mysql8.0` を指定します。

3. 以下に示すように、適切なクラスターレベルの単一あるいは複数のパラメータと関連する IAM ロールの ARN 値を DB クラスターのパラメータグループに設定します。

```
PROMPT> aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name
 AllowAWSAccess \
    --parameters
 "ParameterName=aws_default_s3_role,ParameterValue=arn:aws:iam::123456789012:role/
AllowAuroraS3Role,method=pending-reboot" \
    --parameters
 "ParameterName=aws_default_lambda_role,ParameterValue=arn:aws:iam::123456789012:role/
AllowAuroraLambdaRole,method=pending-reboot"
```

4. 以下に示すように、DB クラスターの新しいパラメータグループを使うように DB クラスターを変更し、クラスターを再起動します。

```
PROMPT> aws rds modify-db-cluster --db-cluster-identifier my-cluster --db-cluster-
parameter-group-name AllowAWSAccess
PROMPT> aws rds reboot-db-instance --db-instance-identifier my-cluster-primary
```

インスタンスが再起動すると、IAM ロールが DB クラスターに関連付けられています。

クラスターのパラメータグループの詳細については、「Aurora MySQL 設定パラメータ」を参照してください。

Amazon Aurora MySQL から他の AWS のサービスへのネットワーク通信の有効化

Amazon Aurora で、AWS の特定の他のサービスを使用するには、Aurora DB クラスターのネットワーク設定で、それらのサービスのエンドポイントへのアウトバウンド接続を許可する必要があります。次のオペレーションでは、このネットワーク設定が必要です。

- AWS Lambda 関数を呼び出す。この機能の詳細については、「Aurora MySQL ネイティブ関数を使用した Lambda 関数の呼び出し」を参照してください。
- Amazon S3 からファイルにアクセスする。この機能の詳細については、「Amazon S3 バケットのテキストファイルから Amazon Aurora MySQL DB クラスターへのデータのロード」および「Amazon Aurora MySQL DB クラスターから Amazon S3 バケット内のテキストファイルへのデータの保存」を参照してください。
- AWS KMS エンドポイントへのアクセス。AWS KMS へのアクセスは、Aurora MySQL でデータベースアクティビティストリーミングを使用するために必要です。この機能の詳細については、「データベースアクティビティストリームを使用した Amazon Aurora のモニタリング」を参照してください。
- SageMaker エンドポイントへのアクセス Aurora MySQL で SageMaker 機械学習を使用するには、SageMaker によるアクセスが必要です。この機能の詳細については、「Aurora MySQL で Amazon Aurora 機械学習を使用する」を参照してください。

サービスのエンドポイントに接続できない場合、Aurora は以下のエラーメッセージを返します。

```
ERROR 1871 (HY000): S3 API returned error: Network Connection
```

```
ERROR 1873 (HY000): Lambda API returned error: Network Connection. Unable to connect to
 endpoint
```

```
ERROR 1815 (HY000): Internal error: Unable to initialize S3Stream
```

Aurora MySQL を使用するデータベースアクティビティストリーミングでは、DB クラスターが AWS KMS エンドポイントにアクセスできない場合、アクティビティストリーミングは機能を停止します。Aurora は、RDS イベントを使用してこの問題について通知します。Aurora は RDS イベントを使用してこの問題について通知します。

対応する AWS のサービスの使用中にこれらのメッセージが表示された場合は、Aurora DB クラスターがパブリックかプライベートかを確認します。Aurora DB クラスターがプライベートの場合は、接続を有効にするように設定する必要があります。

Aurora DB クラスターがパブリックの場合は、パブリックアクセス可能とマークされている必要があります。この場合、AWS Management Console で DB クラスターの詳細を見ると、[パブリックアクセス可能] が [はい] となっています。DB クラスターは Amazon VPC パブリックサブネットにも存在する必要があります。パブリックアクセス可能な DB インスタンスの詳細については、「VPC 内の

DB クラスターの使用」を参照してください。Amazon VPC のパブリックサブネットの詳細については、「VPC とサブネット」を参照してください。

Aurora DB クラスターがパブリックアクセス可能ではなく、VPC パブリックサブネットにある場合は、プライベートです。プライベートの DB クラスターがあり、このネットワーク設定を必要とする機能の 1 つを使用する場合があります。その場合、ネットワークアドレス変換 (NAT) を介してインターネットアドレスに接続できるよう、クラスターを設定します。Amazon S3、Amazon SageMaker、および AWS Lambda の代わりに、DB クラスターのルートテーブルに他のサービスの VPC エンドポイントが関連付けられるように、VPC を設定することもできます。「VPC 内の DB クラスターの使用」を参照してください。VPC での NAT の設定の詳細については、「NAT ゲートウェイ」を参照してください。VPC エンドポイントの設定の詳細については、「VPC エンドポイント」を参照してください。S3 バケットにアクセスするための S3 ゲートウェイエンドポイントを作成することもできます。詳細については、「Amazon S3 のゲートウェイエンドポイント」を参照してください。

また、VPC セキュリティグループのアウトバウンドルールで、ネットワークアクセスコントロールリスト (ACL) のエフェメラルポートを開く必要がある場合もあります。ネットワーク ACL のエフェメラルポートの詳細については、Amazon Virtual Private Cloud ユーザーガイドの「一時ポート」を参照してください。

関連トピック

- Aurora と他の AWS のサービスの統合
- Amazon Aurora DB クラスターの管理

Amazon S3 バケットのテキストファイルから Amazon Aurora MySQL DB クラスターへのデータのロード

`LOAD DATA FROM S3` または `LOAD XML FROM S3` ステートメントを使用して、Amazon S3 バケットに保存されているファイルからデータをロードできます。Aurora MySQL の場合、ファイルは最初にローカルディスクに保存され、次にデータベースにインポートされます。データベースへのインポートが完了すると、ローカルファイルは削除されます。

> **Note**
>
> テキストファイルからテーブルへのデータのロードは、Aurora Serverless v1 ではサポートされていません。Aurora Serverless v2 に対してサポートされています。

目次

Amazon S3 へのアクセスを Aurora に許可する

Amazon S3 バケットからデータをロードする前に、まず Amazon S3 へのアクセス権限を Aurora MySQL DB クラスターに付与する必要があります。

Amazon S3 へのアクセス権限を Aurora MySQL に付与するには

1. バケットおよびオブジェクトのアクセス許可を付与し、Aurora MySQL DB クラスターから Amazon S3 へのアクセスを許可する AWS Identity and Access Management (IAM) ポリシーを作成します。手順については、Amazon S3 リソースにアクセスするための IAM ポリシーの作成を参照してください。

   > **Note**
   >
   > Aurora MySQL バージョン 3.05 以降では、カスタマーマネージド AWS KMS keys を使用して暗号化されたオブジェクトをロードできます。そのためには、IAM ポリシーに `kms:Decrypt` アクセス許可を含めてください。詳細については、「AWS KMS リソースにアクセスするための IAM ポリシーの作成」を参照してください。
   > AWS マネージドキー または Amazon S3 マネージドキー (SSE-S3) を使用して暗号化されたオブジェクトをロードする場合、このアクセス許可は必要ありません。

2. IAM ロールを作成して、「Amazon S3 リソースにアクセスするための IAM ポリシーの作成」で作成した IAM ポリシーを新しい IAM ロールにアタッチします。手順については、「Amazon Aurora が AWS のサービスにアクセスすることを許可する IAM ロールの作成」を参照してください。
3. DB クラスターがカスタム DB クラスターパラメータグループを使用していることを確認します。

   カスタム DB クラスターパラメータグループの作成の詳細については、「DB クラスターのパラメータグループの作成」を参照してください。
4. Aurora MySQL バージョン 2 の場合、`aurora_load_from_s3_role` または `aws_default_s3_role` DB クラスターパラメータを、新しい IAM ロールの Amazon リソースネーム (ARN) に設定します。IAM ロールが `aurora_load_from_s3_role` に指定されていない場合、Aurora は `aws_default_s3_role` に指定されている IAM ロールを使用します。

   Aurora MySQL バージョン 3 の場合、`aws_default_s3_role` を使用します。

   クラスターが Aurora Global Database の一部である場合は、このパラメータをグローバルデータベース内の Aurora クラスターごとに設定します。Aurora Global Database 内のプライマリクラスターのみがデータをロードできますが、フェイルオーバー機構によって別のクラスターが昇格されてプライマリクラスターになる場合があります。

   DB クラスターのパラメータの詳細については、「Amazon Aurora の DB クラスターパラメータと DB インスタンスパラメータ」を参照してください。
5. Aurora MySQL DB クラスター内のデータベースユーザーが Amazon S3 にアクセスできるように、「Amazon Aurora が AWS のサービスにアクセスすることを許可する IAM ロールの作成」で作成したロールをその DB クラスターに関連付けます。Aurora Global Database の場合は、グローバルデータベース内の Aurora クラスターごとにロールを関連付けます。DB クラスターへの IAM ロールの関連付けの詳細については、「IAM ロールと Amazon Aurora MySQL DB クラスターの関連付け」を参照してください。
6. Amazon S3 へのアウトバウンド接続を許可するように Aurora MySQL DB クラスターを設定します。手順については、Amazon Aurora MySQL から他の AWS のサービスへのネットワーク通信の有効化 を参照してください。

   DB クラスターがパブリックアクセス可能ではなく、VPC パブリックサブネットにある場合は、プライベートです。S3 バケットにアクセスするための S3 ゲートウェイエンドポイントを作成することができます。詳細については、「Amazon S3 のゲートウェイエンドポイント」を参照してください。

Aurora Global Database の場合は、グローバルデータベース内の Aurora クラスターごとにアウトバウンド接続を有効にします。

Amazon Aurora MySQL でデータをロードするための権限の付与

`LOAD DATA FROM S3`または`LOAD XML FROM S3`ステートメントを発行するデータベースユーザーは、いずれかのステートメントを発行するための特定のロールまたは特権を持っている必要があります。Aurora MySQL バージョン 3 では、`AWS_LOAD_S3_ACCESS` ロールを付与します。Aurora MySQL バージョン 2 では、`LOAD FROM S3` 権限を付与します。DB クラスターの管理ユーザーにはデフォルトで適切なロールまたは権限が付与されます。他のユーザーに権限を付与するには、次のいずれかのコマンドが使用できます。

Aurora MySQL バージョン 3 では、次のステートメントを使用します:

```
GRANT AWS_LOAD_S3_ACCESS TO 'user'@'domain-or-ip-address'
```

> **Tip**
>
> Aurora MySQL バージョン 3 でロールテクニックを使用する場合は、`SET ROLE role_name` または `SET ROLE ALL` ステートメントを使用してロールを有効化することもできます。MySQL 8.0 ロールシステムに馴染みがない場合は、ロールベースの特権モデルで詳細を確認頂けます。詳細については、「MySQL リファレンスマニュアル」の「Using roles」を参照してください。
> これは現在アクティブなセッションにのみ適用されます。再接続するときは、`SET ROLE` ステートメントを再度実行して権限を付与する必要があります。詳細については、MySQL リファレンスマニュアルの「SET ROLE ステートメント」を参照してください。
> ユーザーが DB インスタンスに接続したときに、`activate_all_roles_on_login` DB クラスターパラメータを使用して、すべてのロールを自動的に有効化できます。このパラメータを設定すると、通常、`SET ROLE` ステートメントを明示的に呼び出してロールをアクティブ化する必要はありません。詳細については、MySQL リファレンスマニュアルの「activate_all_roles_on_login」を参照してください。
> ただし、ストアドプロシージャを別のユーザーから呼び出す場合は、ストアドプロシージャの先頭で `SET ROLE ALL` を明示的に呼び出してロールをアクティブ化する必要があります。

Aurora MySQL バージョン 2 では、次のステートメントを使用します:

```
GRANT LOAD FROM S3 ON *.* TO 'user'@'domain-or-ip-address'
```

AWS_LOAD_S3_ACCESS ロールと LOAD FROM S3 権限は Amazon Aurora に固有であり、外部の MySQL データベースまたは RDS for MySQL DB インスタンスでは使用できません。レプリケーションマスターとしての Aurora DB クラスターと、レプリケーションクライアントとしての MySQL データベースの間でレプリケーションを設定した場合、ロール および権限の GRANT ステートメントはエラーとなり、レプリケーションは停止します。エラーをスキップして、レプリケートを再開できます。RDS for MySQL インスタンスでエラーをスキップするには、 mysql_rds_skip_repl_error プロシージャを使用します。外部 MySQL データベースでエラーをスキップするには、slave_skip_errors システム変数 (Aurora MySQL バージョン 2) または replica_skip_errors システム変数 (Aurora MySQL バージョン 3) を使用します。

> **Note**
>
> データベースユーザーには、データをロードする先のデータベースに対する INSERT 権限が必要です。

Amazon S3 バケットへのパス (URI) の指定

Amazon S3 バケットに保存されているファイルへのパスを指定する構文は次のとおりです。

```
s3-region://bucket-name/file-name-or-prefix
```

パスに指定する値は以下のとおりです。

- region (オプション) - ロード元の Amazon S3 バケットがある AWS リージョン。この値はオプションです。region 値を指定しないと、Aurora は DB クラスターと同じリージョンの Amazon S3 からファイルをロードします。
- bucket-name - ロードするデータが含まれている Amazon S3 バケットの名前。仮想フォルダのパスを識別するオブジェクトプレフィックスがサポートされています。
- file-name-or-prefix - Amazon S3 テキストファイルまたは XML ファイルの名前、あるいはロードする 1 つ以上のテキストファイルまたは XML ファイルを識別するプレフィックス。ロードするテキストファイルを識別するマニフェストファイルを指定することもできます。マニフェストファイルを使用して Amazon S3 からテキストファイルをロードする方法の詳細については、「マニフェストを使用して、ロードするデータファイルを指定する」を参照してください。

S3 バケット内のファイルの URI をコピーするには

1. AWS Management Console にサインインし、Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開きます。
2. ナビゲーションペインで [バケット] を選択し、URI をコピーするバケットを選択します。
3. S3 からロードするプレフィックスまたはファイルを選択します。
4. [S3 URI をコピー] を選択します。

LOAD DATA FROM S3

`LOAD DATA FROM S3` ステートメントを使用して MySQL LOAD DATA INFILE ステートメントでサポートされている任意のテキストファイル形式 (カンマ区切りのテキストデータなど) からデータをロードできます。圧縮ファイルはサポートされていません。

> **Note**
>
> Aurora MySQL DB クラスターが S3 へのアウトバウンド接続を許可していることを確認してください。詳細については、「Amazon Aurora MySQL から他の AWS のサービスへのネットワーク通信の有効化」を参照してください。

構文

```
LOAD DATA [FROM] S3 [FILE | PREFIX | MANIFEST] 'S3-URI'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [PARTITION (partition_name,...)]
    [CHARACTER SET charset_name]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
    ]
    [IGNORE number {LINES | ROWS}]
    [(col_name_or_user_var,...)]
```

```
    [SET col_name = expr,...]
```

> **Note**
>
> Aurora MySQL バージョン 3.05 以降では、キーワード FROM はオプションです。

パラメータ

LOAD DATA FROM S3 ステートメントでは、次の必須パラメータとオプションパラメータを使用します。これらのパラメータの一部の詳細については、MySQL ドキュメントの「LOAD XML Statement」(LOAD DATA ステートメント) を参照してください。

ファイル | プレフィックス | マニフェスト

データを読み取るのが、1 つのファイルからか、特定のプレフィックスに一致するすべてのファイルからか、指定されたマニフェスト内のすべてのファイルからかを指定します。デフォルトは FILE です。

S3-URI

ロードするテキストファイルまたはマニフェストファイルの URI、または、使用する Amazon S3 プレフィックスを指定します。「Amazon S3 バケットへのパス (URI) の指定」で説明されている構文を使用して URI を指定します。

置換 | 無視

入力行とデータベーステーブルの既存の行で一意のキー値が同じである場合、どのアクションを実行するかを決定します。

- テーブル内の既存の行を入力行で置き換える場合は、REPLACE を指定します。
- 入力行を破棄する場合は、IGNORE を指定します。

テーブルに

入力行をロードする先のデータベーステーブルの名前を指定します。

PARTITION

すべての入力行を、指定されたカンマ区切りのパーティション名のリストに記載されているパーティション内に挿入することを要求します。指定されたパーティションのいずれかに入力行を挿入できない場合、ステートメントは失敗し、エラーが返されます。

文字セット

入力ファイルのデータの文字セットを指定します。

フィールド | 列

入力ファイルのフィールドまたは列を区切る方法を指定します。デフォルトでは、フィールドはタブで区切られます。

LINES

入力ファイルの行を区切る方法を指定します。デフォルトでは、行は改行文字 ('\n') で区切られます。

number 行を無視 | 行

入力ファイルの先頭の特定の数の行または列を無視することを指定します。例えば、IGNORE 1 LINES では列名を含む初期のヘッダー行がスキップされます。IGNORE 2 ROWS では、入力ファイルの先頭から 2 行のデータがスキップされます。また、PREFIX を使用すると、IGNORE は初期の入力ファイルの先頭にある特定の行数または行をスキップします。

col_name_or_user_var, ..。

1 つ以上の列名、またはロードする列を名前で指定するユーザー変数のカンマ区切りのリストを指定します。この目的で使用されるユーザー可変の名前は、プレフィックスを @ とするテキストファイルの要素名と一致する必要があります。フィールド値を対応するユーザー可変に保存して後で再利用できます。

例えば、次のステートメントでは、入力ファイルの初期の列を table1 の初期の列内にロードし、さらに table_column2 の table1 列の値として、2 列目の入力値を 100 で割った値を設定します。

```
LOAD DATA FROM S3 's3://mybucket/data.txt'
    INTO TABLE table1
    (column1, @var1)
    SET table_column2 = @var1/100;
```

SET

テーブル内の列の値を入力ファイルに含まれていない値に設定する代入操作のカンマ区切りのリストを指定します。

例えば、次のステートメントは table1 の初期の 2 列を入力ファイルの初期の 2 列の値に設定し、さらに column3 の table1 の値を現在のタイムスタンプに設定します。

```
LOAD DATA FROM S3  's3://mybucket/data.txt'
    INTO TABLE table1
    (column1, column2)
    SET column3 = CURRENT_TIMESTAMP;
```

SET 割り当ての右側のサブクエリを使用できます。列に割り当てる値を返すサブクエリとしては、スカラーサブクエリのみ使用できます。また、サブクエリを使用してロード中のテーブルから選択することはできません。

Amazon S3 バケットからデータをロードする場合、LOAD DATA FROM S3 ステートメントの LOCAL キーワードは使用できません。

マニフェストを使用して、ロードするデータファイルを指定する

LOAD DATA FROM S3 ステートメントの MANIFEST キーワードで、JSON 形式のマニフェストファイルを指定して、DB クラスターのテーブルにロードするテキストファイルをリストできます。

次の JSON スキーマはマニフェストファイルの形式と内容を示しています。

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "additionalProperties": false,
    "definitions": {},
    "id": "Aurora_LoadFromS3_Manifest",
    "properties": {
        "entries": {
            "additionalItems": false,
            "id": "/properties/entries",
            "items": {
                "additionalProperties": false,
                "id": "/properties/entries/items",
                "properties": {
                    "mandatory": {
                        "default": "false",
                        "id": "/properties/entries/items/properties/mandatory",
                        "type": "boolean"
                    },
                    "url": {
                        "id": "/properties/entries/items/properties/url",
                        "maxLength": 1024,
                        "minLength": 1,
```

```
                        "type": "string"
                    }
                },
                "required": [
                    "url"
                ],
                "type": "object"
            },
            "type": "array",
            "uniqueItems": true
        }
    },
    "required": [
        "entries"
    ],
    "type": "object"
}
```

マニフェスト内の各 url では、プレフィックスだけでなく、バケットの名前とファイルの完全なオブジェクトパスを使用して URL を指定する必要があります。マニフェストを使用し、異なるバケットやリージョンからファイルをロードしたり、同じプレフィックスを共有しないファイルをロードしたりできます。URL にリージョンを指定していない場合は、ターゲットの Aurora DB クラスターのリージョンが使用されます。以下の例では、異なるバケットから 4 つのファイルをロードするマニフェストファイルを示しています。

```
{
  "entries": [
    {
      "url":"s3://aurora-bucket/2013-10-04-customerdata",
      "mandatory":true
    },
    {
      "url":"s3-us-west-2://aurora-bucket-usw2/2013-10-05-customerdata",
      "mandatory":true
    },
    {
      "url":"s3://aurora-bucket/2013-10-04-customerdata",
      "mandatory":false
    },
    {
      "url":"s3://aurora-bucket/2013-10-05-customerdata"
    }
```

```
  ]
}
```

ファイルが見つからない場合に、オプションの mandatory フラグによって コマンドがエラーを返すかどうかを指定します。LOAD DATA FROM S3mandatory フラグはデフォルトで false に設定されています。mandatory の設定方法にかかわらず、ファイルが見つからない場合、LOAD DATA FROM S3 は終了します。

マニフェストファイルには、任意の拡張子を付けることができます。次の例では、前の例にあった「LOAD DATA FROM S3」という名前のマニフェストで **customer.manifest** ステートメントを実行しています。

```
LOAD DATA FROM S3 MANIFEST 's3-us-west-2://aurora-bucket/customer.manifest'
    INTO TABLE CUSTOMER
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (ID, FIRSTNAME, LASTNAME, EMAIL);
```

ステートメントが完了すると、正常にロードされた各ファイルのエントリが aurora_s3_load_history テーブルに書き込まれます。

aurora_s3_load_history テーブルを使用してロード済みのファイルを確認する

LOAD DATA FROM S3 ステートメントが成功するたびに、aurora_s3_load_history スキーマの mysql テーブルが、ロードされた各ファイルのエントリで更新されます。

LOAD DATA FROM S3 ステートメントの実行後、aurora_s3_load_history テーブルをクエリすることで、どのファイルがロードされたかを確認できます。ステートメントの 1 回のイテレーションからロードされたファイルを確認するには、WHERE 句を使用して Amazon S3 URI のレコードをフィルタリングし、ステートメントで使用されたマニフェストファイルを見つけます。以前に同じマニフェストファイルを使用した場合は、timestamp フィールドを使用して結果をフィルタリングします。

```
select * from mysql.aurora_s3_load_history where load_prefix = 'S3_URI';
```

以下の表では、aurora_s3_load_history テーブルのフィールドについて説明しています。

フィールド	説明
load_prefix	load ステートメントで指定した URI。この URI は、次のいずれかにマッピングできます。 • LOAD DATA FROM S3 FILE ステートメントの単一のデータファイル • LOAD DATA FROM S3 PREFIX ステートメントの複数のデータファイルにマッピングする Amazon S3 プレフィックス • LOAD DATA FROM S3 MANIFEST ステートメントにロードされるファイルの名前を含む単一のマニフェストファイル
file_name	load_prefix フィールドで指定した URI を使用して Amazon S3 から Aurora にロードされたファイルの名前。
version_number	Amazon S3 バケットにバージョン番号がある場合、ロードされた file_name フィールドで識別されるファイルのバージョン番号。
bytes_loaded	ロードされたファイルのサイズ (バイト単位)。
load_timestamp	LOAD DATA FROM S3 ステートメントが完了したときのタイムスタンプ。

例

次のステートメントでは、Aurora DB クラスターと同じリージョンにある Amazon S3 バケットからデータをロードします。customerdata.txt Amazon S3 バケットにある dbbucket ファイルのカンマ区切りデータを読み取り、そのデータを store-schema.customer-table テーブル内にロードします。

```
LOAD DATA FROM S3 's3://dbbucket/customerdata.csv'
    INTO TABLE store-schema.customer-table
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (ID, FIRSTNAME, LASTNAME, ADDRESS, EMAIL, PHONE);
```

次のステートメントでは、Aurora DB クラスターとは別のリージョンにある Amazon S3 バケットからデータをロードします。employee-data リージョンの my-data Amazon S3 バケットで us-west-2 オブジェクトプレフィックスと一致するすべてのファイルからカンマ区切りのデータを読み取り、そのデータを employees テーブル内にロードします。

```
LOAD DATA FROM S3 PREFIX 's3-us-west-2://my-data/employee_data'
    INTO TABLE employees
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (ID, FIRSTNAME, LASTNAME, EMAIL, SALARY);
```

次のステートメントでは、q1_sales.json という名前の JSON マニフェストファイルに指定されているファイルから sales テーブルにデータをロードします。

```
LOAD DATA FROM S3 MANIFEST 's3-us-west-2://aurora-bucket/q1_sales.json'
    INTO TABLE sales
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (MONTH, STORE, GROSS, NET);
```

LOAD XML FROM S3

LOAD XML FROM S3 ステートメントを使用して、Amazon S3 バケットに保存されている XML ファイルのデータを以下の 3 種類の XML フォーマットのいずれかでロードできます。

- <row> 要素の属性としての列名。属性値はテーブルフィールドの内容を指定します。

  ```
  <row column1="value1" column2="value2" .../>
  ```

- <row> 要素の子要素としての列名。子要素の値は、テーブルフィールドの内容を指定します。

  ```
  <row>
    <column1>value1</column1>
    <column2>value2</column2>
  </row>
  ```

- name 要素で <field> 要素の <row> 属性の列名。<field> 要素の値は、テーブルフィールドの内容を指定します。

  ```
  <row>
    <field name='column1'>value1</field>
  ```

```
  <field name='column2'>value2</field>
</row>
```

構文

```
LOAD XML FROM S3 'S3-URI'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [PARTITION (partition_name,...)]
    [CHARACTER SET charset_name]
    [ROWS IDENTIFIED BY '<element-name>']
    [IGNORE number {LINES | ROWS}]
    [(field_name_or_user_var,...)]
    [SET col_name = expr,...]
```

パラメータ

LOAD XML FROM S3 ステートメントでは、次の必須パラメータとオプションパラメータを使用します。これらのパラメータの一部の詳細については、MySQL ドキュメントの「LOAD XML Statement」を参照してください。

ファイル | プレフィックス

データを単一のファイルからロードするか、特定のプレフィックスに一致するすべてのファイルからロードするかを指定します。デフォルトは FILE です。

置換 | 無視

入力行とデータベーステーブルの既存の行で一意のキー値が同じである場合、どのアクションを実行するかを決定します。

- テーブル内の既存の行を入力行で置き換える場合は、REPLACE を指定します。
- 入力行を破棄する場合は、IGNORE を指定します。デフォルトは IGNORE です。

テーブルに

入力行をロードする先のデータベーステーブルの名前を指定します。

PARTITION

すべての入力行を、指定されたカンマ区切りのパーティション名のリストに記載されているパーティション内に挿入することを要求します。指定されたパーティションのいずれかに入力行を挿入できない場合、ステートメントは失敗し、エラーが返されます。

文字セット

入力ファイルのデータの文字セットを指定します。

行の識別

入力ファイルの行を識別する要素名を指定します。デフォルト: <row>。

number 行を無視 | 行

入力ファイルの先頭の特定の数の行または列を無視することを指定します。例えば、IGNORE 1 LINES ではテキストファイルの初期の行がスキップされます。IGNORE 2 ROWS では、入力 XML の初期の 2 行のデータがスキップされます。

field_name_or_user_var, ..。

ロードする要素を名前で指定する 1 つ以上の XML 要素名またはユーザー変数のカンマ区切りのリストを指定します。この目的で使用されるユーザー可変の名前は、プレフィックスを @ とする XML ファイルの要素名と一致する必要があります。フィールド値を対応するユーザー可変に保存して後で再利用できます。

例えば、次のステートメントでは、入力ファイルの初期の列を table1 の初期の列内にロードし、さらに table_column2 の table1 列の値として、2 列目の入力値を 100 で割った値を設定します。

```
LOAD XML FROM S3 's3://mybucket/data.xml'
   INTO TABLE table1
   (column1, @var1)
   SET table_column2 = @var1/100;
```

SET

テーブル内の列の値を入力ファイルに含まれていない値に設定する代入操作のカンマ区切りのリストを指定します。

例えば、次のステートメントは table1 の初期の 2 列を入力ファイルの初期の 2 列の値に設定し、さらに column3 の table1 の値を現在のタイムスタンプに設定します。

```
LOAD XML FROM S3 's3://mybucket/data.xml'
   INTO TABLE table1
   (column1, column2)
   SET column3 = CURRENT_TIMESTAMP;
```

SET 割り当ての右側のサブクエリを使用できます。列に割り当てる値を返すサブクエリとしては、スカラーサブクエリのみ使用できます。また、サブクエリを使用してロード中のテーブルから選択することはできません。

Amazon Aurora MySQL DB クラスターから Amazon S3 バケット内のテキストファイルへのデータの保存

SELECT INTO OUTFILE S3 ステートメントを使用して、Amazon Aurora MySQL DB クラスターのデータをクエリし、Amazon S3 バケット内のテキストファイルにデータを保存できます。Aurora MySQL では、ファイルを最初にローカルディスクに保存し、次に S3 にエクスポートします。エクスポートが完了すると、ローカルファイルは削除されます。

Amazon S3 マネージドキー (SSE-S3) または AWS KMS key (SSE-KMS: AWS マネージドキー またはカスタマーマネージドキー) を使用して Amazon S3 バケットを暗号化できます。

LOAD DATA FROM S3 ステートメントは、SELECT INTO OUTFILE S3 ステートメントで作成したファイルを使用してデータを Aurora DB クラスター内にロードできます。詳細については、「Amazon S3 バケットのテキストファイルから Amazon Aurora MySQL DB クラスターへのデータのロード」を参照してください。

> **Note**
>
> この機能は Aurora Serverless v1 DB クラスターに対してサポートされていません。Aurora Serverless v2 DB クラスターでサポートされています。
> AWS Management Console、AWS CLI、または Amazon RDS API を使用して、DB クラスターデータと DB クラスタースナップショットデータを Amazon S3 に保存することもできます。詳細については、Amazon S3 への DB クラスターデータのエクスポートおよびAmazon S3 への DB クラスタースナップショットデータのエクスポートを参照してください。

目次

- SELECT INTO OUTFILE S3
  - 構文
  - パラメータ
  - 考慮事項
  - 例

Amazon S3 へのアクセスを Aurora MySQL に許可する

データを Amazon S3 バケット内に保存する前に、Amazon S3 へのアクセス権限を Aurora MySQL DB クラスターに付与する必要があります。

Amazon S3 へのアクセス権限を Aurora MySQL に付与するには

1. バケットおよびオブジェクトのアクセス許可を付与し、Aurora MySQL DB クラスターから Amazon S3 へのアクセスを許可する AWS Identity and Access Management (IAM) ポリシーを作成します。手順については、Amazon S3 リソースにアクセスするための IAM ポリシーの作成を参照してください。

   > **Note**
   >
   > Aurora MySQL バージョン 3.05 以降では、AWS KMS カスタマーマネージドキーを使用してオブジェクトを暗号化できます。そのためには、IAM ポリシーに `kms:GenerateDataKey` アクセス許可を含めてください。詳細については、「AWS KMS リソースにアクセスするための IAM ポリシーの作成」を参照してください。AWS マネージドキー または Amazon S3 マネージドキー (SSE-S3) を使用してオブジェクトを暗号化する場合にはこのアクセス許可は必要ありません。

2. IAM ロールを作成して、「Amazon S3 リソースにアクセスするための IAM ポリシーの作成」で作成した IAM ポリシーを新しい IAM ロールにアタッチします。手順については、Amazon Aurora が AWS のサービスにアクセスすることを許可する IAM ロールの作成 を参照してください。
3. Aurora MySQL バージョン 2 の場合、`aurora_select_into_s3_role` または `aws_default_s3_role` DB クラスターパラメータを、新しい IAM ロールの Amazon リソースネーム (ARN) に設定します。IAM ロールが `aurora_select_into_s3_role` に指定されていない場合、Aurora は `aws_default_s3_role` に指定されている IAM ロールを使用します。

   Aurora MySQL バージョン 3 の場合、`aws_default_s3_role` を使用します。

クラスターが Aurora Global Database の一部である場合は、このパラメータをグローバルデータベース内の Aurora クラスターごとに設定します。

DB クラスターのパラメータの詳細については、「Amazon Aurora の DB クラスターパラメータと DB インスタンスパラメータ」を参照してください。

4. Aurora MySQL DB クラスター内のデータベースユーザーが Amazon S3 にアクセスできるように、「Amazon Aurora が AWS のサービスにアクセスすることを許可する IAM ロールの作成」で作成したロールをその DB クラスターに関連付けます。

   Aurora Global Database の場合は、グローバルデータベース内の Aurora クラスターごとにロールを関連付けます。

   DB クラスターへの IAM ロールの関連付けの詳細については、「IAM ロールと Amazon Aurora MySQL DB クラスターの関連付け」を参照してください。

5. Amazon S3 へのアウトバウンド接続を許可するように Aurora MySQL DB クラスターを設定します。手順については、「Amazon Aurora MySQL から他の AWS のサービスへのネットワーク通信の有効化」を参照してください。

   Aurora Global Database の場合は、グローバルデータベース内の Aurora クラスターごとにアウトバウンド接続を有効にします。

Aurora MySQL にデータを保存する権限の付与

`SELECT INTO OUTFILE S3` ステートメントを発行するデータベースユーザーは、特定のロールまたは権限を保持している必要があります。Aurora MySQL バージョン 3 では、`AWS_SELECT_S3_ACCESS` ロールを付与します。Aurora MySQL バージョン 2 では、`SELECT INTO S3` 権限を付与します。DB クラスターの管理ユーザーにはデフォルトで適切なロールまたは権限が付与されます。他のユーザーに権限を付与するには、次のいずれかのコマンドが使用できます。

Aurora MySQL バージョン 3 では、次のステートメントを使用します:

```
GRANT AWS_SELECT_S3_ACCESS TO 'user'@'domain-or-ip-address'
```

> **Tip**
>
> Aurora MySQL バージョン 3 でロールテクニックを使用する場合は、`SET ROLE role_name` または `SET ROLE ALL` ステートメントを使用してロールを有効化することもできます。MySQL 8.0 ロールシステムに馴染みがない場合は、ロールベースの特権モデルで詳細を確認頂けます。詳細については、「MySQL リファレンスマニュアル」の「Using roles」を参照してください。
> これは現在アクティブなセッションにのみ適用されます。再接続するときは、`SET ROLE` ステートメントを再度実行して権限を付与する必要があります。詳細については、MySQL リファレンスマニュアルの「SET ROLE ステートメント」を参照してください。
> ユーザーが DB インスタンスに接続したときに、`activate_all_roles_on_login` DB クラスターパラメータを使用して、すべてのロールを自動的に有効化できます。このパラメータを設定すると、通常、`SET ROLE` ステートメントを明示的に呼び出してロールをアクティブ化する必要はありません。詳細については、MySQL リファレンスマニュアルの「activate_all_roles_on_login」を参照してください。
> ただし、ストアドプロシージャを別のユーザーから呼び出す場合は、ストアドプロシージャの先頭で `SET ROLE ALL` を明示的に呼び出してロールをアクティブ化する必要があります。

Aurora MySQL バージョン 2 では、次のステートメントを使用します:

```
GRANT SELECT INTO S3 ON *.* TO 'user'@'domain-or-ip-address'
```

`AWS_SELECT_S3_ACCESS` ロールまたは `SELECT INTO S3` 権限は Amazon Aurora MySQL に固有であり、MySQL データベースおよび MySQL DB インスタンスの RDS では使用できません。Aurora MySQL DB クラスターをレプリケーションマスターとし、MySQL データベースをレプリケーションクライアントとして両者間にレプリケーションを設定した場合、ロールおよび権限の `GRANT` ステートメントはエラーとなり、レプリケーションが停止します。エラーをスキップして、レプリケートを再開できます。RDS for MySQL DB インスタンスでエラーをスキップするには、 mysql_rds_skip_repl_error プロシージャを使用します。外部 MySQL データベースでエラーをスキップするには、slave_skip_errors システム変数 (Aurora MySQL バージョン 2) または replica_skip_errors システム変数 (Aurora MySQL バージョン 3) を使用します。

Amazon S3 バケットへのパスの指定

Amazon S3 バケットにデータとマニフェストファイルを保存するためのパスを指定する構文は、次に示すように、`LOAD DATA FROM S3 PREFIX` ステートメントで使用する構文と似ています。

```
s3-region://bucket-name/file-prefix
```

パスに指定する値は以下のとおりです。

- region (オプション) - データの保存先の Amazon S3 バケットがある AWS リージョン。この値はオプションです。region 値を指定しないと、Aurora は DB クラスターと同じリージョンの Amazon S3 にファイルを保存します。
- bucket-name - データの保存先である Amazon S3 バケットの名前。仮想フォルダのパスを識別するオブジェクトプレフィックスがサポートされています。
- file-prefix - Amazon S3 に保存するファイルを識別する Amazon S3 オブジェクトプレフィックス。

SELECT INTO OUTFILE S3 ステートメントで作成したデータファイルでは、次のパスを使用します。*00000* はゼロから始まる 5 桁の整数です。

```
s3-region://bucket-name/file-prefix.part_00000
```

例えば、SELECT INTO OUTFILE S3 ステートメントでデータファイルの保存先のパスとして s3-us-west-2://bucket/prefix を指定し、3 つのデータファイルを作成したとします。この場合、指定した Amazon S3 バケットに保存されるデータファイルは以下のとおりです。

- s3-us-west-2://bucket/prefix.part_00000
- s3-us-west-2://bucket/prefix.part_00001
- s3-us-west-2://bucket/prefix.part_00002

データファイルをリスト化するマニフェストの作成

SELECT INTO OUTFILE S3 ステートメントで MANIFEST ON オプションを使用すると、このステートメントで作成されるテキストファイルをリストするマニフェストファイルを JSON 形式で作成できます。LOAD DATA FROM S3 ステートメントでは、このマニフェストファイルを使用してデータファイルを逆に Aurora MySQL DB クラスター内にロードできます。マニフェストファイルを使用して Amazon S3 から Aurora MySQL DB クラスター内にデータファイルをロードする方法の詳細については、「マニフェストを使用して、ロードするデータファイルを指定する」を参照してください。

SELECT INTO OUTFILE S3 ステートメントで作成されたマニフェストでは、このステートメントで作成された順にデータファイルがリストされます。例えば、SELECT INTO OUTFILE S3 ステートメントでデータファイルの保存先のパスとして s3-us-west-2://bucket/prefix を指定し、マニフェストファイルを作成したとします。この場合、指定した Amazon S3 バケットに保存されるマニフェストファイルは s3-us-west-2://bucket/prefix.manifest という名前で、以下の内容になります。

```
{
  "entries": [
    {
      "url":"s3-us-west-2://bucket/prefix.part_00000"
    },
    {
      "url":"s3-us-west-2://bucket/prefix.part_00001"
    },
    {
      "url":"s3-us-west-2://bucket/prefix.part_00002"
    }
  ]
}
```

SELECT INTO OUTFILE S3

SELECT INTO OUTFILE S3 ステートメントを使用して、DB クラスターからクエリしたデータを直接 Amazon S3 バケット内の区切りテキストファイルに保存できます。

圧縮ファイルはサポートされていません。暗号化ファイルは Aurora MySQL バージョン 2.09.0 以降でサポートされています。

構文

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
        [HIGH_PRIORITY]
        [STRAIGHT_JOIN]
        [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
        [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
        [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
```

```
        [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
         [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
INTO OUTFILE S3 's3_uri'
[CHARACTER SET charset_name]
    [export_options]
    [MANIFEST {ON | OFF}]
    [OVERWRITE {ON | OFF}]
    [ENCRYPTION {ON | OFF | SSE_S3 | SSE_KMS ['cmk_id']}]

export_options:
    [FORMAT {CSV|TEXT} [HEADER]]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
]
```

パラメータ

SELECT INTO OUTFILE S3 ステートメントでは、Aurora に固有の次の必須パラメータとオプションパラメータを使用します。

s3-uri

Amazon S3 のプレフィックスとして使用する URI を指定します。「Amazon S3 バケットへのパスの指定」で説明されている構文を使用してください。

フォーマット {CSV|テキスト} [ヘッダー]

必要に応じて、データを CSV 形式で保存します。

TEXT オプションはデフォルトで、既存の MySQL エクスポート形式を生成します。

CSV オプションは、カンマ区切りのデータ値を生成します。CSV 形式は、 RFC-4180 の仕様に従います。オプションのキーワード HEADER を指定すると、出力ファイルに 1 つのヘッダー行が含まれます。ヘッダー行のラベルは、SELECT ステートメントの列名に対応します。AWS ML サービスで使用するトレーニングデータモデルに CSV ファイルを使用できます。AWS ML サー

ビスでエクスポートされた Aurora データを使用する方法の詳細については、「SageMaker モデルトレーニング用のデータを Amazon S3 にエクスポートする (高度)」を参照してください。

マニフェスト {オン | オフ}

Amazon S3 でマニフェストファイルを作成するかどうかを指定します。マニフェストファイルは、JSON (JavaScript Object Notation) ファイルであり、LOAD DATA FROM S3 MANIFEST ステートメントでデータを Aurora DB クラスター内にロードする際に使用できます。LOAD DATA FROM S3 MANIFEST の詳細については、Amazon S3 バケットのテキストファイルから Amazon Aurora MySQL DB クラスターへのデータのロード を参照してください。

クエリで MANIFEST ON を指定すると、すべてのデータファイルが作成されてアップロードされた後で、Amazon S3 内にマニフェストファイルが作成されます。マニフェストファイルは次のパスで作成されます。

```
s3-region://bucket-name/file-prefix.manifest
```

マニフェストファイルのコンテンツの詳しい形式については、「データファイルをリスト化するマニフェストの作成」を参照してください。

上書き {オン | オフ}

指定した Amazon S3 バケット内の既存のファイルを上書きするかどうかを指定します。OVERWRITE ON を指定すると、s3-uri に指定した URI のファイルプレフィックスと一致する既存のファイルは上書きされます。上書きされないと、エラーが発生します。

ENCRYPTION {ON | OFF | SSE_S3 | SSE_KMS ['*cmk_id*']}

Amazon S3 マネージドキーを使ったサーバー側の暗号化 (SSE-S3) または AWS KMS keys (SSE-KMS、AWS マネージドキー およびカスタマーマネージドキーを含む) を使用するかどうかを指定します。SSE_S3 および SSE_KMS 設定は、Aurora MySQL バージョン 3.05 以降で使用できます。

次の例に示すように、ENCRYPTION 句の代わりに aurora_select_into_s3_encryption_default セッション変数を使用することもできます。SQL 句またはセッション変数のいずれかを使用しますが、両方は使用できません。

```
set session set session aurora_select_into_s3_encryption_default={ON | OFF | SSE_S3
 | SSE_KMS};
```

SSE_S3 および SSE_KMS 設定は、Aurora MySQL バージョン 3.05 以降で使用できます。

aurora_select_into_s3_encryption_default を次の値に設定した場合:

- OFF – S3 バケットのデフォルトの暗号化ポリシーが適用されます。aurora_select_into_s3_encryption_default の初期値は OFF です。
- ON または SSE_S3 – S3 オブジェクトは Amazon S3 マネージドキー (SSE-S3) を使用して暗号化されます。
- SSE_KMS – S3 オブジェクトは AWS KMS key を使用して暗号化されます。

  この場合は、セッション変数 aurora_s3_default_cmk_id も含めます。次に例を示します。

```
set session aurora_select_into_s3_encryption_default={SSE_KMS};
set session aurora_s3_default_cmk_id={NULL | 'cmk_id'};
```

  - aurora_s3_default_cmk_id が NULL のとき、S3 オブジェクトは AWS マネージドキーを使用して暗号化されます。
  - aurora_s3_default_cmk_id ががが空でない文字列 cmk_id の場合、S3 オブジェクトはカスタマーマネージドキーを使用して暗号化されます。

    cmk_id の値は空の文字列にはできません。

SELECT INTO OUTFILE S3 コマンドを使用すると、Aurora は次のように暗号化を決定します。

- SQL コマンドに ENCRYPTION 句が含まれている場合、Aurora は ENCRYPTION の値のみに依存し、セッション変数を使用しません。
- ENCRYPTION 句がない場合、Aurora はセッション変数の値に依存します。

詳細については、Amazon Simple Storage Service ユーザーガイドの「Amazon S3 マネージドキー (SSE-S3) によるサーバー側の暗号化」および「AWS KMS キー (SSE-KMS) によるサーバー側の暗号化」を参照してください。

他のパラメータの詳細については、MySQL ドキュメントの「SELECT ステートメント」と「LOAD DATA ステートメント」を参照してください。

考慮事項

Amazon S3 バケットに書き込まれるファイルの数は、SELECT INTO OUTFILE S3 ステートメントで選択したデータの量と Aurora MySQL のファイルサイズのしきい値によって異なります。デフォ

ルトのファイルサイズのしきい値は 6 GB です。ステートメントで選択したデータがファイルサイズのしきい値より少ない場合は、1 つのファイルが作成されます。それ以外の場合は、複数のファイルが作成されます。このステートメントで作成されるファイルについては、以下の点にも留意してください。

- Aurora MySQL では、データファイルの行がファイル境界で分割されないことが保証されます。複数のファイルの場合、最後のファイルを除くすべてのデータファイルは、通常、ファイルサイズのしきい値に近いサイズになります。ただし、ファイルサイズのしきい値を常に下回る必要があるために、1 つの行が 2 つのデータファイル間にまたがる場合がまれにあります。この場合、Aurora MySQL では行が分割されないようにデータファイルを作成するため、ファイルサイズのしきい値を上回ることがあります。
- Aurora MySQL では各 SELECT ステートメントをアトミックトランザクションとして実行するため、SELECT INTO OUTFILE S3 ステートメントで大きなデータセットを選択すると、実行時間が長引く場合があります。何らかの理由でステートメントが失敗すると、ステートメントの発行をやり直す必要が生じる場合があります。ただし、ステートメントが失敗しても、Amazon S3 にアップロード済みのファイルは保存先の Amazon S3 バケット内に残るため、再実行するステートメントでは初期からではなく残りのデータだけをアップロードできます。
- 選択するデータが 25 GB を超える場合は、複数回の SELECT INTO OUTFILE S3 ステートメントを使用してデータを Amazon S3 に保存することをお勧めします。実行するステートメントごとに、保存するデータ部分を選択し、保存先として file_prefix パラメータに異なる s3-uri を指定します。選択するデータを複数のステートメントでパーティション化すると、1 つのステートメントでエラーから回復しやすくなります。1 つのステートメントでエラーが発生した場合は、データの一部だけを再選択して Amazon S3 にアップロードする必要があります。複数のステートメントを使用すると、1 回のトランザクションの実行時間が短くなり、パフォーマンスも向上します。
- 複数の SELECT INTO OUTFILE S3 ステートメント間で、file_prefix パラメータに同じ s3-uri を指定した場合、これらのステートメントを同時に実行してデータを Amazon S3 に保存しようとしたときの動作は定義されていません。
- Aurora MySQL では、テーブルスキーマやファイルメタデータなどのメタデータは Amazon S3 にアップロードされません。
- 障害から回復する目的などで SELECT INTO OUTFILE S3 を再実行する場合もあります。このような場合は、s3-uri で指定するファイルプレフィックスと同じ既存のデータファイルを Amazon S3 バケットから削除するか、OVERWRITE ON クエリで SELECT INTO OUTFILE S3 を指定します。

SELECT INTO OUTFILE S3 ステートメントの成否に応じて、一般的な MySQL エラー番号およびレスポンスが返されます。MySQL エラー番号およびレスポンスにアクセスできない場合は、最も簡単な確認方法として MANIFEST ON をステートメントで指定します。マニフェストファイルは、ステートメントで最後に書き込まれるファイルです。つまり、マニフェストファイルがあれば、ステートメントが完了したことになります。

現在、実行中に SELECT INTO OUTFILE S3 ステートメントの進行状況を直接モニタリングする方法はありません。ただし、このステートメントを使用して Aurora MySQL から Amazon S3 に大量のデータを書き込む際に、ステートメントで選択されるデータのサイズがわかっている場合があります。このような場合、Amazon S3 でデータファイルの作成をモニタリングすることで、進行状況を推測できます。

そのためには、ステートメントで選択する約 6 GB のデータごとにデータファイルが指定先の Amazon S3 バケットに作成されることに注目します。選択対象のデータのサイズを 6 GB で割り、作成されるデータファイルの推定数を割り出します。次に Amazon S3 にアップロードされたファイルの数をステートメントの実行中にモニタリングして、ステートメントの進行状況を推測できます。

例

次のステートメントでは、employees テーブルからすべてのデータを選択し、そのデータを Aurora MySQL DB クラスターとは異なるリージョンにある Amazon S3 バケット内に保存します。このステートメントで作成されるデータファイルでは、各フィールドの末尾にカンマ (,) 文字、各行の末尾に改行 (\n) 文字が付きます。指定先の Amazon S3 バケットに sample_employee_data ファイルプレフィックスと一致するファイルがあると、ステートメントからエラーが返されます。

```
SELECT * FROM employees INTO OUTFILE S3 's3-us-west-2://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n';
```

次のステートメントでは、employees テーブルからすべてのデータを選択し、そのデータを Aurora MySQL DB クラスターと同じリージョンにある Amazon S3 バケット内に保存します。このステートメントで作成されるデータファイルでは、各フィールドの末尾にカンマ (,) 文字、各行の末尾に改行 (\n) 文字が付きます。このステートメントでは、マニフェストファイルも作成されます。指定先の Amazon S3 バケットに sample_employee_data ファイルプレフィックスと一致するファイルがあると、ステートメントからエラーが返されます。

```
SELECT * FROM employees INTO OUTFILE S3 's3://aurora-select-into-s3-pdx/
sample_employee_data'
```

```
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    MANIFEST ON;
```

次のステートメントでは、employees テーブルからすべてのデータを選択し、そのデータを Aurora DB クラスターとは異なるリージョンにある Amazon S3 バケット内に保存します。このステートメントで作成されるデータファイルでは、各フィールドの末尾にカンマ (,) 文字、各行の末尾に改行 (\n) 文字が付きます。指定先の Amazon S3 バケットで sample_employee_data ファイルプレフィックスと一致する既存のファイルは上書きされます。

```
SELECT * FROM employees INTO OUTFILE S3 's3-us-west-2://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    OVERWRITE ON;
```

次のステートメントでは、employees テーブルからすべてのデータを選択し、そのデータを Aurora MySQL DB クラスターと同じリージョンにある Amazon S3 バケット内に保存します。このステートメントで作成されるデータファイルでは、各フィールドの末尾にカンマ (,) 文字、各行の末尾に改行 (\n) 文字が付きます。このステートメントでは、マニフェストファイルも作成されます。指定先の Amazon S3 バケットで sample_employee_data ファイルプレフィックスと一致する既存のファイルは上書きされます。

```
SELECT * FROM employees INTO OUTFILE S3 's3://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    MANIFEST ON
    OVERWRITE ON;
```

Amazon Aurora MySQL DB クラスターからの Lambda 関数の呼び出し

Amazon Aurora MySQL 互換エディション DB クラスターから AWS Lambda 関数を呼び出すには、ネイティブ関数 lambda_sync または lambda_async を使用します。Aurora MySQL から Lambda 関数を呼び出すには、Aurora DB クラスターから Lambda にアクセスできる必要があります。Aurora MySQL へのアクセス権の付与の詳細については、Lambda へのアクセスを Aurora に許可する を参照してください。lambda_sync 関数および lambda_async の保存された関数の詳細については、Aurora MySQL ネイティブ関数を使用した Lambda 関数の呼び出し を参照してください。

ストアドプロシージャを使用して、AWS Lambda 関数を呼び出すこともできます。しかし、ストアドプロシージャの使用は非推奨です。以下のいずれかのバージョンの Aurora MySQL を使用している場合は、Aurora MySQL ネイティブ関数を使用することを強くお勧めします。

- MySQL 5.7 互換クラスターの場合は、Aurora MySQL バージョン 2。
- MySQL 8.0 互換クラスターは、Aurora MySQL バージョン 3.01 以降。ストアドプロシージャは、Aurora MySQL バージョン 3 では使用できません。

トピック

Lambda へのアクセスを Aurora に許可する

Aurora MySQL DB クラスターから Lambda 関数を呼び出すには、まず Lambda へのアクセス権限をクラスターに付与する必要があります。

Lambda へのアクセス権限を Aurora MySQL に付与するには

1. Aurora MySQL DB クラスターからの Lambda 関数の呼び出しを許可するアクセス許可を付与する AWS Identity and Access Management (IAM) ポリシーを作成します。手順については、「AWS Lambda リソースにアクセスするための IAM ポリシーの作成」を参照してください。
2. IAM ロールを作成して、「AWS Lambda リソースにアクセスするための IAM ポリシーの作成」で作成した IAM ポリシーを新しい IAM ロールにアタッチします。手順については、「Amazon Aurora が AWS のサービスにアクセスすることを許可する IAM ロールの作成」を参照してください。
3. DB クラスターの `aws_default_lambda_role` パラメータに新しい IAM ロールの Amazon リソースネーム (ARN) を設定します。

   クラスターが Aurora Global Database の一部である場合は、グローバルデータベース内の Aurora クラスターごとに同じ設定を適用します。

   DB クラスターのパラメータの詳細については、「Amazon Aurora の DB クラスターパラメータと DB インスタンスパラメータ」を参照してください。

4. Aurora MySQL DB クラスター内のデータベースユーザーが Lambda 関数を呼び出せるように、「Amazon Aurora が AWS のサービスにアクセスすることを許可する IAM ロールの作成」で作成したロールをその DB クラスターに関連付けます。DB クラスターへの IAM ロールの関連付けの詳細については、「IAM ロールと Amazon Aurora MySQL DB クラスターの関連付け」を参照してください。

   クラスターが Aurora Global Database の一部である場合は、グローバルデータベース内の Aurora クラスターごとにロールを関連付けます。

5. Lambda へのアウトバウンド接続を許可するように Aurora MySQL DB クラスターを設定します。手順については、「Amazon Aurora MySQL から他の AWS のサービスへのネットワーク通信の有効化」を参照してください。

   クラスターが Aurora Global Database の一部である場合は、グローバルデータベース内の Aurora クラスターごとにアウトバウンド接続を有効にします。

Aurora MySQL ネイティブ関数を使用した Lambda 関数の呼び出し

> **Note**
>
> Aurora MySQL バージョン 2 または Aurora MySQL バージョン 3.01 以降を使用しているときには、ネイティブ関数 `lambda_sync` および `lambda_async` を呼び出すことができます。Aurora MySQL のバージョンの詳細については、「Amazon Aurora MySQL のデータベースエンジンの更新」を参照してください。

ネイティブ関数 `lambda_sync` および `lambda_async` を呼び出すことで、Aurora MySQL DB クラスターから AWS Lambda 関数を呼び出すことができます。このアプローチは、Aurora MySQL で実行しているデータベースを 他の AWS のサービスと統合するときに便利です。例えば、データベースの特定のテーブルに行が挿入されるたびに Amazon Simple Notification Service (Amazon SNS) を使用して通知を送信するような場合があります。

目次

- lambda_sync 関数のパラメータ
- lambda_sync 関数の例
- lambda_async 関数の構文
- lambda_async 関数のパラメータ
- lambda_async 関数の例
- トリガーを使用した Lambda 関数の呼び出し

ネイティブ関数を使用した Lambda 関数の呼び出し

`lambda_sync` 関数と `lambda_async` 関数は、Lambda 関数を同期的および非同期に呼び出す組み込みのネイティブ関数です。別のアクションに移る前に、Lambda 関数の結果を知る必要がある場合は、同期関数 `lambda_sync` を使用します。別のアクションに移る前に、呼び出した Lambda 関数の結果を知る必要がない場合は、非同期関数 `lambda_async` を使用します。

Aurora MySQL バージョン 3 でのロールの付与

Aurora MySQL バージョン 3 では、ネイティブ関数を呼び出すユーザーは `AWS_LAMBDA_ACCESS` ロールが付与されている必要があります。ユーザーにこのロールを付与するには、DB インスタンスに管理者ユーザーとして接続し、次のステートメントを実行します。

```
GRANT AWS_LAMBDA_ACCESS TO user@domain-or-ip-address
```

次のステートメントを実行することで、このロールを取り消すことができます。

```
REVOKE AWS_LAMBDA_ACCESS FROM user@domain-or-ip-address
```

> **Tip**
>
> Aurora MySQL バージョン 3 でロールテクニックを使用する場合は、`SET ROLE role_name` または `SET ROLE ALL` ステートメントを使用してロールを有効化することもできます。MySQL 8.0 ロールシステムに馴染みがない場合は、ロールベースの特権モデルで詳細を確認頂けます。詳細については、「MySQL リファレンスマニュアル」の「Using roles」を参照してください。
> これは現在アクティブなセッションにのみ適用されます。再接続するときは、`SET ROLE` ステートメントを再度実行して権限を付与する必要があります。詳細については、MySQL リファレンスマニュアルの「SET ROLE ステートメント」を参照してください。

ユーザーが DB インスタンスに接続したときに、activate_all_roles_on_login DB クラスターパラメータを使用して、すべてのロールを自動的に有効化できます。このパラメータを設定すると、通常、SET ROLE ステートメントを明示的に呼び出してロールをアクティブ化する必要はありません。詳細については、MySQL リファレンスマニュアルの「activate_all_roles_on_login」を参照してください。
ただし、ストアドプロシージャを別のユーザーから呼び出す場合は、ストアドプロシージャの先頭で SET ROLE ALL を明示的に呼び出してロールをアクティブ化する必要があります。

Lambda 関数を呼び出そうとしたときに次のようなエラーが表示される場合は、SET ROLE ステートメントを実行します。

```
SQL Error [1227] [42000]: Access denied; you need (at least one of) the Invoke Lambda
 privilege(s) for this operation
```

Aurora MySQL バージョン 2 での権限の付与

Aurora MySQL バージョン 2 では、ネイティブ関数を呼び出すユーザーには INVOKE LAMBDA 権限が付与されている必要があります。ユーザーにこの権限を付与するには、DB インスタンスに管理者ユーザーとして接続し、次のステートメントを実行します。

```
GRANT INVOKE LAMBDA ON *.* TO user@domain-or-ip-address
```

次のステートメントを実行することで、この権限を取り消すことができます。

```
REVOKE INVOKE LAMBDA ON *.* FROM user@domain-or-ip-address
```

lambda_sync 関数の構文

lambda_sync 関数を RequestResponse 呼び出しタイプで同期的に呼び出します。関数は、JSON ペイロードで Lambda 呼び出しの結果を返します。関数の構文は次のとおりです。

```
lambda_sync (
  lambda_function_ARN,
  JSON_payload
)
```

lambda_sync 関数のパラメータ

`lambda_sync` 関数には以下のパラメータがあります。

lambda_function_ARN

呼び出す Lambda 関数の Amazon リソースネーム (ARN)。

JSON_payload

呼び出した Lambda 関数の JSON 形式の ペイロード

> **Note**
>
> Aurora MySQL バージョン 3 は MySQL 8.0 からの JSON 分析関数をサポートしています。ただし、Aurora MySQL バージョン 2 には、これらの関数は含まれていません。Lambda 関数が、数値や文字列など、アトミック値を返す場合、JSON 分析は必要ありません。

lambda_sync 関数の例

`lambda_sync` に基づく次のクエリは、関数 ARN を使用して Lambda 関数 `BasicTestLambda` を同期的に呼び出します。関数のペイロードは `{"operation": "ping"}` です。

```
SELECT lambda_sync(
    'arn:aws:lambda:us-east-1:123456789012:function:BasicTestLambda',
    '{"operation": "ping"}');
```

lambda_async 関数の構文

`lambda_async` 関数を `Event` 呼び出しタイプで非同期に呼び出します。関数は、JSON ペイロードで Lambda 呼び出しの結果を返します。関数の構文は次のとおりです。

```
lambda_async (
  lambda_function_ARN,
  JSON_payload
)
```

lambda_async 関数のパラメータ

`lambda_async` 関数には以下のパラメータがあります。

lambda_function_ARN

呼び出す Lambda 関数の Amazon リソースネーム (ARN)。

JSON_payload

呼び出した Lambda 関数の JSON 形式の ペイロード

> **Note**
>
> Aurora MySQL バージョン 3 は MySQL 8.0 からの JSON 分析関数をサポートしています。ただし、Aurora MySQL バージョン 2 には、これらの関数は含まれていません。Lambda 関数が、数値や文字列など、アトミック値を返す場合、JSON 分析は必要ありません。

lambda_async 関数の例

`lambda_async` に基づく次のクエリは、関数 ARN を使用して Lambda 関数 `BasicTestLambda` を非同期的に呼び出します。関数のペイロードは `{"operation": "ping"}` です。

```
SELECT lambda_async(
    'arn:aws:lambda:us-east-1:123456789012:function:BasicTestLambda',
    '{"operation": "ping"}');
```

トリガーを使用した Lambda 関数の呼び出し

トリガーを使用して、データ変更ステートメントで Lambda を呼び出すことができます。次の例では、`lambda_async` ネイティブ関数を使用して変数に結果を保存します。

```
mysql>SET @result=0;
mysql>DELIMITER //
mysql>CREATE TRIGGER myFirstTrigger
      AFTER INSERT
          ON Test_trigger FOR EACH ROW
      BEGIN
      SELECT lambda_async(
          'arn:aws:lambda:us-east-1:123456789012:function:BasicTestLambda',
          '{"operation": "ping"}')
          INTO @result;
      END; //
```

```
mysql>DELIMITER ;
```

> **Note**
>
> トリガーは SQL ステートメントごとに 1 回実行されるのではなく、行ごとに 1 回、一度に 1 行ずつ変更されます。トリガーが実行されると、プロセスは同期されます。data-modifying ステートメントは、トリガーが完了したときにのみ返されます。
> 書き込みトラフィックが多いテーブルのトリガーから AWS Lambda 関数を呼び出すときは注意してください。INSERT、UPDATE、DELETE のトリガーは行ごとにアクティブになります。INSERT、UPDATE、または DELETE トリガーがあるテーブルの書き込み負荷が高いと、AWS Lambda 関数の大量の呼び出しが発生します。

Aurora MySQL ストアドプロシージャを使用した Lambda 関数の呼び出し (非推奨)

mysql.lambda_async プロシージャを呼び出すことで、Aurora MySQL DB クラスターから AWS Lambda 関数を呼び出すことができます。このアプローチは、Aurora MySQL で実行しているデータベースを 他の AWS のサービスと統合するときに便利です。例えば、データベースの特定のテーブルに行が挿入されるたびに Amazon Simple Notification Service (Amazon SNS) を使用して通知を送信するような場合があります。

目次

Aurora MySQL バージョンに関する考慮事項

Aurora MySQL バージョン 2 から、これらのストアドプロシージャの代わりにネイティブ関数メソッドを使用して Lambda 関数を呼び出すことができます。ネイティブ関数の詳細については、「ネイティブ関数を使用した Lambda 関数の呼び出し」を参照してください。

Aurora MySQL バージョン 2 では、ストアドプロシージャ mysql.lambda_async はサポートされなくなりました。代わりに、ネイティブの Lambda 関数を使用することを強くお勧めします。

Aurora MySQL バージョン 3 では、ストアドプロシージャは使用できません。

mysql.lambda_async プロシージャを使用した Lambda 関数の呼び出し (非推奨)

`mysql.lambda_async` プロシージャは、Lambda 関数を非同期的に呼び出す組み込みストアドプロシージャです。このプロシージャを使用するには、EXECUTE ストアドプロシージャに対する `mysql.lambda_async` 権限がデータベースユーザーに必要です。

構文

`mysql.lambda_async` プロシージャの構文は次のとおりです。

```
CALL mysql.lambda_async (
  lambda_function_ARN,
  lambda_function_input
)
```

パラメータ

`mysql.lambda_async` プロシージャには以下のパラメータがあります。

lambda_function_ARN

呼び出す Lambda 関数の Amazon リソースネーム (ARN)。

lambda_function_input

呼び出した Lambda 関数の入力文字列 (JSON 形式)。

例

ベストプラクティスとして、`mysql.lambda_async` プロシージャへの呼び出しはストアドプロシージャでラップし、トリガーやクライアントコードなどのさまざまな出典から呼び出せるようにすることをお勧めします。このアプローチにより、インピーダンス不整合の問題を回避し、Lambda 関数を簡単に呼び出せるようになります。

> **Note**
>
> 書き込みトラフィックが多いテーブルのトリガーから AWS Lambda 関数を呼び出すときは注意してください。INSERT、UPDATE、DELETE のトリガーは行ごとにアクティブになります。INSERT、UPDATE、または DELETE トリガーがあるテーブルの書き込み負荷が高いと、AWS Lambda 関数の大量の呼び出しが発生します。

mysql.lambda_async プロシージャの呼び出しは非同期ですが、トリガーは同期です。大量のトリガーのアクティベーションを発生させるステートメントは、AWS Lambda 関数の呼び出しが完了するのを待機しませんが、クライアントに制御を返す前にトリガーが完了するのを待ちます。

Example 例: AWS Lambda 関数を呼び出して E メールを送信する

次の例では、データベースコードで呼び出せるストアドプロシージャを作成し、Lambda 関数を使用して E メールを送信します。

AWS Lambda 機能

```
import boto3

ses = boto3.client('ses')

def SES_send_email(event, context):

    return ses.send_email(
        Source=event['email_from'],
        Destination={
            'ToAddresses': [
            event['email_to'],
            ]
        },

        Message={
            'Subject': {
            'Data': event['email_subject']
            },
            'Body': {
                'Text': {
                    'Data': event['email_body']
                }
            }
        }
    )
```

ストアドプロシージャ

```
DROP PROCEDURE IF EXISTS SES_send_email;
```

```
DELIMITER ;;
  CREATE PROCEDURE SES_send_email(IN email_from VARCHAR(255),
                                  IN email_to VARCHAR(255),
                                  IN subject VARCHAR(255),
                                  IN body TEXT) LANGUAGE SQL
  BEGIN
    CALL mysql.lambda_async(
         'arn:aws:lambda:us-west-2:123456789012:function:SES_send_email',
         CONCAT('{"email_to" : "', email_to,
             '", "email_from" : "', email_from,
             '", "email_subject" : "', subject,
             '", "email_body" : "', body, '"}')
     );
  END
  ;;
DELIMITER ;
```

ストアドプロシージャを呼び出して AWS Lambda 関数を呼び出す

```
mysql> call SES_send_email('example_from@amazon.com', 'example_to@amazon.com', 'Email
 subject', 'Email content');
```

Example 例: AWS Lambda 関数を呼び出してトリガーからイベントを発行する

次の例では、Amazon SNS を使用してイベントを発行するストアドプロシージャを作成 します。コードでは、行がテーブルに追加されると、トリガーからプロシージャを呼び出します。

AWS Lambda 機能

```
import boto3

sns = boto3.client('sns')

def SNS_publish_message(event, context):

    return sns.publish(
        TopicArn='arn:aws:sns:us-west-2:123456789012:Sample_Topic',
        Message=event['message'],
        Subject=event['subject'],
        MessageStructure='string'
    )
```

ストアドプロシージャ

```
DROP PROCEDURE IF EXISTS SNS_Publish_Message;
DELIMITER ;;
CREATE PROCEDURE SNS_Publish_Message (IN subject VARCHAR(255),
                                      IN message TEXT) LANGUAGE SQL
BEGIN
  CALL mysql.lambda_async('arn:aws:lambda:us-
west-2:123456789012:function:SNS_publish_message',
     CONCAT('{ "subject" : "', subject,
            '", "message" : "', message, '" }')
     );
END
;;
DELIMITER ;
```

テーブル

```
CREATE TABLE 'Customer_Feedback' (
  'id' int(11) NOT NULL AUTO_INCREMENT,
  'customer_name' varchar(255) NOT NULL,
  'customer_feedback' varchar(1024) NOT NULL,
  PRIMARY KEY ('id')
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

[Trigger] (トリガー)

```
DELIMITER ;;
CREATE TRIGGER TR_Customer_Feedback_AI
  AFTER INSERT ON Customer_Feedback
  FOR EACH ROW
BEGIN
  SELECT CONCAT('New customer feedback from ', NEW.customer_name),
 NEW.customer_feedback INTO @subject, @feedback;
  CALL SNS_Publish_Message(@subject, @feedback);
END
;;
DELIMITER ;
```

通知をトリガーするには、テーブルに行を挿入します

```
mysql> insert into Customer_Feedback (customer_name, customer_feedback) VALUES ('Sample
 Customer', 'Good job guys!');
```

Amazon CloudWatch Logs への Amazon Aurora MySQL ログの発行

Aurora MySQL DB クラスターを設定して、全般ログ、スローログ、監査ログおよびエラーログのデータを Amazon CloudWatch Logs のロググループに発行できます。CloudWatch Logs を使用すると、ログデータのリアルタイム分析や、CloudWatch を使用したアラームの作成、メトリクスの表示を行うことができます。CloudWatch Logs を使用して、耐久性の高いストレージにログレコードを格納できます。

ログを CloudWatch Logs に発行するには、それぞれのログを有効にする必要があります。エラーログはデフォルトで有効になっていますが、他のタイプのログは明示的に有効にする必要があります。MySQL でログを有効にする方法については、MySQL ドキュメントの「一般クエリログおよびスロークエリログの出力先の選択」を参照してください。Aurora MySQL 監査ログを有効にする方法については、「アドバンストな監査の有効化」を参照してください。

> **Note**
>
> - Aurora は、無効化された監査ログデータをエクスポートする場合に、既存のロググループまたはログストリームを削除しません。既存の監査ログデータが無効化されている場合、既存のログは保持期間により、CloudWatch Logs で使用可能となり、保管された監査ログデータに変更を加えることもできます。ログストリーミングとロググループは、CloudWatch Logs コンソール、AWS CLI または CloudWatch Logs API を使用して削除できます。
> - CloudWatch Logs に監査ログを発行する別の方法は、[高度な監査] を有効にし、カスタウ DB クラスターパラメータグループを作成して、`server_audit_logs_upload` パラメータを 1 に設定することです。`server_audit_logs_upload` DB クラスターパラメータのデフォルト値は `0` です。[高度な監査] を有効にする方法については、「Amazon Aurora MySQL DB クラスターでのアドバンストな監査の使用」を参照してください。
>
>   この代替メソッドを使用する場合、CloudWatch Logs にアクセスして `aws_default_logs_role` クラスターレベルパラメータをこのロールの ARN に設定するには、IAM ロールが必要です。ロールの作成の詳細については、「AWS のサービスにアクセスするための IAM ロールの設定」を参照してください。ただし、`AWSServiceRoleForRDS` サービスにリンクされたロールがある場合、CloudWatch Logs へのアクセスが提供され、カスタム定義のロールが上書きされます。Amazon RDS のサービスにリンクされたロールの詳細については、「Amazon Aurora のサービスにリンクされたロールの使用」を参照してください。

- 監査ログを CloudWatch Logs にエクスポートしない場合は、監査ログをエクスポートするすべてのメソッドが無効になっていることを確認してください。これらのメソッドは、AWS Management Console、AWS CLI、RDS API、および `server_audit_logs_upload` パラメータです。
- Aurora Serverless v1 DB クラスターの手順は、プロビジョンドインスタンスや Aurora Serverless v2 インスタンスを含む DB クラスターの手順とは少し異なります。Aurora Serverless v1 クラスターでは、設定パラメータによって有効にしたすべてのログを自動的にアップロードします。

  したがって、Aurora Serverless v1 DB クラスターでログのアップロードを有効または無効にするには、DB クラスターのパラメータグループでログタイプ別にオンとオフを切り替えます。AWS Management Console、AWS CLI、または RDS API でクラスター自体の設定は変更しません。Aurora Serverless v1 クラスターの MySQL ログのオンとオフの詳細については、「Aurora Serverless v1 のパラメータグループ」を参照してください。

コンソール

コンソールを使用して、プロビジョンドクラスターの Aurora MySQL ログを CloudWatch Logs に発行できます。

コンソールから Aurora MySQL ログを発行するには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. ログデータを公開する Aurora MySQL DB クラスターを選択します。
4. Modify を選択します。
5. [ログのエクスポート] セクションで、CloudWatch Logs に公開するログを選択します。
6. [続行] を選択し、概要ページで [Modify DB Cluster (DB クラスターの変更)] を選択します。

AWS CLI

AWS CLI を使用して、プロビジョンドクラスターの Aurora MySQL ログを公開できます。これを行うには、以下のオプションを指定して modify-db-cluster AWS CLI コマンドを実行します。

- `--db-cluster-identifier`— DB クラスター識別子。

- `--cloudwatch-logs-export-configuration` — DB クラスターの CloudWatch Logs へのエクスポートに使用できるログタイプの構成設定。

以下の AWS CLI コマンドのいずれかを実行することで、Aurora MySQL ログを公開することもできます。

- create-db-cluster
- restore-db-cluster-from-s3
- restore-db-cluster-from-snapshot
- restore-db-cluster-to-point-in-time

以下のオプションを使用して、この AWS CLI コマンドの 1 つを実行します。

- `--db-cluster-identifier`— DB クラスター識別子。
- `--engine` — データベースエンジン。
- `--enable-cloudwatch-logs-exports` — DB クラスターの CloudWatch Logs へのエクスポートに使用できるログタイプの構成設定。

実行する AWS CLI コマンドに応じて、他のオプションが必要となる場合があります。

Example

次のコマンドでは、ログファイルが CloudWatch Logs に発行されるよう既存の Aurora MySQL DB クラスターを変更します。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["error","general","slowquery","audit"]}'
```

Windows の場合:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
```

```
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["error","general","slowquery","audit"]}'
```

Example

次のコマンドでは、ログファイルが CloudWatch Logs に発行されるよう Aurora MySQL DB クラスターを作成します。

Linux、macOS、Unix の場合:

```
aws rds create-db-cluster \
    --db-cluster-identifier mydbcluster \
    --engine aurora \
    --enable-cloudwatch-logs-exports '["error","general","slowquery","audit"]'
```

Windows の場合:

```
aws rds create-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --engine aurora ^
    --enable-cloudwatch-logs-exports '["error","general","slowquery","audit"]'
```

RDS API

RDS API を使用して、プロビジョンドクラスターの Aurora MySQL ログを発行できます。これを行うには、以下のオプションを指定して ModifyDBCluster オペレーションを実行します。

- `DBClusterIdentifier`— DB クラスター識別子。
- `CloudwatchLogsExportConfiguration` — DB クラスターの CloudWatch Logs へのエクスポートに使用できるログタイプの構成設定。

以下の RDS API オペレーションのいずれかを実行することで、RDS API を使用して Aurora MySQL ログを発行することもできます。

- CreateDBCluster
- RestoreDBClusterFromS3
- RestoreDBClusterFromSnapshot

- RestoreDBClusterToPointInTime

次のパラメータを指定して、RDS API オペレーションを実行します。

- `DBClusterIdentifier`— DB クラスター識別子。
- `Engine` — データベースエンジン。
- `EnableCloudwatchLogsExports` — DB クラスターの CloudWatch Logs へのエクスポートに使用できるログタイプの構成設定。

実行する AWS CLI コマンドに応じて、他のパラメータが必要となる場合があります。

Amazon CloudWatch でログイベントをモニタリングする

Aurora MySQL ログイベントを有効にすると、Amazon CloudWatch Logs でイベントをモニタリングできます。新しいクラスターロググループは、*`cluster-name`* が DB クラスター名となり、*`log_type`* がログタイプとなる次のプレフィックスの Aurora DB クラスターに自動的に作成されます。

```
/aws/rds/cluster/cluster-name/log_type
```

例えば、エクスポート関数を設定して、`mydbcluster` という名前の DB クラスターのスロークエリログを作成すると、スロークエリデータは、`/aws/rds/cluster/mydbcluster/slowquery` ロググループのスロークエリログストリーミングに保存されます。

クラスターのすべてのインスタンスにおけるイベントは、異なるログストリーミングを使用して、ロググループにプッシュされます。この動作は、次の条件のうちのどちらが true であるかによって異なります。

- 指定された名前のロググループが存在する。

  Aurora は既存のロググループを使用して、クラスターにログデータをエクスポートします。事前定義されたログ保持期間、メトリクスフィルター、カスタムアクセスを持つロググループを作成するために、AWS CloudFormationのような自動設定を使用できます。
- 指定された名前のロググループが存在しない。

  一致するログエントリがインスタンスのログファイルで検出されると、Aurora MySQL は CloudWatch Logs に新しいロググループを自動的に作成します。ロググループは、失効しないデフォルトのログ保持期間を使用します。

ログの保持期間を変更するには、CloudWatch Logs コンソール、AWS CLI、または CloudWatch Logs API を使用します。CloudWatch Logs でログの保持期間を変更する方法の詳細については、「CloudWatch Logs でのログデータ保管期間の変更」を参照してください。

DB クラスターのログイベント内の情報を検索するには、CloudWatch Logs コンソール、AWS CLI、または CloudWatch Logs API を使用します。検索およびログデータのフィルタ処理の詳細については、「ログデータの検索およびフィルタ処理」を参照してください。

Amazon Aurora MySQL ラボモード

Aurora ラボモードは、現在の Aurora データベースバージョンで使用できるが、デフォルトでは有効でない Aurora 機能を有効にするために使用されます。本番稼働用 DB クラスターに Aurora ラボモード機能を使用することは推奨されませんが、Aurora ラボモードを使用して開発環境とテスト環境の DB クラスターに対してこれらの機能を有効にすることができます。Aurora ラボモードが有効なときに利用できる Aurora 機能の詳細については、「Aurora ラボ モードの機能」を参照してください。

`aurora_lab_mode` パラメーターはインスタンスレベルのパラメーターであり、デフォルトのクラスターパラメーターグループにあります。デフォルトパラメータグループでは、パラメータは `0` (無効) に設定されます。Aurora ラボモードを有効にするには、カスタムパラメータグループを作成して、`aurora_lab_mode` パラメータをカスタムパラメータグループの 1 (有効) に設定し、カスタムパラメータグループを使用するように、Aurora クラスターの 1 つ以上の DB インスタンスを変更します。次に、ラボモード機能を試用するのに適切なインスタンスのエンドポイントに接続します。DB パラメータグループの変更については、「DB パラメータグループのパラメータの変更」を参照してください。パラメータグループおよび Amazon Aurora の詳細については、「Aurora MySQL 設定パラメータ」を参照してください。

Aurora ラボ モードの機能

次の表は、Aurora ラボモードが有効なときに現時点で利用できる Aurora 機能の一覧です。これらの機能を使用する前に、Aurora ラボモードを有効にする必要があります。

機能	説明
スキャンバッチ処理	Aurora MySQL のスキャンバッチ処理により、インメモリ、スキャン指向のクエリが大幅に高速になります。この機能を使用すると、バッチ処理によりテーブルフルスキャン、インデックスフルスキャン、インデックス範囲スキャンのパフォーマンスが向上します。
ハッシュ結合	等価結合を使用して大量のデータを結合する必要がある場合は、この機能によりクエリのパフォーマンスが向上することがあります。バージョン 2 では、この機能を Aurora MySQL ラボモードなしで使用できます。この機能の使用

機能	説明
	方法については、「ハッシュ結合を使用した大規模な Aurora MySQL 結合クエリの最適化」を参照してください。
高速 DDL	この機能を使用すると、`ALTER TABLE tbl_name ADD COLUMN col_name column_definition` オペレーションをほぼ瞬時に実行できます。このオペレーションでは、テーブルをコピーする必要はありません。他の DML ステートメントに実質的な影響を及ぼすこともありません。テーブルのコピーにはテンポラリストレージを使用しないため、スモールインスタンスクラスの大きなテーブルに対しても、DDL ステートメントが現実的になります。高速 DDL は現在、デフォルトの値を持たない、null が許容される列を、表の末尾に追加する際にのみ使用できます。この機能の使用方法については、「高速 DDL を使用して Amazon Aurora のテーブルを変更する」を参照してください。

Amazon Aurora MySQL を使用する際のベストプラクティス

このトピックには、Amazon Aurora MySQL DB クラスターの使用およびデータ移行のベストプラクティスとオプションに関する情報が含まれます。このトピックの情報は、Amazon Aurora DB クラスターの管理 にあるガイドラインや手順を一部要約し、改めて説明したものです。

目次

接続先の DB インスタンスの確認

Aurora MySQL DB クラスター内のどの DB インスタンスに接続しているかを確認するには、次の例に示すように、innodb_read_only グローバル可変をチェックします。

```
SHOW GLOBAL VARIABLES LIKE 'innodb_read_only';
```

リーダー DB インスタンスに接続している場合、innodb_read_only 可変が ON に設定されます。プロビジョニングされたクラスターのプライマリインスタンスなどのライター DB インスタンスに接続している場合、この設定は OFF です。

この方法は、ワークロードを分散させるロジックや、書き込みオペレーションで適切な接続が使用されているかを確認するロジックを、アプリケーションコードに追加する場合に便利です。

Aurora MySQL のパフォーマンスとスケーリングのためのベストプラクティス

次のベストプラクティスを適用して、Aurora MySQL クラスターのパフォーマンスとスケーラビリティを向上させることができます。

トピック

- 開発やテストのための T インスタンスクラスの使用
- Asynchronous Key Prefetch を使用した Aurora MySQL インデックス付き結合クエリの最適化
- ハッシュ結合を使用した大規模な Aurora MySQL 結合クエリの最適化
- Amazon Aurora を使用した MySQL データベースの読み取りスケーリング
- タイムスタンプ操作の最適化

開発やテストのための T インスタンスクラスの使用

db.t2、db.t3、または db.t4g DB インスタンスクラスを使用する Amazon Aurora MySQL インスタンスは、長い時間大量のワークロードをサポートしないアプリケーションに最適です。T インスタンスは、適度なベースラインパフォーマンスを実現したり、ワークロードの必要に応じて非常に高いパフォーマンスまでバーストする機能を実現できるように設計されています。常時または一貫して CPU をフルに使用するわけではないが、バーストが必要なことがあるワークロード向けに用意されています。T DB インスタンスクラスを、開発サーバーおよびテストサーバー、または他の本稼働以外のサーバーにのみ使用することをお勧めします。T インスタンスクラスの詳細については、「バーストパフォーマンスインスタンス」を参照してください。

Aurora クラスターが 40 TB より大きい場合は、T インスタンスクラスを使用しないでください。データベースに大量のデータがある場合、スキーマオブジェクトを管理するためのメモリオーバーヘッドが T インスタンスの容量を超えることがあります。

Amazon Aurora MySQL T インスタンスに対して MySQL パフォーマンススキーマを有効にしないでください。パフォーマンススキーマが有効な場合、インスタンスはメモリ不足になることがあります。

> **Tip**
>
> データベースがときにはアイドル状態になるが、それ以外では相当なワークロードがある場合は、T インスタンスの代替として Aurora Serverless v2 を使用できます。Aurora Serverless v2 では、容量範囲を定義すると、Aurora は現在のワークロードに応じてデータベースを自動的にスケールアップまたはスケールダウンします。使用方法の詳細については、「Aurora Serverless v2 を使用する」を参照してください。Aurora Serverless v2 で使用できるデータベースエンジンのバージョンについては、「Aurora Serverless v2 の要件と制限」を参照してください。

T インスタンスを Aurora MySQL DB クラスターの DB インスタンスとして使用する場合は、次のことをお勧めします。

- DB クラスター内のすべてのインスタンスに同じ DB インスタンスクラスを使用します。例えば、ライターインスタンスに `db.t2.medium` を使用する場合は、リーダーインスタンスにも `db.t2.medium` を使用することをお勧めします。
- メモリ関連の構成設定 (`innodb_buffer_pool_size` など) を調整しないでください。Aurora は、T インスタンスのメモリバッファに、高度に調整された一連のデフォルト値を使用します。これらの特殊なデフォルトは、メモリに制約のあるインスタンスで Aurora を実行するために必要です。T インスタンスでメモリ関連の設定を変更すると、バッファサイズを増やすための変更であっても、メモリ不足状態が発生する可能性が高くなります。
- CPU クレジットバランス (`CPUCreditBalance`) をモニタリングして、持続可能なレベルにあることを確認します。つまり、CPU のクレジットは使用されるのと同じレートで累積されています。

  インスタンス用の CPU クレジットが枯渇した場合、利用可能な CPU が急減するため、そのインスタンスに対する読み取りおよび書き込みレイテンシーが長くなります。この状況になると、インスタンス全体のパフォーマンスが大幅に低下します。

CPU クレジット残高が持続可能なレベルにない場合、サポートされているいずれかの R DB インスタンスクラスを使用するように DB インスタンスを変更すること (コンピューティングのスケーリング) をお勧めします。

モニタリングメトリクスの詳細については、「Amazon RDS コンソールでのメトリクスの表示」を参照してください。

- ライターインスタンスとリーダーインスタンスの間のレプリカラグ (AuroraReplicaLag) をモニタリングします。

  ライターインスタンスよりも前にリーダーインスタンスで CPU クレジットが枯渇した場合、結果として生じるラグにより、リーダーインスタンスが頻繁に再起動することがあります。このような結果になるのは一般的に、アプリケーション側で負荷の高い読み取り操作がリーダーインスタンス間に分散されるときに、ライターインスタンス側で書き込み操作の負荷が最小限に抑えられている場合です。

  レプリカラグの増加が持続している場合、DB クラスターのリーダーインスタンスの CPU クレジット残高が枯渇していないことを確認します。

  CPU クレジット残高が持続可能なレベルにない場合は、サポートされているいずれかの R DB インスタンスクラスを使用するように DB インスタンスを変更すること (コンピューティングのスケーリング) をお勧めします。

- バイナリログが有効な DB クラスターのトランザクションあたりの挿入の数を 100 万以下に維持します。

  DB クラスターの DB クラスターパラメータグループで binlog_format パラメータを OFF 以外の値に設定している場合、DB クラスターに 1,000,000 行以上の挿入を含むトランザクションがあると、DB クラスターでメモリが不足することがあります。解放可能なメモリ (FreeableMemory) メトリクスをモニタリングして、DB クラスターで使用可能なメモリが不足しているかどうかを判断できます。その後、書き込みオペレーション (VolumeWriteIOPS) メトリクスをモニタリングして、書き込みインスタンスで書き込みオペレーションの負荷が高いかどうかを確認します。メモリが不足し、書き込みオペレーションの負荷が高い場合は、トランザクションの挿入数を 100 万未満に制限するようにアプリケーションを更新することをお勧めします。または、サポートされているいずれかの R DB インスタンスクラスを使用するようにインスタンスを変更すること (コンピューティングのスケーリング) もできます。

Asynchronous Key Prefetch を使用した Aurora MySQL インデックス付き結合クエリの最適化

Aurora MySQL は Asynchronous Key Prefetch (AKP) を使用すると、インデックス間でテーブルを結合するクエリのパフォーマンスが向上することがあります。この機能は、JOIN クエリで Batched Key Access (BKA) 結合アルゴリズムと Multi-Range Read (MRR) 最適化機能が必要な場合、クエリの実行に必要な行を予測することで、パフォーマンスを向上させます。BKA と MRR の詳細については、MySQL ドキュメントの「Block Nested-Loop 結合と Batched Key Access 結合」および「Multi-Range Read の最適化」を参照してください。

AKP 機能を利用するには、クエリで BKA と MRR の両方を使用する必要があります。通常、このようなクエリは、クエリの JOIN 句でセカンダリインデックスを使用するが、プライマリインデックスからの一部の列を必要とする場合に発生します。例えば、JOIN 句が小さい外部テーブルと大きい内部テーブル間のインデックス値の等価結合を表し、大きいテーブルに対するインデックスの選択性が高い場合に、AKP を使用できます。AKP は、BKA および MRR と連携し、JOIN 句の評価時にセカンダリからプライマリへのインデックスのルックアップを行います。AKP は、JOIN 句の評価時にクエリの実行に必要な行を特定します。次に、バックグラウンドスレッドを使用して、クエリの実行前に、これらの行を含むページを非同期的にメモリ内にロードします。

AKP は、Aurora MySQL バージョン 2.10 以降、およびバージョン 3 でサポートされています。Aurora MySQL のバージョンの詳細については、「Amazon Aurora MySQL のデータベースエンジンの更新」を参照してください。

Asynchronous Key Prefetch の有効化

AKP 機能を有効にするには、MySQL サーバー可変 `aurora_use_key_prefetch` を `on` に設定します。デフォルトでは、この値は `on` に設定されます。ただし、BKA 結合アルゴリズムを有効にして、コストベースの MRR 機能を無効にするまでは、AKP を有効にすることはできません。そのためには、MySQL サーバー可変 `optimizer_switch` に以下の値を設定する必要があります。

- `batched_key_access` を `on` に設定します。この値は BKA 結合アルゴリズムの使用を制御します。デフォルトでは、この値は `off` に設定されます。
- `mrr_cost_based` を `off` に設定します。この値は、コストベースの MRR 機能の使用を制御します。デフォルトでは、この値は `on` に設定されます。

現在、これらの値はセッションレベルでのみ設定できます。次の例は、これらの値を設定し、SET ステートメントを実行して現在のセッションで AKP を有効にする方法を示しています。

```
mysql> set @@session.aurora_use_key_prefetch=on;
mysql> set @@session.optimizer_switch='batched_key_access=on,mrr_cost_based=off';
```

同様に、SET ステートメントを使用して AKP と BKA 結合アルゴリズムを無効にし、現在のセッションでコストベースの MRR 機能を再度有効にすることができます。次に例を示します。

```
mysql> set @@session.aurora_use_key_prefetch=off;
mysql> set @@session.optimizer_switch='batched_key_access=off,mrr_cost_based=on';
```

batched_key_access および mrr_cost_based オプティマイザスイッチの詳細については、MySQL ドキュメントの「切り替え可能な最適化の制御」を参照してください。

Asynchronous Key Prefetch のクエリの最適化

クエリで AKP 機能を利用できるかどうかを確認できます。そのためには、EXPLAIN ステートメントを使って、実行する前にクエリをプロファイリングします。EXPLAIN ステートメントは、指定されたクエリで使用する実行プランに関する情報を提供します。

EXPLAIN ステートメントの出力で、Extra 列は実行プランに含まれている追加情報を示します。AKP 機能の適用先がクエリで使用されているテーブルである場合、この列には次のいずれかの値が含まれます。

- `Using Key Prefetching`
- `Using join buffer (Batched Key Access with Key Prefetching)`

次の例では、EXPLAIN を使用することで、AKP を利用できるクエリの実行プランを表示しています。

```
mysql> explain select sql_no_cache
    ->     ps_partkey,
    ->     sum(ps_supplycost * ps_availqty) as value
    -> from
    ->     partsupp,
    ->     supplier,
    ->     nation
    -> where
    ->     ps_suppkey = s_suppkey
```

```
    ->     and s_nationkey = n_nationkey
    ->     and n_name = 'ETHIOPIA'
    -> group by
    ->     ps_partkey having
    ->         sum(ps_supplycost * ps_availqty) > (
    ->             select
    ->                 sum(ps_supplycost * ps_availqty) * 0.0000003333
    ->             from
    ->                 partsupp,
    ->                 supplier,
    ->                 nation
    ->             where
    ->                 ps_suppkey = s_suppkey
    ->                 and s_nationkey = n_nationkey
    ->                 and n_name = 'ETHIOPIA'
    ->         )
    -> order by
    ->     value desc;
+----+-------------+----------+------+-----------------------+---------------
+---------+-----------------------------------+------+----------
+-------------------------------------------------------------+
| id | select_type | table    | type | possible_keys         | key           | key_len
 | ref                               | rows | filtered | Extra
                            |
+----+-------------+----------+------+-----------------------+---------------
+---------+-----------------------------------+------+----------
+-------------------------------------------------------------+
|  1 | PRIMARY     | nation   | ALL  | PRIMARY               | NULL          | NULL
 | NULL                              |   25 |   100.00 | Using where; Using temporary;
 Using filesort                |
|  1 | PRIMARY     | supplier | ref  | PRIMARY,i_s_nationkey | i_s_nationkey | 5
 | dbt3_scale_10.nation.n_nationkey | 2057 |   100.00 | Using index
                            |
|  1 | PRIMARY     | partsupp | ref  | i_ps_suppkey          | i_ps_suppkey  | 4
 | dbt3_scale_10.supplier.s_suppkey |   42 |   100.00 | Using join buffer (Batched Key
 Access with Key Prefetching) |
|  2 | SUBQUERY    | nation   | ALL  | PRIMARY               | NULL          | NULL
 | NULL                              |   25 |   100.00 | Using where
                            |
|  2 | SUBQUERY    | supplier | ref  | PRIMARY,i_s_nationkey | i_s_nationkey | 5
 | dbt3_scale_10.nation.n_nationkey | 2057 |   100.00 | Using index
                            |
```

```
|  2 | SUBQUERY    | partsupp | ref  | i_ps_suppkey          | i_ps_suppkey  | 4
 | dbt3_scale_10.supplier.s_suppkey |   42 |   100.00 | Using join buffer (Batched Key
 Access with Key Prefetching) |
+----+-------------+----------+------+-----------------------+--------------
+---------+-----------------------------------+------+----------
+-------------------------------------------------------------+
6 rows in set, 1 warning (0.00 sec)
```

EXPLAIN 出力形式の詳細については、MySQL ドキュメントの「拡張 EXPLAIN 出力形式」を参照してください。

ハッシュ結合を使用した大規模な Aurora MySQL 結合クエリの最適化

等価結合を使用して大量のデータを結合する必要がある場合は、ハッシュ結合によりクエリのパフォーマンスが向上することがあります。Aurora MySQL に対してハッシュ結合を有効にすることができます。

ハッシュ結合列には任意の複合表現を使用できます。ハッシュ結合列では、以下のようなデータ型間での比較が可能です。

- `int`、`bigint`、`numeric`、`bit` などの厳密数値データ型のカテゴリ内で項目を比較できます。
- `float`、`double` などの近似数値データ型のカテゴリ内で項目を比較できます。
- 文字列型間で文字セットと照合が同じであれば、文字列型間で項目を比較できます。
- 日付およびタイムスタンプデータ型間で、型が同じあれば、項目を比較できます。

> **Note**
>
> 異なるカテゴリのデータ型を比較することはできません。

Aurora MySQL のハッシュ結合には、以下の制限が適用されます。

- 左右外部結合は、Aurora MySQL バージョン 2 ではサポートされていませんが、バージョン 3 ではサポートされています。
- サブクエリが初期にマテリアライズされない限り、サブクエリなどの準結合はサポートされていません。
- 複数テーブルの更新や削除はサポートされていません。

> **Note**
>
> 単一テーブルの更新や削除はサポートされていません。

- BLOB および空間データ型の列をハッシュ結合の結合列にすることはできません。

ハッシュ結合を有効にする

ハッシュ結合を有効にするには:

- Aurora MySQL バージョン 2 - DB パラメータまたは DB クラスターパラメータ `aurora_disable_hash_join` を `0` に設定します。`aurora_disable_hash_join` をオフにすると、`optimizer_switch` の値が `hash_join=on` に設定されます。
- Aurora MySQL バージョン 3 — MySQL サーバーパラメータ `optimizer_switch` を `block_nested_loop=on` に設定します。

ハッシュ結合は、Aurora MySQL バージョン 3 ではデフォルトで有効であり、Aurora MySQL バージョン 2 ではデフォルトで無効になっています。次の例は、Aurora MySQL バージョン 3 でハッシュ結合を有効にする方法を示しています。ステートメント `select @@optimizer_switch` をまず発行して、他にどのような設定が SET パラメータ文字列にあるか確認することができます。`optimizer_switch` パラメータの設定の一つを更新しても、他の設定は消去されたり修正されたりしません。

```
mysql> SET optimizer_switch='block_nested_loop=on';
```

> **Note**
>
> Aurora MySQL バージョン 3 では、ハッシュ結合サービスはすべてのマイナーバージョンで利用可能で、デフォルトで有効になっています。
> Aurora MySQL バージョン 2 の場合、ハッシュ結合サポートはすべてのマイナーバージョンで利用可能です。Aurora MySQL バージョン 2 の場合、ハッシュ結合機能は常に `aurora_disable_hash_join` の値によって制御されます。

この設定では、オプティマイザーはコスト、クエリの特徴、リソースの可用性に基づいてハッシュ結合を選択します。コスト見積りが正しくない場合に、オプティマイザーにハッシュ結合を選択させる

ことができます。そのためには、MySQL サーバー可変 hash_join_cost_based を off に設定します。以下の例に示しているのは、オプティマイザーにハッシュ結合を選択させる方法です。

```
mysql> SET optimizer_switch='hash_join_cost_based=off';
```

> **Note**
>
> この設定は、コストベースのオプティマイザの決定を上書きします。この設定はテストや開発に役立ちますが、本番環境で使用することは推奨されません。

ハッシュ結合のクエリの最適化

クエリでハッシュ結合を利用できるかどうかを調べるには、初期に EXPLAIN ステートメントを使用してクエリのプロファイリングを行います。EXPLAIN ステートメントは、指定されたクエリで使用する実行プランに関する情報を提供します。

EXPLAIN ステートメントの出力で、Extra 列は実行プランに含まれている追加情報を示します。クエリで使用するテーブルにハッシュ結合が適用される場合、この列には以下のような値が含まれます。

- `Using where; Using join buffer (Hash Join Outer table table1_name)`
- `Using where; Using join buffer (Hash Join Inner table table2_name)`

以下の例に示しているのは、EXPLAIN を使用してハッシュ結合クエリの実行プランを表示する方法です。

```
mysql> explain SELECT sql_no_cache * FROM hj_small, hj_big, hj_big2
    ->     WHERE hj_small.col1 = hj_big.col1 and hj_big.col1=hj_big2.col1 ORDER BY 1;
+----+-------------+----------+------+---------------+------+---------+------+------
+----------------------------------------------------------------+
| id | select_type | table    | type | possible_keys | key  | key_len | ref  | rows |
 Extra                                                          |
+----+-------------+----------+------+---------------+------+---------+------+------
+----------------------------------------------------------------+
|  1 | SIMPLE      | hj_small | ALL  | NULL          | NULL | NULL    | NULL |    6 |
 Using temporary; Using filesort                                |
```

```
|  1 | SIMPLE      | hj_big   | ALL  | NULL          | NULL | NULL    | NULL |   10 |
 Using where; Using join buffer (Hash Join Outer table hj_big)  |
|  1 | SIMPLE      | hj_big2  | ALL  | NULL          | NULL | NULL    | NULL |   15 |
 Using where; Using join buffer (Hash Join Inner table hj_big2) |
+----+-------------+----------+------+---------------+------+---------+------+------
+-----------------------------------------------------------------+
3 rows in set (0.04 sec)
```

出力では、`Hash Join Inner table` はハッシュテーブルの構築に使用されるテーブルであり、`Hash Join Outer table` はハッシュテーブルの検証に使用されるテーブルです。

拡張 EXPLAIN 出力形式の詳細については、MySQL 製品ドキュメントの「Extended EXPLAIN Output Format」(拡張 EXPLAIN 出力形式)を参照してください。

Aurora MySQL 2.08 以降では、SQL ヒントを使用して、クエリがハッシュ結合を使用するかどうか、および結合の構築側とプローブ側に使用するテーブルに影響を与えることができます。詳細については、「Aurora MySQL のヒント」を参照してください。

Amazon Aurora を使用した MySQL データベースの読み取りスケーリング

MySQL DB インスタンスで Amazon Aurora を使用することで、Amazon Aurora の読み取りスケーリング機能を活用して MySQL DB インスタンスの読み取りワークロードを拡張できます。Aurora を使用して MySQL DB インスタンスの読み取りを拡張するには、Aurora MySQL DB クラスターを作成し、MySQL DB インスタンスのリードレプリカに指定します。次に、Aurora MySQL クラスターに接続して読み取りクエリを処理します。出典データベースは、RDS for MySQL DB インスタンス、または Amazon RDS の外部で実行されている MySQL データベースです。詳細については、「Amazon Aurora を使用した MySQL データベースの読み取りスケーリング」を参照してください。

タイムスタンプ操作の最適化

システム変数 `time_zone` の値を `SYSTEM` に設定すると、タイムゾーン計算を必要とする各 MySQL 関数呼び出しは、システムライブラリ呼び出しを行います。このような `TIMESTAMP` 値を高い並行性で返したり変更したりする SQL ステートメントを実行すると、レイテンシー、ロック競合、および CPU 使用率が増加する可能性があります。詳細については、MySQL ドキュメントの「time_zone」を参照してください。

この現象を回避するには、`time_zone` DB クラスターパラメータの値を `UTC` に変更することをお勧めします。詳細については、「DB クラスターパラメータグループのパラメータの変更」を参照してください。

time_zone パラメータは動的 (データベースサーバーの再起動は不要) ですが、新しい値は新しい接続にのみ使用されます。すべての接続が新しい time_zone 値を使用するようにするには、DB クラスターパラメータを更新した後、アプリケーション接続をリサイクルすることをお勧めします。

Aurora MySQL の高可用性のベストプラクティス

次のベストプラクティスを適用して、Aurora MySQL クラスターの可用性を向上させることができます。

トピック

- Amazon Aurora を使用した MySQL データベースのディザスタリカバリ
- ダウンタイムを短縮して MySQL から Amazon Aurora MySQL に移行する
- Aurora MySQL DB インスタンスの低パフォーマンス、自動再起動、フェイルオーバーの回避

Amazon Aurora を使用した MySQL データベースのディザスタリカバリ

MySQL DB インスタンスで Amazon Aurora を使用することで、ディザスタリカバリ用のオフサイトバックアップを作成できます。MySQL DB インスタンスのディザスタリカバリに Aurora を使用するには、Amazon Aurora DB クラスターを作成し、MySQL DB インスタンスのリードレプリカに指定します。これは、RDS for MySQL DB インスタンス、または Amazon RDS の外部で実行されている MySQL データベースに適用されます。

> **Important**
>
> MySQL DB インスタンスと Amazon Aurora MySQL DB クラスターの間でレプリケーションを設定する場合、レプリケーションをモニタリングして、レプリケーションが正常に動作していることを確認し、必要な場合は修復する必要があります。

Amazon Aurora MySQL DB クラスターを作成して MySQL DB インスタンスのリードレプリカに指定するには、「Amazon Aurora を使用した MySQL データベースの読み取りスケーリング」の手順に従ってください。

災害対策モデルの詳細については、「Amazon Aurora MySQL クラスターに最適な災害対策オプションを選択する方法」を参照してください。

ダウンタイムを短縮して MySQL から Amazon Aurora MySQL に移行する

ライブアプリケーションをサポートする MySQL データベースから Amazon Aurora MySQL DB クラスターにデータをインポートするときは、移行中のサービス中断時間を短縮することが必要になる場合があります。これを行うには、『Amazon Relational Database Service ユーザーガイド』の「ダウンタイムを短縮して MySQL または MariaDB DB インスタンスにデータをインポートする」で説明されている手順を使用できます。この手順は、巨大なデータベースを使用する場合に特に役立ちます。この手順を使用すると、ネットワーク経由で AWS に渡されるデータの量を最小限に抑えることで、インポートのコストを削減できます。

このステップでは、データベースのデータのコピーを Amazon EC2 インスタンスに送信し、そのデータを新しい RDS for MySQL DB インスタンスにインポートするステップを示します。Amazon Aurora は MySQL と互換性があるため、ターゲット Amazon RDS MySQL DB インスタンスの代わりに Amazon Aurora DB クラスターを使用することができます。

Aurora MySQL DB インスタンスの低パフォーマンス、自動再起動、フェイルオーバーの回避

負荷の高いワークロードや、DB インスタンスに割り当てられたリソースを超えて急増するワークロードを実行している場合、アプリケーションと Aurora データベースを実行しているリソースを使い果たしてしまう可能性があります。CPU 使用率、メモリ使用量、使用されているデータベース接続数など、データベースインスタンスに関するメトリクスを取得するには、Amazon CloudWatch、パフォーマンスインサイト、および拡張モニタリングが提供するメトリクスを参照できます。DB インスタンスのモニタリングについては、「Amazon Aurora クラスターでのメトリクスのモニタリング」を参照してください。

ワークロードが使用しているリソースを使い果たした場合、DB インスタンスは遅くなったり、再起動したり、別の DB インスタンスにフェイルオーバーしたりする可能性があります。これを避けるには、リソースの使用状況を監視し、DB インスタンスで実行されているワークロードを調べ、必要に応じて最適化を行います。最適化を行ってもインスタンスのメトリクスが改善されず、リソースの枯渇も緩和されない場合は、上限に達する前に DB インスタンスをスケールアップすることを検討してください。利用可能な DB インスタンスクラスとその仕様の詳細については、「Aurora DB インスタンスクラス」を参照してください。

Aurora MySQL に関する推奨事項

Aurora MySQL では、MySQL との互換性のために次の機能が利用可能です。ただし、Aurora 環境では、パフォーマンス、スケーラビリティ、安定性、または互換性の問題があります。したがって、こ

れらの機能の使用については、特定のガイドラインに従うことをお勧めします。例えば、Aurora の本稼働デプロイには、特定の機能を使用しないことをお勧めします。

トピック

- Aurora MySQL でのマルチスレッドレプリケーションの使用
- ネイティブ MySQL 関数を使用した AWS Lambda 関数の呼び出し
- Amazon Aurora MySQL での XA トランザクションの回避
- DML ステートメント中に外部キーを有効にしておく
- ログバッファをフラッシュする頻度の設定
- Aurora MySQL デッドロックの最小化とトラブルシューティング

Aurora MySQL でのマルチスレッドレプリケーションの使用

マルチスレッドのバイナリログレプリケーションでは、SQL スレッドはリレーログからイベントを読み取り、SQL ワーカースレッドが適用されるようにキューに入れます。SQL ワーカースレッドは、コーディネータスレッドによって管理されます。バイナリログイベントは、可能な場合はパラレルに適用されます。

マルチスレッドレプリケーションは、Aurora MySQL バージョン 3 および Aurora MySQL バージョン 2.12.1 以降でサポートされています。

Aurora MySQL バージョン 3.04 以前では、Aurora MySQL DB クラスターがバイナリログレプリケーションのリードレプリカとして使用されている場合、Aurora はシングルスレッドレプリケーションをデフォルトで使用します。

Aurora MySQL バージョン 2 以前には、MySQL Community Edition から継承したマルチスレッドレプリケーションに関する不具合がいくつかあります。これらのバージョンでは、本番環境でのマルチスレッドレプリケーションの使用はお勧めしません。

マルチスレッドレプリケーションを使用する場合は、完全にテストした上で使用することをお勧めします。

Amazon Aurora におけるレプリケーションの使用の詳細については、「Amazon Aurora でのレプリケーション」を参照してください。Aurora MySQL でのマルチスレッドレプリケーションの詳細については、「マルチスレッドバイナリログレプリケーション」を参照してください。

ネイティブ MySQL 関数を使用した AWS Lambda 関数の呼び出し

ネイティブ MySQL 関数 `lambda_sync` および `lambda_async` を使用して、Lambda 関数を呼び出すことをお勧めします。

非推奨の `mysql.lambda_async` プロシージャを使用している場合は、`mysql.lambda_async` プロシージャの呼び出しをストアドプロシージャにラップすることをお勧めします。このストアドプロシージャは、トリガーやクライアントコードなどさまざまな出典から呼び出すことができます。この方法により、インピーダンス不整合の問題を回避し、データベースプログラマーが Lambda 関数を簡単に呼び出せるようにすることができます。

Amazon Aurora からの Lambda 関数の呼び出しについて詳細については、「Amazon Aurora MySQL DB クラスターからの Lambda 関数の呼び出し」を参照してください。

Amazon Aurora MySQL での XA トランザクションの回避

Aurora MySQL では eXtended Architecture (XA) トランザクションは使用しないことをお勧めします。これは、XA が PREPARED 状態の場合、復旧時間が長くなる可能性があるためです。Aurora MySQL で XA トランザクションを使用する必要がある場合は、以下のベストプラクティスに従ってください。

- XA トランザクションを PREPARED 状態で開いたままにしない。
- XA トランザクションを可能な限り小さくする。

MySQL で XA トランザクションを使用する方法の詳細については、MySQL ドキュメントの「XA トランザクション」を参照してください。

DML ステートメント中に外部キーを有効にしておく

`foreign_key_checks` 可変が `0` (オフ) に設定されている場合は、データ定義言語 (DDL) ステートメントを実行しないことを強くお勧めします。

外部キーの制約に一時的に違反する行を挿入または更新する必要がある場合は、以下のステップに従います。

1. `foreign_key_checks` を `0` に設定します。
2. データ操作言語 (DML) に変更を加えます。
3. 完了した変更が外部キーの制約に違反していないことを確認します。

4. foreign_key_checks を 1 (オン) に設定します。

さらに、外部キーの制約に関する以下のベストプラクティスに従います。

- クライアントアプリケーションが foreign_key_checks 可変の一部として 0 可変を init_connect に設定しないことを確認します。
- mysqldump などの論理的なバックアップからの復元が失敗するか、または不完全な場合は、同じセッションで他のオペレーションをスタートする前に、foreign_key_checks が 1 に設定されていることを確認します。論理的なバックアップのスタート時に foreign_key_checks が 0 に設定されています。

ログバッファをフラッシュする頻度の設定

MySQL Community Edition では、トランザクションを永続的にするには、InnoDB ログバッファを耐久性のあるストレージにフラッシュする必要があります。innodb_flush_log_at_trx_commit パラメータを使用して、ログバッファをディスクにフラッシュする頻度を設定します。

innodb_flush_log_at_trx_commit パラメータをデフォルト値の 1 に設定すると、トランザクションがコミットされるたびにログバッファがフラッシュされます。この設定は、データベースを ACID に準拠させるのに役立ちます。デフォルト設定の 1 を維持することをお勧めします。

innodb_flush_log_at_trx_commit をデフォルト以外の値に変更すると、データ操作言語 (DML) のレイテンシーを短縮できますが、ログレコードの耐久性は損なわれます。この耐久性の欠如により、データベースは ACID に準拠していません。サーバー再起動時にデータが損失するリスクを避けるため、データベースは ACID に準拠させることをお勧めします。このパラメータの詳細については、MySQL ドキュメントの「innodb_flush_log_at_trx_commit」を参照してください。

Aurora MySQL では、REDO ログ処理はストレージレイヤーにオフロードされるため、DB インスタンスではログファイルへのフラッシュは発生しません。書き込みが発行されると、REDO ログはライター DB インスタンスから Aurora クラスターボリュームに直接送信されます。ネットワークを介して行われる唯一の書き込みは REDO ログレコードです。データベース層からページが書き込まれることはありません。

デフォルトでは、トランザクションをコミットする各スレッドは、Aurora クラスターボリュームからの確認を待ちます。この確認は、このレコードとそれ以前のすべての REDO ログレコードが書き込まれ、クォーラムに達したことを示しています。ログレコードを永続化してクォーラムを達成すると、オートコミットでも明示的コミットでも、トランザクションが永続的になります。Aurora スト

レージアーキテクチャの詳細については、「Amazon Aurora ストレージのわかりやすい解説」を参照してください。

Aurora MySQL は MySQL コミュニティエディションのようにログをデータファイルにフラッシュしません。ただし、innodb_flush_log_at_trx_commit パラメータを使用すると、REDO ログレコードを Aurora クラスターボリュームに書き込む際の耐久性の制約を緩和できます。

Aurora MySQL バージョン 2 の場合:

- innodb_flush_log_at_trx_commit = 0 または 2 – データベースは REDO ログレコードが Aurora クラスターボリュームに書き込まれることを確認するまで待ちません。
- innodb_flush_log_at_trx_commit = 1 – データベースは REDO ログレコードが Aurora クラスターボリュームに書き込まれることを確認するまで待ちます。

Aurora MySQL バージョン 3 の場合:

- innodb_flush_log_at_trx_commit = 0 – データベースは REDO ログレコードが Aurora クラスターボリュームに書き込まれることを確認するまで待ちません。
- innodb_flush_log_at_trx_commit = 1 または 2 – データベースは REDO ログレコードが Aurora クラスターボリュームに書き込まれることを確認するまで待ちます。

したがって、Aurora MySQL バージョン 2 で 0 または 2 に設定された値と同じデフォルト以外動作を Aurora MySQL バージョン 3 で取得するには、パラメータを 0 に設定します。

これらの設定はクライアントの DML レイテンシーを低減できますが、フェイルオーバーまたは再起動の際にデータが失われる可能性もあります。したがって、innodb_flush_log_at_trx_commit パラメータはデフォルト値の 1 を維持することをお勧めします。

MySQL Community Edition と Aurora MySQL の両方でデータ損失が発生する可能性がありますが、アーキテクチャが異なるため、動作はデータベースごとに異なります。このようなアーキテクチャの違いにより、さまざまな程度のデータ損失が発生する可能性があります。データベースが ACID に準拠していることを確認するには、必ず innodb_flush_log_at_trx_commit を 1 に設定してください。

> **Note**
>
> Aurora MySQL バージョン 3 では、`innodb_flush_log_at_trx_commit` を 1 以外の値に変更する前に、まず `innodb_trx_commit_allow_data_loss` の値を 1 に変更する必要があります。これにより、データ損失のリスクを認識できます。

Aurora MySQL デッドロックの最小化とトラブルシューティング

同じデータページのレコードを同時に変更する場合、一意のセカンダリインデックスや外部キーに対する制約違反が定期的に発生するワークロードを実行していると、デッドロックやロック待機タイムアウトが増加する可能性があります。これらのデッドロックとタイムアウトは、MySQL Community Edition のバグ修正によるものです。

この修正は、MySQL Community Edition バージョン 5.7.26 以降に含まれており、Aurora MySQL バージョン 2.10.3 以降にバックポートされました。この修正は、InnoDB テーブルのレコードに加えられた変更に対して、これらのタイプのデータ操作言語 (DML) オペレーションに追加のロックを実装することで、直列化可能性を強制的に適用するために必要です。この問題は、以前の MySQL Community Edition のバグ修正によって生じたデッドロック問題の調査の一環として発見されました。

この修正により、InnoDB ストレージエンジンでのタプル (行) 更新の部分的なロールバックの内部処理が変更されました。外部キーまたは一意のセカンダリインデックスに制約違反が発生するオペレーションを行うと、部分的にロールバックが発生します。これには、同時 `INSERT...ON DUPLICATE KEY UPDATE`、`REPLACE INTO`,、`INSERT IGNORE` ステートメント (upserts) が含まれますが、これらに限定されません。

ここでいう部分ロールバックとは、アプリケーションレベルのトランザクションのロールバックを指すのではなく、制約違反が発生した場合に、内部 InnoDB が変更をクラスター化されたインデックスにロールバックすることを指します。例えば、upsert オペレーション中に重複するキー値が見つかったとします。

通常の挿入オペレーションでは、InnoDB はインデックスごとにクラスター化されたインデックスエントリとセカンダリインデックスエントリをアトミックに作成します。InnoDB が upsert オペレーション中に一意のセカンダリインデックスで重複値を検出した場合、クラスター化されたインデックスに挿入されたエントリを元に戻し (部分ロールバック)、更新を既存の重複行に適用する必要があります。この内部部分ロールバックステップ中は、InnoDB はオペレーションの一部と見なされる各レコードをロックする必要があります。この修正により、部分ロールバック後に追加のロックを導入することにより、トランザクションの直列化可能性が保証されます。

InnoDB デッドロックの最小化

データベースインスタンスのデッドロックの発生頻度を減らすには、次の方法を使用できます。その他の例は、MySQL のドキュメントにあります。

1. デッドロックの可能性を削減するために、関連する一連の変更を行った直後にトランザクションをコミットしてください。これを行うには、大きなトランザクション (コミット間の複数行の更新) を小さなトランザクションに分割します。行をバッチ挿入する場合は、特に前述の upsert オペレーションを使用するときは、バッチ挿入のサイズを小さくします。

   部分的なロールバックの頻度を削減するために、次の複数の方法を試行できます。

   a. バッチ挿入オペレーションを、一度に 1 行ずつ挿入するオペレーションに置き換えます。これにより、競合が発生する可能性のあるトランザクションによって、ロックが保持される時間を削減できます。

   b. `REPLACE INTO` を使用する代わりに、SQL ステートメントを次のような複数ステートメントのトランザクションとして書き換えます。

   ```
   BEGIN;
   DELETE conflicting rows;
   INSERT new rows;
   COMMIT;
   ```

   c. `INSERT...ON DUPLICATE KEY UPDATE` を使用する代わりに、SQL ステートメントを次のような複数ステートメントのトランザクションとして書き換えます。

   ```
   BEGIN;
   SELECT rows that conflict on secondary indexes;
   UPDATE conflicting rows;
   INSERT new rows;
   COMMIT;
   ```

2. アクティブまたはアイドルで長時間稼働するトランザクションは、ロックを保持する可能性があるので避けてください。これには、コミットされていないトランザクションで、長期間開かれている可能性のあるインタラクティブな MySQL クライアントセッションが含まれます。トランザクションサイズまたはバッチサイズを最適化する場合、同時実行性、重複数、テーブル構造などのさまざまな要因で、影響が異なる可能性があります。どのような変更でも、ワークロードに基づいて実装し、テストする必要があります。

3. 状況によっては、2 つのトランザクションが 1 つまたは複数のテーブル内の同じデータセットに異なる順序でアクセスしようとすると、デッドロックが発生することがあります。これを防止す

るには、同じ順序でデータにアクセスするようにトランザクションを変更して、アクセスをシリアル化できます。例えば、完了するトランザクションのキューを作成します。このアプローチでは、複数のトランザクションが同時に発生する場合、デッドロックを回避するのに役立ちます。

4. インデックスを慎重に選択してテーブルに追加することで、選択性が向上し、行にアクセスする必要性が減り、ロックが減少します。
5. ギャップロックが発生した場合は、セッションまたはトランザクションのトランザクション分離レベルを `READ COMMITTED` に変更することで、ギャップロックを防止できます。InnoDB 分離レベルとその動作の詳細については、MySQL ドキュメントの「トランザクション分離レベル」を参照してください。

> **Note**
>
> デッドロックが発生する可能性を削減するための予防策を講じることはできますが、デッドロックはデータベースで想定される動作であり、発生する可能性はゼロにはなりません。アプリケーションには、デッドロックが発生した場合の対処に必要なロジックが必要です。例えば、アプリケーションに再試行とバックオフのロジックを実装します。問題の根本原因に対処するのが最善ですが、デッドロックが発生した場合、アプリケーションには待機後に再試行するオプションがあります。

InnoDB デッドロックのモニタリング

MySQL では、アプリケーションのトランザクションがテーブルレベルおよび行レベルのロックを取得しようすると循環待機になり、デッドロックが発生する可能性があります。InnoDB のデッドロックは、InnoDB ストレージエンジンによってすぐに状態を検出し、トランザクションの 1 つを自動的にロールバックするため、ときどき発生するデッドロックは必ずしも問題にはなりません。デッドロックが頻繁に発生する場合は、パフォーマンスの問題を軽減し、デッドロックを回避するために、アプリケーションを見直して修正することをお勧めします。デッドロック検出がオン (デフォルト) の場合、InnoDB はトランザクションのデッドロックを自動的に検出し、1 つまたは複数のトランザクションをロールバックしてデッドロックを解消します。InnoDB は小さなトランザクションを選択してロールバックしようとします。トランザクションのサイズは、挿入、更新、削除された行の数によって決まります。

- `SHOW ENGINE` ステートメント — `SHOW ENGINE INNODB STATUS \G` ステートメントには、前回の再起動以降にデータベースで発生した最新のデッドロックの詳細が含まれます。

- MySQL エラーログ — `SHOW ENGINE` ステートメントの出力が不十分なデッドロックが頻繁に発生する場合は、innodb_print_all_deadlocks DB クラスターパラメータを有効にできます。

  このパラメータを有効にすると、InnoDB ユーザートランザクションのすべてのデッドロックに関する情報が Aurora MySQL エラーログに記録されます。
- Amazon CloudWatch メトリクス — CloudWatch メトリクス `Deadlocks` を使用して、デッドロックを積極的にモニタリングすることもお勧めします。詳細については、「Amazon Aurora のインスタンスレベルのメトリクス」を参照してください。
- Amazon CloudWatch Logs — CloudWatch Logs を使用すると、メトリクスの表示、ログデータの分析、アラームのリアルタイム表示を行うことができます。詳細については、「Monitor errors in Amazon Aurora MySQL and Amazon RDS for MySQL using Amazon CloudWatch and send notifications using Amazon SNS」(Amazon CloudWatch を使用して Amazon Aurora MySQL と Amazon RDS for MySQL のエラーをモニタリングし、Amazon SNS を使用して通知を送信する) を参照してください。

  `innodb_print_all_deadlocks` を有効にした CloudWatch Logs を使用すると、デッドロックの回数が指定したしきい値を超えた場合に通知するようにアラームを設定できます。しきい値を定義するには、傾向を観察して、通常のワークロードに基づいた値を使用することをお勧めします。
- Performance Insights — Performance Insights を使用すると、`innodb_deadlocks` および `innodb_lock_wait_timeout` メトリクスをモニタリングできます。これらのメトリクスの詳細については、「Aurora MySQL の非ネイティブカウンター」を参照してください。

Amazon Aurora MySQL データベースのパフォーマンスのトラブルシューティング

このトピックでは、Aurora MySQL DB の一般的なパフォーマンス問題と、これらの問題を迅速に修正するためのトラブルシューティング方法または情報収集方法に焦点を当てます。データベースのパフォーマンスを次の 2 つのカテゴリに分類します。

- サーバーパフォーマンス — データベースサーバー全体の動作が遅い。
- クエリパフォーマンス — 1 つまたは複数のクエリの実行に時間がかかる。

AWS モニタリングオプション

トラブルシューティングに役立てるため、次の AWS モニタリングオプションを使用することをお勧めします。

- Amazon CloudWatch — Amazon CloudWatch は AWS で実行されている AWS リソースやアプリケーションをリアルタイムにモニタリングします。CloudWatch を使用してメトリクスを収集および追跡できます。メトリクスとは、リソースやアプリケーションについて測定できる変数です。詳細については、「Amazon CloudWatch とは」を参照してください。

  DB インスタンスのすべてのシステムメトリクスとプロセス情報を AWS Management Console に表示できます。Aurora MySQL DB クラスターを設定して、全般ログ、スローログ、監査ログおよびエラーログのデータを Amazon CloudWatch Logs のロググループに発行できます。これにより、傾向を確認したり、ホストが影響を受けた場合にログを維持したり、異常や変化を簡単に特定するための「通常の」パフォーマンスのベースラインを作成することができます。詳細については、「Amazon CloudWatch Logs への Amazon Aurora MySQL ログの発行」を参照してください。
- 拡張モニタリング — Aurora MySQL データベースに対して追加の Amazon CloudWatch メトリクスを有効にするには、拡張モニタリングを有効にします。Aurora DB クラスターを作成または変更するときは、[拡張モニタリングを有効化] を選択します。これにより、Aurora はパフォーマンスメトリクスを CloudWatch にパブリッシュできます。利用可能な主なメトリクスには、CPU 使用率、データベース接続、ストレージ使用量、クエリレイテンシーなどがあります。これらはパフォーマンスのボトルネックの特定に役立ちます。

  DB インスタンスに対して転送される情報量は、拡張モニタリング機能に対して定義された詳細度に正比例します。モニタリング間隔を短くすると、OS メトリクスのレポート回数が増え、モニタリングコストが高くなります。コストを管理するには、AWS アカウント 内のインスタンスごとに

異なる詳細度を設定します。インスタンス作成時のデフォルトの精度は 60 秒です。詳細については、「拡張モニタリングのコスト」を参照してください。

- Performance Insights — すべてのデータベースコールメトリクスを表示できます。これには DB ロック、待機、処理された行数などがあり、これらはすべてトラブルシューティングに使用できます。Aurora DB クラスターを作成または変更するときは、[Performance Insights を有効にする] を選択します。デフォルトでは、Performance Insights のデータ保持期間は 7 日間ですが、長期的なパフォーマンスの傾向を分析するようにカスタマイズできます。7 日を超える保存期間については、有料プランにアップグレードする必要があります。詳細については、「Performance Insights の料金」を参照してください。Aurora DB インスタンスごとにデータ保持期間を個別に設定できます。詳細については、「Amazon Aurora での Performance Insights を使用したDB 負荷のモニタリング」を参照してください。

Aurora MySQL データベースのパフォーマンス問題の最も一般的な原因

次の手順を使用して、Aurora MySQL データベースのパフォーマンス問題をトラブルシューティングできます。これらのステップは調査の論理的な順序で示していますが、直線的なものではありません。1 つの発見が複数のステップにまたがることもあり、それによって一連の調査パスが提供されます。

1. ワークロード — データベースのワークロードを理解します。
2. ログ記録 — すべてのデータベースログを確認します。
3. クエリパフォーマンス — クエリ実行プランが変更されていないか確認します。コードを変更すると、プランが変更される可能性があります。

Aurora MySQL データベースのワークロードに関する問題のトラブルシューティング

データベースワークロードは読み取りおよび書き込みとして見なすことができます。「通常の」データベースワークロードを理解していれば、需要の変化に合わせてクエリとデータベースサーバーを調整できます。パフォーマンスが変化する理由は多く存在するため、最初のステップは何が変わったのかを理解することです。

- メジャーバージョンまたはマイナーバージョンのアップグレードは行われましたか?

  メジャーバージョンアップグレードには、エンジンコード、特にオプティマイザの変更が含まれ、それによってクエリ実行プランが変更される可能性があります。データベースバージョン、特にメ

ジャーバージョンをアップグレードするときは、データベースのワークロードを分析し、それに応じて調整することが非常に重要です。調整には、テストの結果に応じて、クエリの最適化と書き換え、またはパラメータ設定の追加と更新が含まれる場合があります。何が影響を引き起こしているのかを理解することで、その特定の分野に集中できるようになります。

詳細については、MySQL ドキュメントの「MySQL 8.0 の新機能」と「MySQL 8.0 で追加、廃止、または削除されたサーバー変数とステータス変数とオプション」、および「Aurora MySQL バージョン 2 と Aurora MySQL バージョン 3 の比較」を参照してください。

- 処理中のデータ (行数) は増加しましたか?
- 同時に実行されているクエリは他にもありますか?
- スキーマやデータベースに変更はありますか?
- コードに欠陥や修正はありましたか?

目次

インスタンスホスト x メトリクス

CPU、メモリ、ネットワークアクティビティなどのインスタンスホストのメトリクスをモニタリングして、ワークロードが変更されたかどうかを把握します。ワークロードの変化を理解するには、主に 2 つの概念があります。

- 使用率 — CPU やディスクなどのデバイスの使用状況。時間ベースの場合とキャパシティベースの場合があります。
  - 時間ベース — 特定の観測期間にリソースがビジー状態である時間です。

- キャパシティベース — システムまたはコンポーネントが提供できるスループットの量 (キャパシティに対する割合)。
- 飽和度 — リソースで処理できる量よりも多くの作業が必要となる度合い。キャパシティベースの使用率が 100% に達すると、余分な作業は処理できなくなり、キューに入れる必要があります。

CPU の使用

次のツールを使用して、CPU の使用状況と飽和度を識別できます。

- CloudWatch が `CPUUtilization` メトリクスを提供します。この値が 100% に達すると、インスタンスは飽和状態になります。ただし、CloudWatch メトリクスは 1 分間で平均化され、詳細さに欠けています。

  CloudWatch のメトリクスの詳細については、「Amazon Aurora のインスタンスレベルのメトリクス」を参照してください。
- 拡張モニタリングは、オペレーティングシステム `top` コマンドによって返されるメトリクスを提供します。負荷平均と次の CPU 状態が 1 秒単位で表示されます。
  - `Idle (%)` = アイドル時間
  - `IRQ (%)` = ソフトウェア割り込み
  - `Nice (%)` = 優先順位が niced のプロセスの良好な時間。
  - `Steal (%)` = 他のテナントへのサービス提供に費やした時間 (仮想化関連)
  - `System (%)` = システム時刻
  - `User (%)` = ユーザー時間
  - `Wait (%)` = I/O 待機

  拡張モニタリングのメトリクスの詳細については、「Aurora の OS メトリクス」を参照してください。

メモリ使用量

システムがメモリ不足に陥っていて、リソース消費が飽和状態に達しつつある場合は、ページスキャン、ページング、スワップ、メモリ不足などのエラーが頻繁に発生しているはずです。

次のツールを使用して、メモリの使用状況と飽和度を識別できます。

CloudWatch は、一部の OS キャッシュと現在の空きメモリをフラッシュすることで再利用できるメモリを示す `FreeableMemory` メトリクスを提供します。

CloudWatch のメトリクスの詳細については、「Amazon Aurora のインスタンスレベルのメトリクス」を参照してください。

拡張モニタリングでは、メモリ使用量の問題を特定するのに役立つ以下のメトリクスが提供されます。

- `Buffers`（KB）- ストレージデバイスへの書き込み前に I/O バッファリングリクエストに使用されたメモリの量 (キロバイト単位)。
- `Cached`（KB）- ファイルシステムベースの I/O のキャッシュに使用されたメモリの量。
- `Free`（KB）- 未割り当てのメモリの量 (キロバイト単位)。
- `Swap` - キャッシュ、フリー、および合計。

例えば、DB インスタンスが `Swap` メモリを使用していることがわかった場合、ワークロードの合計メモリ量は、インスタンスで現在使用可能なメモリ量よりも多くなっています。DB インスタンスのサイズを増やすか、使用するメモリ量が少なくなるようにワークロードを調整することをお勧めします。

拡張モニタリングのメトリクスの詳細については、「Aurora の OS メトリクス」を参照してください。

パフォーマンススキーマと `sys` スキーマを使用して、どの接続やコンポーネントがメモリを使用しているかを見極める方法の詳細については、「Aurora MySQL データベースのメモリ使用量に関する問題のトラブルシューティング」を参照してください。

ネットワークスループット

CloudWatch は、ネットワークスループットの合計について、すべて 1 分間の平均値として次のメトリクスを提供します。

- `NetworkReceiveThroughput` - Aurora DB クラスター内の各インスタンスが各クライアントから受信したネットワークスループットの量。
- `NetworkTransmitThroughput` - Aurora DB クラスター内の各インスタンスが各クライアントに対して送信したネットワークスループットの量。
- `NetworkThroughput` - Aurora DB クラスター内の各インスタンスが各クライアントで送受信したネットワークスループットの量。
- `StorageNetworkReceiveThroughput` - DB クラスター内の各インスタンスが、Aurora のストレージサブシステムから受信した、ネットワークスループットの量。

- StorageNetworkTransmitThroughput - Aurora DB クラスター内の各インスタンスが、Aurora のストレージサブシステムに送信した、ネットワークスループットの量。
- StorageNetworkThroughput - Aurora DB クラスター内の各インスタンスが、Aurora のストレージサブシステムとの間で送受信した、ネットワークスループットの量。

CloudWatch のメトリクスの詳細については、「Amazon Aurora のインスタンスレベルのメトリクス」を参照してください。

拡張モニタリングでは、network が受信した (RX) および送信した (TX) グラフが最大 1 秒の精度で表示されます。

拡張モニタリングのメトリクスの詳細については、「Aurora の OS メトリクス」を参照してください。

データベースメトリクス

以下の CloudWatch メトリクスを調べて、ワークロードの変化を確認します。

- BlockedTransactions - 1 秒あたりのブロックされたデータベース内のトランザクションの平均数。
- BufferCacheHitRatio – バッファキャッシュから提供されたリクエストの割合 (パーセント)。
- CommitThroughput - 1 秒あたりのコミット操作の平均回数。
- DatabaseConnections - データベースインスタンスへのクライアントネットワーク接続の数。
- Deadlocks - 1 秒あたりのデータベース内のデッドロックの平均回数。
- DMLThroughput - 1 秒あたりの挿入、更新、削除の平均回数。
- ResultSetCacheHitRatio - クエリキャッシュから提供されたリクエストの割合 (パーセント)。
- RollbackSegmentHistoryListLength - コミットされたトランザクションが削除とマークされたレコードを記録する UNDO ログ。
- RowLockTime - InnoDB テーブルのローロックの取得にかかった合計時間。
- SelectThroughput - 1 秒あたりの選択クエリの平均回数。

CloudWatch のメトリクスの詳細については、「Amazon Aurora のインスタンスレベルのメトリクス」を参照してください。

ワークロードを調べる際には、以下の点を考慮してください。

1. DB インスタンスクラスに最近変更があったか。例えば、インスタンスサイズを 8xlarge から 4xlarge に減らしたり、db.r5 から db.r6 に変更したりしたか？
2. クローンを作成して問題を再現できますか? それとも 1 つのインスタンスでのみ発生していますか?
3. サーバーリソースの消耗、CPU 使用率の上昇、またはメモリの消耗は発生していますか?「はい」の場合は、ハードウェアの追加が必要な場合があります。
4. 1 つまたは複数のクエリに時間がかかりますか?
5. 変化の原因はアップグレード、特にメジャーバージョンアップグレードですか?「はい」の場合は、アップグレード前とアップグレード後のメトリクスを比較してください。
6. リーダー DB インスタンスの数に変更はありますか?
7. 一般ロギング、監査ロギング、またはバイナリロギングを有効にしましたか? 詳細については、「Aurora MySQL データベースのログ記録」を参照してください。
8. バイナリログ (binlog) レプリケーションの使用を有効化、無効化、または変更しましたか?
9. 長時間実行されるトランザクションで、多数の行ロックが発生していませんか? InnoDB 履歴リストの長さ (HLL) を調べて、トランザクションが長時間実行されているかどうかを確認してください。

   詳細については、「InnoDB 履歴リストの長さが大幅に増加しました」および「Amazon Aurora MySQL DB クラスターで SELECT クエリの実行が遅いのはなぜですか?」というブログ記事を参照してください。

   a. 書き込みトランザクションが原因で HLL が大きくなっている場合は、UNDO ログが蓄積されている (定期的にクリーンアップされていない) ことを意味します。書き込みトランザクションが多いと、この蓄積が急速に増加する可能性があります。MySQL では、UNDO は SYSTEM テーブルスペースに保存されます。SYSTEM テーブルスペースは圧縮できません。UNDO ログにより、SYSTEM テーブルスペースが数 GB、さらには TB にまで増えることもあります。削除後、データの論理バックアップ (ダンプ) を作成して割り当てられたスペースを解放し、そのダンプを新しい DB インスタンスにインポートします。

   b. 読み取りトランザクション (実行時間の長いクエリ) が原因で HLL が大きくなっている場合は、クエリが大量の一時スペースを使用している可能性があります。再起動して一時スペースを解放します。Temp セクションで Performance Insights の DB メトリクスに変更 (`created_tmp_tables` など) がないかを調べます。詳細については、「Amazon Aurora での Performance Insights を使用したDB 負荷のモニタリング」を参照してください。
10. 長時間実行されるトランザクションを、変更される行数が少ないより小さなトランザクションに分割できますか?

11.ブロックされたトランザクションの変化やデッドロックの増加はありませんか?
Locks セクションで Performance Insights の DB メトリクスにステータス変数の変更 (innodb_row_lock_time、 innodb_row_lock_waits、および innodb_dead_locks など) がないかを調べます。1 分または 5 分間隔を使用します。

12.増加した待機イベントがあるか？ Performance Insights の待機イベントと待機タイプを 1 分または 5 分間隔で調べます。上位の待機イベントを分析し、それらがワークロードの変化やデータベースの競合と相関関係があるかどうかを確認します。例えば、buf_pool mutex はバッファプールの競合を示しています。詳細については、「待機イベントを使用した Aurora MySQL のチューニング」を参照してください。

Aurora MySQL データベースのメモリ使用量に関する問題のトラブルシューティング

CloudWatch、拡張モニタリング、Performance Insights は、オペレーティングシステムレベルでのメモリ使用量 (データベースプロセスによるメモリ使用量など) の概要を提供しますが、エンジン内の接続やコンポーネント別のメモリ使用量を詳しく知ることはできません。

このトラブルシューティングには、パフォーマンススキーマと sys スキーマを使用できます。Aurora MySQL バージョン 3 では、パフォーマンススキーマを有効にすると、追加の計測がデフォルトで有効になります。Aurora MySQL バージョン 2 では、パフォーマンススキーマのメモリ使用量の計測のみがデフォルトで有効になります。パフォーマンススキーマでメモリ使用量を追跡するために使用できるテーブルと、パフォーマンススキーマのメモリ計測の有効化の詳細については、MySQL ドキュメントの「Memory summary tables」を参照してください。パフォーマンススキーマと Performance Insights の詳細については、「Aurora MySQL における Performance Insights の Performance Schema の有効化」を参照してください。

パフォーマンススキーマでは、現在のメモリ使用量を追跡するための詳細情報を参照できます。一方、MySQL の sys スキーマでは、パフォーマンススキーマテーブルの上部のビューで、どこでメモリが使用されているかをすばやく特定できます。

sys スキーマには、接続、コンポーネント、クエリ別にメモリ使用量を追跡できる以下のビューがあります。

ビュー	説明
memory_by_host_by_current_bytes	ホスト別にエンジンメモリ使用量に関する情報を表示します。どのアプリケーションサーバー

ビュー	説明
	またはクライアントホストがメモリを消費しているかを特定するのに役立ちます。
memory_by_thread_by_current_bytes	スレッド ID 別にエンジンメモリ使用量に関する情報を表示します。MySQL のスレッド ID は、クライアント接続またはバックグラウンドスレッドである場合があります。sys.processlist ビューまたは performance_schema.threads テーブルを使用して、スレッド ID を MySQL 接続 ID にマッピングできます。
memory_by_user_by_current_bytes	ユーザー別のエンジンメモリ使用量に関する情報を表示します。どのユーザーアカウントまたはクライアントがメモリを消費しているかを特定するのに役立ちます。
memory_global_by_current_bytes	エンジンコンポーネント別のエンジンメモリ使用量に関する情報を表示します。エンジンバッファまたはコンポーネント別にメモリ使用量をグローバルに特定するのに役立ちます。例えば、InnoDB バッファプールの `memory/innodb/buf_buf_pool` イベントやプリペアドステートメントの `memory/sql/Prepared_statement::main_mem_root` イベントが表示される場合があります。
memory_global_total	データベースエンジンで追跡している合計メモリ使用量の概要を表示します。

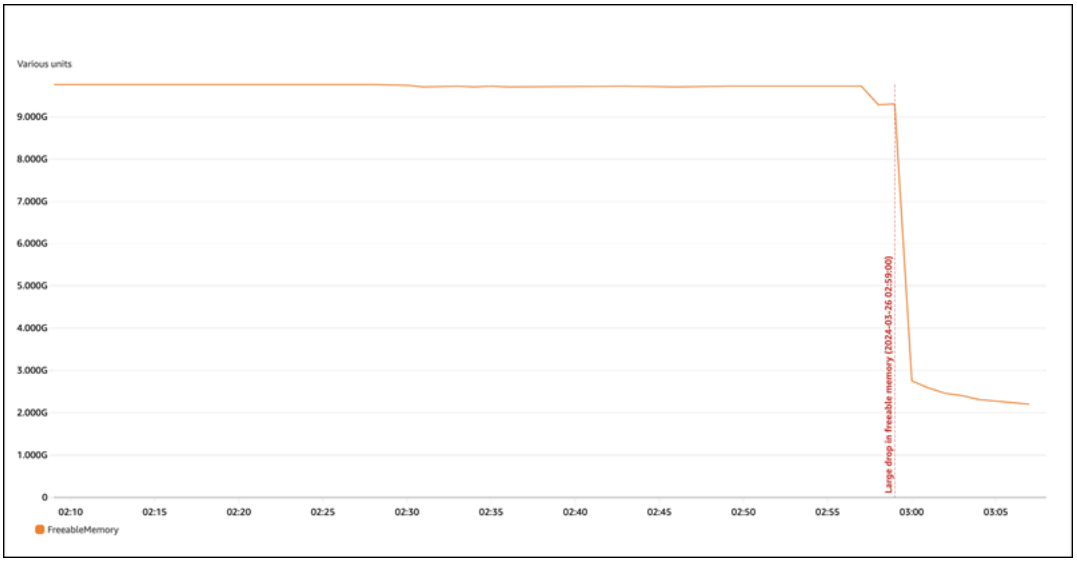
Aurora MySQL バージョン 3.05 以降では、パフォーマンススキーマのステートメント概要テーブルでステートメントダイジェスト別の最大メモリ使用量を追跡することもできます。ステートメント概要テーブルには、正規化されたステートメントダイジェストとその実行に関する集約統計が表示されます。`MAX_TOTAL_MEMORY` 列は、統計の最後のリセット後またはデータベースインスタンスの再起動後の最大メモリ使用量をクエリダイジェスト別に特定するのに利用できます。大量のメモリを消費している可能性がある特定のクエリを特定するのに役立ちます。

> **Note**
>
> パフォーマンススキーマと sys スキーマには、サーバーの現在のメモリ使用量と、接続およびエンジンコンポーネント別のメモリ使用量のハイウォーターマークが表示されます。パフォーマンススキーマはメモリ内に保持されるため、DB インスタンスを再起動すると情報がリセットされます。履歴を長期にわたって保持するには、このデータの取得と保存をパフォーマンススキーマの外部に設定することをお勧めします。

トピック

例 1: 連続的に高いメモリ使用量

CloudWatch の `FreeableMemory` を概観すると、2024-03-26 02:59 UTC にメモリ使用量の大幅増を確認できます。



このグラフでは、詳細がわかりません。どのコンポーネントのメモリ使用量が最も多いかを判断するには、データベースにログインして `sys.memory_global_by_current_bytes` を確認できます。このテーブルには、MySQL が追跡するメモリイベントのリストと、イベント別のメモリ割り当てに関する情報が表示されます。各メモリ追跡イベントは `memory/%` で始まり、その後にイベントに関連するエンジンコンポーネントや機能に関する他の情報が続きます。

例えば、memory/performance_schema/% はパフォーマンススキーマに関連するメモリイベントであり、memory/innodb/% は InnoDB に関連するイベントです。イベントの命名規則の詳細については、MySQL ドキュメントの「Performance Schema instrument naming conventions」を参照してください。

次のクエリから、current_alloc に基づいて有力な原因を確認できますが、memory/performance_schema/% イベントも多数あることがわかります。

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes LIMIT 10;

+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
| event_name                                                                    |
 current_count | current_alloc | current_avg_alloc | high_count | high_alloc |
 high_avg_alloc |
+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
| memory/sql/Prepared_statement::main_mem_root                                  |
 512817 | 4.91 GiB      | 10.04 KiB         |     512823 | 4.91 GiB   | 10.04 KiB
 |
| memory/performance_schema/prepared_statements_instances                       |
   252 | 488.25 MiB    | 1.94 MiB          |        252 | 488.25 MiB | 1.94 MiB       |
| memory/innodb/hash0hash                                                       |
     4 | 79.07 MiB     | 19.77 MiB         |          4 | 79.07 MiB  | 19.77 MiB      |
| memory/performance_schema/events_errors_summary_by_thread_by_error            |
  1028 | 52.27 MiB     | 52.06 KiB         |       1028 | 52.27 MiB  | 52.06 KiB      |
| memory/performance_schema/events_statements_summary_by_thread_by_event_name |
     4 | 47.25 MiB     | 11.81 MiB         |          4 | 47.25 MiB  | 11.81 MiB      |
| memory/performance_schema/events_statements_summary_by_digest                |
     1 | 40.28 MiB     | 40.28 MiB         |          1 | 40.28 MiB  | 40.28 MiB      |
| memory/performance_schema/memory_summary_by_thread_by_event_name              |
     4 | 31.64 MiB     | 7.91 MiB          |          4 | 31.64 MiB  | 7.91 MiB       |
| memory/innodb/memory                                                          |
 15227 | 27.44 MiB     | 1.85 KiB          |      20619 | 33.33 MiB  | 1.66 KiB       |
| memory/sql/String::value                                                      |
 74411 | 21.85 MiB     |  307 bytes        |      76867 | 25.54 MiB  |  348 bytes     |
| memory/sql/TABLE                                                              |
  8381 | 21.03 MiB     | 2.57 KiB          |       8381 | 21.03 MiB  | 2.57 KiB       |
+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
```

```
10 rows in set (0.02 sec)
```

パフォーマンススキーマはメモリに保存されることを以前に説明しました。つまり、パフォーマンススキーマも performance_schema メモリ計測で追跡されます。

> **Note**
>
> パフォーマンススキーマが大量のメモリを使用していることがわかった場合、そのメモリ使用量を制限するには、要件に応じてデータベースパラメータを調整できます。詳細については、MySQL ドキュメントの「The Performance Schema memory-allocation model」を参照してください。

見やすくするために、パフォーマンススキーマイベントを除外して同じクエリを再実行できます。出力は、次のように表示されます。

- メモリ使用量が多いのは memory/sql/Prepared_statement::main_mem_root です。
- current_alloc 列を見ると、MySQL では、このイベントに現在 4.91 GiB が割り当てられていることがわかります。
- high_alloc column によると、4.91 GiB は、統計の最後のリセット後またはサーバーの再起動後からの current_alloc のハイウォーターマークであることがわかります。つまり、memory/sql/Prepared_statement::main_mem_root は最高値になっています。

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes WHERE event_name NOT LIKE
 'memory/performance_schema/%' LIMIT 10;

+-----------------------------------------------+---------------+--------------
+------------------+-----------+-----------+----------------+
| event_name                                    | current_count | current_alloc |
 current_avg_alloc | high_count | high_alloc | high_avg_alloc |
+-----------------------------------------------+---------------+--------------
+------------------+-----------+-----------+----------------+
| memory/sql/Prepared_statement::main_mem_root  |        512817 | 4.91 GiB      | 10.04
 KiB         |     512823 | 4.91 GiB   | 10.04 KiB      |
| memory/innodb/hash0hash                       |             4 | 79.07 MiB     | 19.77
 MiB         |          4 | 79.07 MiB  | 19.77 MiB      |
| memory/innodb/memory                          |         17096 | 31.68 MiB     | 1.90
 KiB          |      22498 | 37.60 MiB  | 1.71 KiB       |
```

```
| memory/sql/String::value                          |        122277 | 27.94 MiB     |  239
 bytes        |     124699 | 29.47 MiB  |  247 bytes     |
| memory/sql/TABLE                                  |          9927 | 24.67 MiB     | 2.55
 KiB          |       9929 | 24.68 MiB  | 2.55 KiB       |
| memory/innodb/lock0lock                           |          8888 | 19.71 MiB     | 2.27
 KiB          |       8888 | 19.71 MiB  | 2.27 KiB       |
| memory/sql/Prepared_statement::infrastructure |        257623 | 16.24 MiB     |   66
 bytes        |     257631 | 16.24 MiB  |   66 bytes     |
| memory/mysys/KEY_CACHE                            |             3 | 16.00 MiB     | 5.33
 MiB          |          3 | 16.00 MiB  | 5.33 MiB       |
| memory/innodb/sync0arr                            |             3 | 7.03 MiB      | 2.34
 MiB          |          3 | 7.03 MiB   | 2.34 MiB       |
| memory/sql/THD::main_mem_root                     |           815 | 6.56 MiB      | 8.24
 KiB          |        849 | 7.19 MiB   | 8.67 KiB       |
+---------------------------------------------------+---------------+---------------
+------------------+-----------+-----------+----------------+
10 rows in set (0.06 sec)
```

イベントの名前から、このメモリはプリペアドステートメントに使用されていることがわかります。このメモリを使用している接続を確認したい場合は、memory_by_thread_by_current_bytes を参照できます。

次の例では、各接続に約 7 MiB が割り当てられ、ハイウォーターマークは約 6.29 MiB (current_max_alloc) です。この例では、プリペアドステートメントで 80 のテーブルと 800 の接続に sysbench を使用しているため、これは当然と言えます。このシナリオでメモリ使用量を減らしたい場合は、アプリケーションによるプリペアドステートメントの使用を最適化してメモリ消費量を削減できます。

```
mysql> SELECT * FROM sys.memory_by_thread_by_current_bytes;

+-----------+-------------------------------------------+--------------------
+------------------+------------------+------------------+-----------------+
| thread_id | user                                      | current_count_used |
 current_allocated | current_avg_alloc | current_max_alloc | total_allocated |
+-----------+-------------------------------------------+--------------------
+------------------+------------------+------------------+-----------------+
|        46 | rdsadmin@localhost                        |                405 | 8.47 MiB
          | 21.42 KiB         | 8.00 MiB          | 155.86 MiB      |
|        61 | reinvent@10.0.4.4                         |               1749 | 6.72 MiB
          | 3.93 KiB          | 6.29 MiB          | 14.24 MiB       |
|       101 | reinvent@10.0.4.4                         |               1845 | 6.71 MiB
          | 3.72 KiB          | 6.29 MiB          | 14.50 MiB       |
```

```
|        55 | reinvent@10.0.4.4                        |              1674 | 6.68 MiB
          | 4.09 KiB          | 6.29 MiB          | 14.13 MiB       |
|        57 | reinvent@10.0.4.4                        |              1416 | 6.66 MiB
          | 4.82 KiB          | 6.29 MiB          | 13.52 MiB       |
|       112 | reinvent@10.0.4.4                        |              1759 | 6.66 MiB
          | 3.88 KiB          | 6.29 MiB          | 14.17 MiB       |
|        66 | reinvent@10.0.4.4                        |              1428 | 6.64 MiB
          | 4.76 KiB          | 6.29 MiB          | 13.47 MiB       |
|        75 | reinvent@10.0.4.4                        |              1389 | 6.62 MiB
          | 4.88 KiB          | 6.29 MiB          | 13.40 MiB       |
|       116 | reinvent@10.0.4.4                        |              1333 | 6.61 MiB
          | 5.08 KiB          | 6.29 MiB          | 13.21 MiB       |
|        90 | reinvent@10.0.4.4                        |              1448 | 6.59 MiB
          | 4.66 KiB          | 6.29 MiB          | 13.58 MiB       |
|        98 | reinvent@10.0.4.4                        |              1440 | 6.57 MiB
          | 4.67 KiB          | 6.29 MiB          | 13.52 MiB       |
|        94 | reinvent@10.0.4.4                        |              1433 | 6.57 MiB
          | 4.69 KiB          | 6.29 MiB          | 13.49 MiB       |
|        62 | reinvent@10.0.4.4                        |              1323 | 6.55 MiB
          | 5.07 KiB          | 6.29 MiB          | 13.48 MiB       |
|        87 | reinvent@10.0.4.4                        |              1323 | 6.55 MiB
          | 5.07 KiB          | 6.29 MiB          | 13.25 MiB       |
|        99 | reinvent@10.0.4.4                        |              1346 | 6.54 MiB
          | 4.98 KiB          | 6.29 MiB          | 13.24 MiB       |
|       105 | reinvent@10.0.4.4                        |              1347 | 6.54 MiB
          | 4.97 KiB          | 6.29 MiB          | 13.34 MiB       |
|        73 | reinvent@10.0.4.4                        |              1335 | 6.54 MiB
          | 5.02 KiB          | 6.29 MiB          | 13.23 MiB       |
|        54 | reinvent@10.0.4.4                        |              1510 | 6.53 MiB
          | 4.43 KiB          | 6.29 MiB          | 13.49 MiB       |
.                                                                                .
.                                                                                .
.                                                                                .
|       812 | reinvent@10.0.4.4                        |              1259 | 6.38 MiB
          | 5.19 KiB          | 6.29 MiB          | 13.05 MiB       |
|       214 | reinvent@10.0.4.4                        |              1279 | 6.38 MiB
          | 5.10 KiB          | 6.29 MiB          | 12.90 MiB       |
|       325 | reinvent@10.0.4.4                        |              1254 | 6.38 MiB
          | 5.21 KiB          | 6.29 MiB          | 12.99 MiB       |
|       705 | reinvent@10.0.4.4                        |              1273 | 6.37 MiB
          | 5.13 KiB          | 6.29 MiB          | 13.03 MiB       |
```

```
|       530 | reinvent@10.0.4.4                          |               1268 | 6.37 MiB
          | 5.15 KiB          | 6.29 MiB          | 12.92 MiB       |
|       307 | reinvent@10.0.4.4                          |               1263 | 6.37 MiB
          | 5.17 KiB          | 6.29 MiB          | 12.87 MiB       |
|       738 | reinvent@10.0.4.4                          |               1260 | 6.37 MiB
          | 5.18 KiB          | 6.29 MiB          | 13.00 MiB       |
|       819 | reinvent@10.0.4.4                          |               1252 | 6.37 MiB
          | 5.21 KiB          | 6.29 MiB          | 13.01 MiB       |
|        31 | innodb/srv_purge_thread                    |              17810 | 3.14 MiB
          |  184 bytes        | 2.40 MiB          | 205.69 MiB      |
|        38 | rdsadmin@localhost                         |                599 | 1.76 MiB
          | 3.01 KiB          | 1.00 MiB          | 25.58 MiB       |
|         1 | sql/main                                   |               3756 | 1.32 MiB
          |  367 bytes        | 355.78 KiB        | 6.19 MiB        |
|       854 | rdsadmin@localhost                         |                 46 | 1.08 MiB
          | 23.98 KiB         | 1.00 MiB          | 5.10 MiB        |
|        30 | innodb/clone_gtid_thread                   |               1596 | 573.14
 KiB        |  367 bytes        | 254.91 KiB        | 970.69 KiB      |
|        40 | rdsadmin@localhost                         |                235 | 245.19
 KiB        | 1.04 KiB          | 128.88 KiB        | 808.64 KiB      |
|       853 | rdsadmin@localhost                         |                 96 | 94.63
 KiB         | 1009 bytes        | 29.73 KiB         | 422.45 KiB      |
|        36 | rdsadmin@localhost                         |                 33 | 36.29
 KiB         | 1.10 KiB          | 16.08 KiB         | 74.15 MiB       |
|        33 | sql/event_scheduler                        |                  3 | 16.27
 KiB         | 5.42 KiB          | 16.04 KiB         | 16.27 KiB       |
|        35 | sql/compress_gtid_table                    |                  8 | 14.20
 KiB         | 1.77 KiB          | 8.05 KiB          | 18.62 KiB       |
|        25 | innodb/fts_optimize_thread                 |                 12 | 1.86 KiB
          |  158 bytes        |  648 bytes        | 1.98 KiB        |
|        23 | innodb/srv_master_thread                   |                 11 | 1.23 KiB
          |  114 bytes        |  361 bytes        | 24.40 KiB       |
|        24 | innodb/dict_stats_thread                   |                 11 | 1.23 KiB
          |  114 bytes        |  361 bytes        | 1.35 KiB        |
|         5 | innodb/io_read_thread                      |                  1 |  144
 bytes        |  144 bytes        |  144 bytes        |  144 bytes      |
|         6 | innodb/io_read_thread                      |                  1 |  144
 bytes        |  144 bytes        |  144 bytes        |  144 bytes      |
|         2 | sql/aws_oscar_log_level_monitor            |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|         4 | innodb/io_ibuf_thread                      |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|         7 | innodb/io_write_thread                     |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
```

```
|         8 | innodb/io_write_thread                     |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|         9 | innodb/io_write_thread                     |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        10 | innodb/io_write_thread                     |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        11 | innodb/srv_lra_thread                      |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        12 | innodb/srv_akp_thread                      |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        18 | innodb/srv_lock_timeout_thread             |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |  248 bytes      |
|        19 | innodb/srv_error_monitor_thread            |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        20 | innodb/srv_monitor_thread                  |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        21 | innodb/buf_resize_thread                   |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        22 | innodb/btr_search_sys_toggle_thread        |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        32 | innodb/dict_persist_metadata_table_thread  |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
|        34 | sql/signal_handler                         |                  0 |    0
 bytes        |    0 bytes        |    0 bytes        |    0 bytes      |
+-----------+--------------------------------------------+-------------------
+-------------------+------------------+------------------+-----------------+
831 rows in set (2.48 sec)
```

前に述べたように、ここでスレッド ID (thd_id) 値は、サーバーのバックグラウンドスレッドまたはデータベース接続を参照できます。スレッド ID 値をデータベース接続 ID にマッピングする場合は、performance_schema.threads テーブルまたは sys.processlist ビューを使用できます。conn_id は接続 ID です。

```
mysql> SELECT thd_id,conn_id,user,db,command,state,time,last_wait FROM sys.processlist
 WHERE user='reinvent@10.0.4.4';

+--------+---------+------------------+----------+---------+----------------+------
+-------------------------------------------------+
| thd_id | conn_id | user             | db       | command | state          | time |
 last_wait                                        |
+--------+---------+------------------+----------+---------+----------------+------
+-------------------------------------------------+
```

```
|    590 |     562 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                          |
|    578 |     550 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL           |    0 |
 idle                                            |
|    579 |     551 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                          |
|    580 |     552 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                       |
|    581 |     553 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                       |
|    582 |     554 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL           |    0 |
 idle                                            |
|    583 |     555 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL           |    0 |
 idle                                            |
|    584 |     556 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                       |
|    585 |     557 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                          |
|    586 |     558 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                       |
|    587 |     559 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                          |
.
                                                 .
.
                                                 .
.
                                                 .
|    323 |     295 | reinvent@10.0.4.4 | sysbench | Sleep   | NULL           |    0 |
 idle                                            |
|    324 |     296 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                       |
|    325 |     297 | reinvent@10.0.4.4 | sysbench | Execute | closing tables |    0 |
 wait/io/redo_log_flush                          |
|    326 |     298 | reinvent@10.0.4.4 | sysbench | Execute | updating       |    0 |
 wait/io/table/sql/handler                       |
|    438 |     410 | reinvent@10.0.4.4 | sysbench | Execute | System lock    |    0 |
 wait/lock/table/sql/handler                     |
|    280 |     252 | reinvent@10.0.4.4 | sysbench | Sleep   | starting       |    0 |
 wait/io/socket/sql/client_connection            |
|     98 |      70 | reinvent@10.0.4.4 | sysbench | Query   | freeing items  |    0 |
 NULL                                            |
+--------+---------+-------------------+----------+---------+----------------+------
+-------------------------------------------------+
```

```
804 rows in set (5.51 sec)
```

次に sysbench ワークロードを停止し、接続を終了してメモリを解放します。イベントをもう一度確認すると、メモリが解放されたことを確認できますが、high_alloc には以前としてハイウォーターマークが表示されています。high_alloc 列は、メモリ使用量の急増を特定するのに非常に役立ちます。current_alloc は、現在割り当てられているメモリのみを示すため、使用量の急増をすぐに特定できない場合があります。

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes WHERE event_name='memory/sql/
Prepared_statement::main_mem_root' LIMIT 10;

+-----------------------------------------------+---------------+---------------
+------------------+-----------+-----------+----------------+
| event_name                                    | current_count | current_alloc |
 current_avg_alloc | high_count | high_alloc | high_avg_alloc |
+-----------------------------------------------+---------------+---------------
+------------------+-----------+-----------+----------------+
| memory/sql/Prepared_statement::main_mem_root |            17 | 253.80 KiB    | 14.93
 KiB          |     512823 | 4.91 GiB   | 10.04 KiB      |
+-----------------------------------------------+---------------+---------------
+------------------+-----------+-----------+----------------+
1 row in set (0.00 sec)
```

performance_schema をリセットする場合は、high_alloc のメモリ概要テーブルを切り捨てることができますが、これに伴ってすべてのメモリ計測がリセットされます。詳細については、MySQL ドキュメントの「Performance Schema general table characteristics」を参照してください。

次の例では、切り捨て後に high_alloc がリセットされていることがわかります。

```
mysql> TRUNCATE `performance_schema`.`memory_summary_global_by_event_name`;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM sys.memory_global_by_current_bytes WHERE event_name='memory/sql/
Prepared_statement::main_mem_root' LIMIT 10;

+-----------------------------------------------+---------------+---------------
+------------------+-----------+-----------+----------------+
| event_name                                    | current_count | current_alloc |
 current_avg_alloc | high_count | high_alloc | high_avg_alloc |
+-----------------------------------------------+---------------+---------------
+------------------+-----------+-----------+----------------+
```

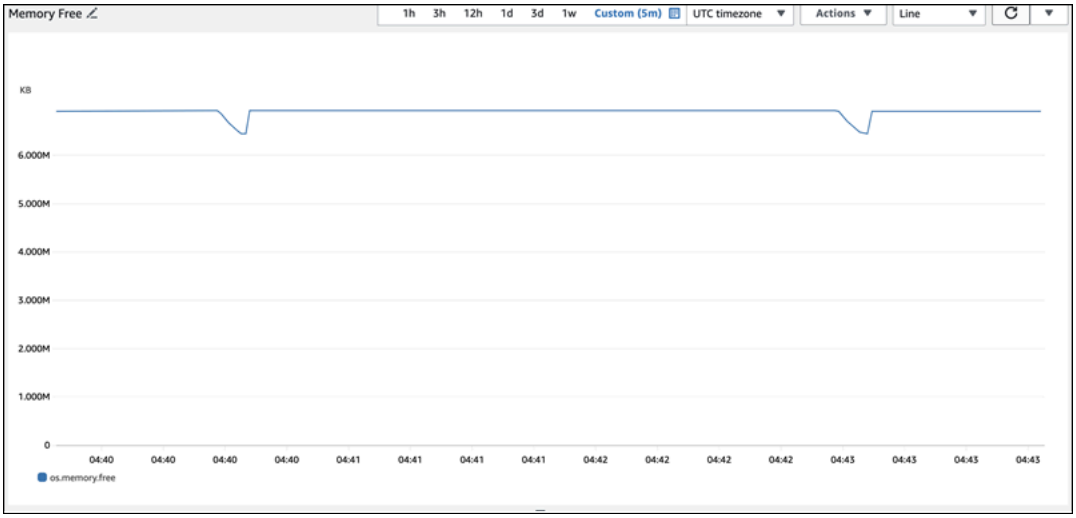
```
| memory/sql/Prepared_statement::main_mem_root |            17 | 253.80 KiB    | 14.93
 KiB         |         17 | 253.80 KiB | 14.93 KiB      |
+----------------------------------------------+---------------+---------------
+------------------+-----------+-----------+---------------+
1 row in set (0.00 sec)
```

例 2: 一時的なメモリスパイク

もう 1 つのよくある現象は、データベースサーバーでメモリ使用量が短時間に急増することです。これは空きメモリの定期的な低下を示している場合があります。メモリは解放済みであるため、sys.memory_global_by_current_bytes の current_alloc を使用したトラブルシューティングは困難です。

> **Note**
>
> パフォーマンススキーマの統計をリセットしたり、データベースインスタンスを再起動したりすると、この情報は sys または performance_schema で使用できなくなります。この情報を保持するには、外部メトリクス収集を設定することをお勧めします。

次のグラフに示す拡張モニタリングの os.memory.free メトリクスでは、メモリ使用量が 7 秒間だけ急増したことがわかります。拡張モニタリングでは、1 秒という短い間隔で監視できるため、このような一時的なスパイクを検出するのに最適です。



ここでメモリ消費の原因を診断しやすくするために、sys メモリの概要ビューの high_alloc と、パフォーマンススキーマのステートメント概要テーブルを組み合わせて使用し、問題のあるセッションや接続を特定できます。

予想どおり、現在のメモリ使用量は多くないため、current_alloc の sys スキーマビューでは大きな問題を確認できません。

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes LIMIT 10;

+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
| event_name                                                                    |
 current_count | current_alloc | current_avg_alloc | high_count | high_alloc |
 high_avg_alloc |
+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
| memory/innodb/hash0hash                                                       |
     4 | 79.07 MiB     | 19.77 MiB         |          4 | 79.07 MiB  | 19.77 MiB      |
| memory/innodb/os0event                                                        |
 439372 | 60.34 MiB     |  144 bytes        |     439372 | 60.34 MiB  |  144 bytes
 |
| memory/performance_schema/events_statements_summary_by_digest                |
     1 | 40.28 MiB     | 40.28 MiB         |          1 | 40.28 MiB  | 40.28 MiB      |
| memory/mysys/KEY_CACHE                                                        |
     3 | 16.00 MiB     | 5.33 MiB          |          3 | 16.00 MiB  | 5.33 MiB       |
| memory/performance_schema/events_statements_history_long                      |
     1 | 14.34 MiB     | 14.34 MiB         |          1 | 14.34 MiB  | 14.34 MiB      |
| memory/performance_schema/events_errors_summary_by_thread_by_error            |
   257 | 13.07 MiB     | 52.06 KiB         |        257 | 13.07 MiB  | 52.06 KiB      |
| memory/performance_schema/events_statements_summary_by_thread_by_event_name |
     1 | 11.81 MiB     | 11.81 MiB         |          1 | 11.81 MiB  | 11.81 MiB      |
| memory/performance_schema/events_statements_summary_by_digest.digest_text    |
     1 | 9.77 MiB      | 9.77 MiB          |          1 | 9.77 MiB   | 9.77 MiB       |
| memory/performance_schema/events_statements_history_long.digest_text         |
     1 | 9.77 MiB      | 9.77 MiB          |          1 | 9.77 MiB   | 9.77 MiB       |
| memory/performance_schema/events_statements_history_long.sql_text            |
     1 | 9.77 MiB      | 9.77 MiB          |          1 | 9.77 MiB   | 9.77 MiB       |
+-------------------------------------------------------------------------------
+---------------+---------------+-------------------+------------+------------
+----------------+
10 rows in set (0.01 sec)
```

ビューを high_alloc の順に並べ替えると、memory/temptable/physical_ram コンポーネントにまさに問題があることがわかります。使用量の最高値が 515.00 MiB になっています。

その名前が示すように、memory/temptable/physical_ram は、MySQL 8.0 で MySQL に導入された TEMP ストレージエンジンのメモリ使用量を計測します。MySQL で一時テーブルを使用する方法の詳細については、MySQL ドキュメントの「Internal temporary table use in MySQL」を参照してください。

> **Note**
>
> この例では、sys.x$memory_global_by_current_bytes ビューを使用しています。

```
mysql> SELECT event_name, format_bytes(current_alloc) AS "currently allocated",
 sys.format_bytes(high_alloc) AS "high-water mark"
FROM sys.x$memory_global_by_current_bytes ORDER BY high_alloc DESC LIMIT 10;

+-------------------------------------------------------------------------------
+--------------------+-----------------+
| event_name                                                                    |
 currently allocated | high-water mark |
+-------------------------------------------------------------------------------
+--------------------+-----------------+
| memory/temptable/physical_ram                                                 | 4.00
 MiB             | 515.00 MiB      |
| memory/innodb/hash0hash                                                       | 79.07
 MiB            | 79.07 MiB       |
| memory/innodb/os0event                                                        | 63.95
 MiB            | 63.95 MiB       |
| memory/performance_schema/events_statements_summary_by_digest                | 40.28
 MiB            | 40.28 MiB       |
| memory/mysys/KEY_CACHE                                                        | 16.00
 MiB            | 16.00 MiB       |
| memory/performance_schema/events_statements_history_long                      | 14.34
 MiB            | 14.34 MiB       |
| memory/performance_schema/events_errors_summary_by_thread_by_error           | 13.07
 MiB            | 13.07 MiB       |
| memory/performance_schema/events_statements_summary_by_thread_by_event_name | 11.81
 MiB            | 11.81 MiB       |
| memory/performance_schema/events_statements_summary_by_digest.digest_text    | 9.77
 MiB             | 9.77 MiB        |
| memory/performance_schema/events_statements_history_long.sql_text            | 9.77
 MiB             | 9.77 MiB        |
+-------------------------------------------------------------------------------
+--------------------+-----------------+
```

```
10 rows in set (0.00 sec)
```

例 1: 連続的に高いメモリ使用量 では、各接続の現在のメモリ使用量を調べて、どの接続が問題のメモリを使用しているかを判断しました。この例では、メモリは解放済みであるため、現在の接続のメモリ使用量を調べても役に立ちません。

より深く掘り下げて問題の原因であるステートメント、ユーザー、ホストを見つけるには、パフォーマンススキーマを使用します。パフォーマンススキーマには、イベント名、ステートメントダイジェスト、ホスト、スレッド、ユーザーなど、さまざまな次元で分類された複数のステートメント概要テーブルがあります。各ビューでは、特定のステートメントの実行場所や実行内容をより深く掘り下げることができます。このセクションでは、主に MAX_TOTAL_MEMORY について説明していますが、「パフォーマンススキーマのステートメント概要テーブル」ですべての例に関する詳細を参照できます。

```
mysql> SHOW TABLES IN performance_schema LIKE 'events_statements_summary_%';

+------------------------------------------------------------+
| Tables_in_performance_schema (events_statements_summary_%) |
+------------------------------------------------------------+
| events_statements_summary_by_account_by_event_name         |
| events_statements_summary_by_digest                        |
| events_statements_summary_by_host_by_event_name            |
| events_statements_summary_by_program                       |
| events_statements_summary_by_thread_by_event_name          |
| events_statements_summary_by_user_by_event_name            |
| events_statements_summary_global_by_event_name             |
+------------------------------------------------------------+
7 rows in set (0.00 sec)
```

まず、events_statements_summary_by_digest を調べて MAX_TOTAL_MEMORY を確認します。

これにより、以下がわかります。

- メモリ使用量が多い筆頭は、ダイジェスト 20676ce4a690592ff05debcffcbc26faeb76f22005e7628364d7a498769d0c4a を使用するクエリのようです。MAX_TOTAL_MEMORY は 537450710 で、sys.x$memory_global_by_current_bytes の memory/temptable/physical_ram イベントで確認したハイウォーターマークと一致しています。

- これまでに 4 回 (COUNT_STAR) 実行されており、最初は 2024-03-26 04:08:34.943256、最後は 2024-03-26 04:43:06.998310 です。

```
mysql> SELECT SCHEMA_NAME,DIGEST,COUNT_STAR,MAX_TOTAL_MEMORY,FIRST_SEEN,LAST_SEEN
FROM performance_schema.events_statements_summary_by_digest ORDER BY MAX_TOTAL_MEMORY
 DESC LIMIT 5;

+-------------+----------------------------------------------------------------
+------------+-----------------+---------------------------
+----------------------------+
| SCHEMA_NAME | DIGEST                                                          |
 COUNT_STAR | MAX_TOTAL_MEMORY | FIRST_SEEN                 | LAST_SEEN
  |
+-------------+----------------------------------------------------------------
+------------+-----------------+---------------------------
+----------------------------+
| sysbench    | 20676ce4a690592ff05debcffcbc26faeb76f22005e7628364d7a498769d0c4a |
     4 |        537450710 | 2024-03-26 04:08:34.943256 | 2024-03-26 04:43:06.998310 |
| NULL        | f158282ea0313fefd0a4778f6e9b92fc7d1e839af59ebd8c5eea35e12732c45d |
     4 |          3636413 | 2024-03-26 04:29:32.712348 | 2024-03-26 04:36:26.269329 |
| NULL        | 0046bc5f642c586b8a9afd6ce1ab70612dc5b1fd2408fa8677f370c1b0ca3213 |
     2 |          3459965 | 2024-03-26 04:31:37.674008 | 2024-03-26 04:32:09.410718 |
| NULL        | 8924f01bba3c55324701716c7b50071a60b9ceaf17108c71fd064c20c4ab14db |
     1 |          3290981 | 2024-03-26 04:31:49.751506 | 2024-03-26 04:31:49.751506 |
| NULL        | 90142bbcb50a744fcec03a1aa336b2169761597ea06d85c7f6ab03b5a4e1d841 |
     1 |          3131729 | 2024-03-26 04:15:09.719557 | 2024-03-26 04:15:09.719557 |
+-------------+----------------------------------------------------------------
+------------+-----------------+---------------------------
+----------------------------+
5 rows in set (0.00 sec)
```

問題があるダイジェストがわかったので、クエリテキスト、実行したユーザー、実行場所などの詳細を取得できます。返されたダイジェストテキストから、これは 4 つの一時テーブルを作成して 4 つのテーブルスキャンを実行する一般的なテーブル式 (CTE) であり、非常に効率の悪いものであることがわかります。

```
mysql> SELECT
 SCHEMA_NAME,DIGEST_TEXT,QUERY_SAMPLE_TEXT,MAX_TOTAL_MEMORY,SUM_ROWS_SENT,SUM_ROWS_EXAMINED,SUM
FROM performance_schema.events_statements_summary_by_digest
WHERE DIGEST='20676ce4a690592ff05debcffcbc26faeb76f22005e7628364d7a498769d0c4a'\G;
```

```
*************************** 1. row ***************************
           SCHEMA_NAME: sysbench
           DIGEST_TEXT: WITH RECURSIVE `cte` ( `n` ) AS ( SELECT ? FROM `sbtest1` UNION
 ALL SELECT `id` + ? FROM `sbtest1` ) SELECT * FROM `cte`
     QUERY_SAMPLE_TEXT: WITH RECURSIVE cte (n) AS (   SELECT 1  from sbtest1 UNION ALL
  SELECT id + 1 FROM sbtest1) SELECT * FROM cte
      MAX_TOTAL_MEMORY: 537450710
         SUM_ROWS_SENT: 80000000
     SUM_ROWS_EXAMINED: 80000000
SUM_CREATED_TMP_TABLES: 4
     SUM_NO_INDEX_USED: 4
1 row in set (0.01 sec)
```

events_statements_summary_by_digest テーブルとその他のパフォーマンススキーマのステートメント概要テーブルの詳細については、MySQL ドキュメントの「Statement summary tables」を参照してください。

EXPLAIN または EXPLAIN ANALYZE ステートメントを実行して詳細を確認することもできます。

> **Note**
>
> EXPLAIN ANALYZE は、EXPLAIN よりも多くの情報を提供できますが、クエリも実行するので注意する必要があります。

```
-- EXPLAIN
mysql> EXPLAIN WITH RECURSIVE cte (n) AS (SELECT 1  FROM sbtest1 UNION ALL SELECT id +
 1 FROM sbtest1) SELECT * FROM cte;

+----+-------------+------------+------------+-------+---------------+------+---------
+------+----------+----------+-------------+
| id | select_type | table      | partitions | type  | possible_keys | key  | key_len |
 ref  | rows     | filtered | Extra       |
+----+-------------+------------+------------+-------+---------------+------+---------
+------+----------+----------+-------------+
|  1 | PRIMARY     | <derived2> | NULL       | ALL   | NULL          | NULL | NULL    |
 NULL | 19221520 |   100.00 | NULL        |
|  2 | DERIVED     | sbtest1    | NULL       | index | NULL          | k_1  | 4       |
 NULL |  9610760 |   100.00 | Using index |
|  3 | UNION       | sbtest1    | NULL       | index | NULL          | k_1  | 4       |
 NULL |  9610760 |   100.00 | Using index |
```

```
+----+-------------+-----------+-----------+-------+--------------+------+---------
+------+----------+----------+-------------+
3 rows in set, 1 warning (0.00 sec)

-- EXPLAIN format=tree
mysql> EXPLAIN format=tree WITH RECURSIVE cte (n) AS (SELECT 1 FROM sbtest1 UNION ALL
 SELECT id + 1 FROM sbtest1) SELECT * FROM cte\G;

*************************** 1. row ***************************
EXPLAIN: -> Table scan on cte  (cost=4.11e+6..4.35e+6 rows=19.2e+6)
    -> Materialize union CTE cte  (cost=4.11e+6..4.11e+6 rows=19.2e+6)
        -> Index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
        -> Index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
1 row in set (0.00 sec)

-- EXPLAIN ANALYZE
mysql> EXPLAIN ANALYZE WITH RECURSIVE cte (n) AS (SELECT 1 from sbtest1 UNION ALL
 SELECT id + 1 FROM sbtest1) SELECT * FROM cte\G;

*************************** 1. row ***************************
EXPLAIN: -> Table scan on cte  (cost=4.11e+6..4.35e+6 rows=19.2e+6) (actual
 time=6666..9201 rows=20e+6 loops=1)
    -> Materialize union CTE cte  (cost=4.11e+6..4.11e+6 rows=19.2e+6) (actual
 time=6666..6666 rows=20e+6 loops=1)
        -> Covering index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
 (actual time=0.0365..2006 rows=10e+6 loops=1)
        -> Covering index scan on sbtest1 using k_1  (cost=1.09e+6 rows=9.61e+6)
 (actual time=0.0311..2494 rows=10e+6 loops=1)
1 row in set (10.53 sec)
```

ここで、クエリを実行したユーザーがわかるでしょうか。パフォーマンススキーマを見ると、destructive_operator ユーザーの MAX_TOTAL_MEMORY が 537450710 になっており、前の結果と一致しています。

> **Note**
>
> パフォーマンススキーマはメモリに保存されるため、監査の唯一のソースとして依存すべきではありません。実行したステートメントやユーザーの履歴を保持する必要がある場合は、[監査ログ記録] を有効にすることをお勧めします。メモリ使用量に関する情報も保持する必要がある場合は、これらの値をエクスポートして保存するようにモニタリングを設定することをお勧めします。

```
mysql> SELECT USER,EVENT_NAME,COUNT_STAR,MAX_TOTAL_MEMORY FROM
 performance_schema.events_statements_summary_by_user_by_event_name
ORDER BY MAX_CONTROLLED_MEMORY DESC LIMIT 5;

+----------------------+---------------------------+------------+------------------+
| USER                 | EVENT_NAME                | COUNT_STAR | MAX_TOTAL_MEMORY |
+----------------------+---------------------------+------------+------------------+
| destructive_operator | statement/sql/select      |          4 |        537450710 |
| rdsadmin             | statement/sql/select      |       4172 |          3290981 |
| rdsadmin             | statement/sql/show_tables |          2 |          3615821 |
| rdsadmin             | statement/sql/show_fields |          2 |          3459965 |
| rdsadmin             | statement/sql/show_status |         75 |          1914976 |
+----------------------+---------------------------+------------+------------------+
5 rows in set (0.00 sec)

mysql> SELECT HOST,EVENT_NAME,COUNT_STAR,MAX_TOTAL_MEMORY FROM
 performance_schema.events_statements_summary_by_host_by_event_name
WHERE HOST != 'localhost' AND COUNT_STAR>0 ORDER BY MAX_CONTROLLED_MEMORY DESC LIMIT 5;

+------------+----------------------+------------+------------------+
| HOST       | EVENT_NAME           | COUNT_STAR | MAX_TOTAL_MEMORY |
+------------+----------------------+------------+------------------+
| 10.0.8.231 | statement/sql/select |          4 |        537450710 |
+------------+----------------------+------------+------------------+
1 row in set (0.00 sec)
```

Aurora MySQL データベースのメモリ不足の問題のトラブルシューティング

Aurora MySQL aurora_oom_response インスタンスレベルパラメータは、DB インスタンスによって、システムメモリをモニタリングしてさまざまなステートメントおよび接続で消費されるメモリを推測できるようにします。システムでメモリ不足が発生した場合、メモリを解放するためのアクションリストを実行できます。これは、メモリ不足 (OOM) を原因とするデータベースの再起動を避ける目的で実行されます。このインスタンスレベルのパラメータでは、メモリが少ない場合に DB インスタンスが実行すべきアクションを、カンマ区切りの文字列で指定できます。この aurora_oom_response パラメータは、Aurora MySQL バージョン 2 および 3 でサポートされています。

aurora_oom_response パラメータには、以下の値とそれらの組み合わせを使用できます。空の文字列はアクションが実行されないことを意味し、実質的にこの機能はオフになります。そのため、データベースは OOM の再起動が発生しやすくなります。

- `decline` – DB インスタンスのメモリが少なくなった場合、新しいクエリを拒否します。
- `kill_connect` – 大量のメモリを消費しているデータベース接続を閉じ、現在のトランザクションとデータ定義言語 (DDL) ステートメントを終了します。この応答は、Aurora MySQL バージョン 2 ではサポートされていません。

  詳細については、MySQL ドキュメントの「KILL ステートメント」を参照してください。
- `kill_query` – インスタンスのメモリが低しきい値を超えるまで、メモリ使用量の高い順にクエリを終了します。DDL ステートメントは終了されません。

  詳細については、MySQL ドキュメントの「KILL ステートメント」を参照してください。
- `print` – 大量のメモリを使用するクエリのみを出力します。
- `tune` - 内部テーブルキャッシュを調整して、メモリをシステムに戻します。Aurora MySQL は、メモリが少ない状態では `table_open_cache` や `table_definition_cache` などのキャッシュに使用されるメモリを低減します。最終的に、Aurora MySQL は、システムのメモリ不足がなくなると、メモリ使用量を通常に戻します。

  詳細については、MySQL ドキュメントの「table_open_cache」と「table_definition_cache」を参照してください。
- `tune_buffer_pool` – バッファプールのサイズを小さくしてメモリを解放し、データベースサーバーが接続を処理できるようにします。この応答は、Aurora MySQL バージョン 3.06 以降でサポートされています。

  `tune_buffer_pool` を `kill_query` または `aurora_oom_response` パラメータ値の `kill_connect` とペアにする必要があります。そうしない場合、パラメータ値に `tune_buffer_pool` を含めても、バッファプールのサイズ変更は行われません。

3.06 以前のバージョンの Aurora MySQL の場合は、メモリが 4 GiB 以下の DB インスタンスクラスでメモリプレッシャーがかかっているときのデフォルトアクションに `print`、`tune`、`decline`、`kill_query` があります。4 GiB を超えるメモリがある DB インスタンスクラスでは、このパラメータ値はデフォルトで空 (無効) になっています。

Aurora MySQL バージョン 3.06 以降では、メモリが 4 GiB 以下の DB インスタンスクラスの場合、Aurora MySQL は最もメモリ消費量の多い接続 (`kill_connect`) も閉じます。4 GiB を超えるメモリがある DB インスタンスクラスでは、このパラメータ値はデフォルトで `print` になっています。

メモリ不足の問題が頻繁に発生する場合は、performance_schema が有効になっていればメモリのサマリーテーブルを使用してメモリ使用量をモニタリングできます。

Aurora MySQL データベースのログ記録

Aurora MySQL ログは、データベースのアクティビティとエラーに関する重要な情報を提供します。これらのログを有効にすることで、問題の特定とトラブルシューティング、データベースパフォーマンスの把握、データベースアクティビティの監査が可能になります。データベースのパフォーマンスと可用性を最適化するために、すべての Aurora MySQL DB インスタンスでこれらのログを有効にすることをお勧めします。次のタイプのログを有効にできます。各ログには、データベース処理への影響を明らかにするのに役立つ特定の情報が含まれています。

- エラー - Aurora MySQL ではスタートアップ時、シャットダウン時、およびエラー検出時にのみ、エラーログへの書き込みが行われます。DB インスタンスでは、新しいエントリがエラーログに書き込まれないまま、数時間または数日が経過することがあります。最近のエントリがない場合、それは、サーバーにログエントリになり得るエラーが発生しなかったためです。エラーロギングはデフォルトで有効になっています。詳細については、「Aurora MySQL エラーログ」を参照してください。
- 一般 — 一般ログには、データベースエンジンによって実行されたすべての SQL ステートメントを含む、データベースアクティビティに関する詳細情報が含まれます。一般的なロギングの有効化とロギングパラメータの設定の詳細については、「Aurora MySQL のスロークエリと一般ログ」、および MySQL ドキュメントの「一般クエリログ」を参照してください。

  > **Note**
  >
  > 一般ログは非常に大きくなり、ストレージを消費する可能性があります。詳細については、「Aurora MySQL のログのローテーションと保持」を参照してください。

- スロークエリ — スロークエリログは、実行に long_query_time 秒を超える時間がかかり、検証に min_examined_row_limit 行以上を要した SQL ステートメントで構成されています。スロークエリログを使用すると、実行に時間がかかるため最適化の対象となるクエリを見つけることができます。

  long_query_time のデフォルト値は 10 (秒) です。高い値から始めて最も遅いクエリを特定し、その後微調整することをおすすめします。

  log_slow_admin_statements や log_queries_not_using_indexes などの関連パラメータを使用することもできます。rows_examined と rows_returned を比較しま

す。rows_examined が rows_returned よりはるかに大きい場合は、それらのクエリがブロックする可能性があります。

Aurora MySQL バージョン 3 では、log_slow_extra を有効にしてより詳細な情報を取得することができます。詳細については、MySQL ドキュメントの「スロークエリログの内容」を参照してください。クエリ実行をインタラクティブにデバッグするようにセッションレベルで long_query_time を変更することもできます。これは log_slow_extra がグローバルに有効になっている場合に特に便利です。

スロークエリロギングの有効化とロギングパラメータの設定の詳細については、「Aurora MySQL のスロークエリと一般ログ」、および MySQL ドキュメントの「スロークエリログ」を参照してください。

- 監査 — 監査ログはデータベースのアクティビティをモニタリングして記録します。Aurora MySQL の監査ログは、高度な監査と呼ばれます。高度な監査を有効にするには、特定の DB クラスターパラメータを設定します。詳細については、「Amazon Aurora MySQL DB クラスターでのアドバンストな監査の使用」を参照してください。
- バイナリ — バイナリログ (binlog) には、テーブル作成操作やテーブルデータへの変更など、データベースの変更を説明するイベントが含まれます。また、行ベースのロギングが使用されていない限り、変更を加えた可能性のあるステートメント (行と一致しない DELETE など) のイベントも含まれます。バイナリログには、各ステートメントがデータの更新にかかった時間に関する情報も含まれています。

  バイナリロギングを有効にしてサーバーを実行させると、パフォーマンスがわずかに低下します。ただし、一般に、レプリケーションや復元操作を設定できるバイナリログの利点は、このわずかなパフォーマンスの低下よりも上回ります。

  > **Note**
  >
  > Aurora MySQL では、復元操作にバイナリロギングは必要ありません。

  バイナリロギングの有効化とバイナリログ形式の設定の詳細については、「Aurora MySQL バイナリログの設定」、MySQL ドキュメントの「バイナリログ」を参照してください。

エラーログ、一般ログ、スローログ、クエリログ、監査ログを Amazon CloudWatch Logs にパブリッシュできます。詳細については、「Amazon CloudWatch Logs へのデータベースログの発行」を参照してください。

スローログ、一般ログ、バイナリログファイルを要約するもう 1 つの便利なツールは pt-query-digest です。

Aurora MySQL データベースのクエリパフォーマンスのトラブルシューティング

MySQL は、クエリプランの評価方法に影響するシステム変数、切り替え可能な最適化、オプティマイザとインデックスのヒント、オプティマイザのコストモデルを通じてクエリオプティマイザの制御を提供します。これらのデータポイントは、さまざまな MySQL 環境を比較するときだけでなく、以前のクエリ実行プランを現在の実行プランと比較したり、任意の時点の MySQL クエリの全体的な実行状況を把握したりするのに役立ちます。

クエリのパフォーマンスは、実行プラン、テーブルスキーマとサイズ、統計、リソース、インデックス、パラメータ設定など、さまざまな要因に左右されます。クエリの調整には、ボトルネックの特定と実行パスの最適化が必要です。

- クエリの実行プランを見つけ、クエリが適切なインデックスを使用しているかどうかを確認します。`EXPLAIN` を使用し、各プランの詳細を確認することで、クエリを最適化できます。
- Aurora MySQL バージョン 3 (MySQL 8.0 コミュニティエディションと互換性あり) は `EXPLAIN ANALYZE` ステートメントを使用します。`EXPLAIN ANALYZE` ステートメントは、MySQL がクエリのどこで時間を費やしているのか、またその理由を示すプロファイリングツールです。`EXPLAIN ANALYZE` を使用すると、Aurora MySQL はクエリを計画、準備、実行しつつ、行をカウントし、実行プランのさまざまなポイントで費やされた時間を測定します。クエリが完了すると、`EXPLAIN ANALYZE` は、クエリ結果の代わりにプランとその測定値を印刷します。
- `ANALYZE` ステートメントを使用してスキーマの統計情報を最新の状態に維持してください。クエリオプティマイザは、統計が古いと不適切な実行プランを選択することがあります。これにより、テーブルとインデックスの両方のカーディナリティの推定が不正確になり、クエリのパフォーマンスが低下する可能性があります。innodb_table_stats テーブルの `last_update` 列には、スキーマ統計が最後に更新された時刻が表示されます。これは「古くなっている」ことを示す良い指標です。
- データの分布の歪みなど、テーブルのカーディナリティに考慮されていないその他の問題が発生することもあります。詳細については、MySQL ドキュメントの「InnoDB テーブルの ANALYZE TABLE の複雑度の推定」と「MySQL のヒストグラム統計」を参照してください。

クエリにかかった時間の確認

クエリにかかった時間を決定する方法は次のとおりです。

- プロファイリング
- パフォーマンススキーマ
- クエリオプティマイザ

プロファイリング

デフォルトでは、プロファイリングは無効です。プロファイリングを有効にし、スロークエリを実行してプロファイルを確認します。

```
SET profiling = 1;
Run your query.
SHOW PROFILE;
```

1. 最も多くの時間が費やされているステージを特定します。MySQL ドキュメントの「一般的なスレッド状態」によると、SELECT ステートメントの行の読み取りと処理は、多くの場合、特定のクエリの存続期間中で最も長い実行状態です。EXPLAIN ステートメントを使用すると、MySQL がこのクエリを実行する方法を理解できます。
2. スロークエリログを確認して rows_examined と rows_sent を評価し、各環境でワークロードが類似していることを確認します。詳細については、「Aurora MySQL データベースのログ記録」を参照してください。
3. 特定されたクエリの一部であるテーブルに対して、以下のコマンドを実行します。

   ```
   SHOW TABLE STATUS\G;
   ```

4. 各環境でクエリを実行する前と後に、次の出力をキャプチャします。

   ```
   SHOW GLOBAL STATUS;
   ```

5. 各環境で以下のコマンドを実行して、このサンプルクエリのパフォーマンスに影響する他のクエリ/セッションがないかどうかを確認します。

   ```
   SHOW FULL PROCESSLIST;

   SHOW ENGINE INNODB STATUS\G;
   ```

   サーバー上のリソースがビジー状態になると、クエリを含むサーバー上の他のすべての操作に影響することがあります。また、クエリの実行時に定期的に情報をキャプチャしたり、有用な間隔で情報をキャプチャする cron ジョブを設定することもできます。

パフォーマンススキーマ

パフォーマンススキーマは、パフォーマンスへの影響を最小限に抑えながら、サーバーのランタイムパフォーマンスに関する有用な情報を提供します。これは DB インスタンスに関するスキーマ情報を提供する information_schema とは異なります。詳細については、「Aurora MySQL における Performance Insights の Performance Schema の有効化」を参照してください。

クエリオプティマイザトレース

特定のクエリプランが実行対象として選択された理由を理解するために、MySQL クエリオプティマイザにアクセスするように optimizer_trace をセットアップできます。

オプティマイザトレースを実行すると、オプティマイザが使用できるすべてのパスとその選択に関する詳細な情報が表示されます。

```
SET SESSION OPTIMIZER_TRACE="enabled=on";
SET optimizer_trace_offset=-5, optimizer_trace_limit=5;

-- Run your query.
SELECT * FROM table WHERE x = 1 AND y = 'A';

-- After the query completes:
SELECT * FROM information_schema.OPTIMIZER_TRACE;
SET SESSION OPTIMIZER_TRACE="enabled=off";
```

クエリオプティマイザ設定の確認

Aurora MySQL バージョン 3 (MySQL 8.0 コミュニティエディションと互換性あり) には、Aurora MySQL バージョン 2 (MySQL 5.7 コミュニティエディションと互換性あり) と比較して、オプティマイザ関連の多くの変更があります。optimizer_switch にカスタム値がある場合は、デフォルトの違いを確認して、ワークロードに最適な optimizer_switch 値を設定することをお勧めします。また、Aurora MySQL バージョン 3 で使用できるオプションをテストして、クエリがどのように実行されるかを調べることをお勧めします。

> **Note**
>
> Aurora MySQL バージョン 3 では、innodb_stats_persistent_sample_pages パラメータのコミュニティデフォルト値である 20 を使用しています。

optimizer_switch 値を表示するには、次のコマンドを使用できます。

```
SELECT @@optimizer_switch\G;
```

次の表は、Aurora MySQL バージョン 2 および 3 のデフォルトの optimizer_switch 値を示しています。

設定	Aurora MySQL バージョン 2	Aurora MySQL バージョン 3
batched_key_access	オフ	オフ
block_nested_loop	オン	オン
condition_fanout_filter	オン	オン
derived_condition_pushdown	–	オン
derived_merge	オン	オン
duplicateweedout	オン	オン
engine_condition_pushdown	オン	オン
firstmatch	オン	オン
hash_join	オフ	オン
hash_join_cost_based	オン	–
hypergraph_optimizer	–	オフ
index_condition_pushdown	オン	オン
index_merge	オン	オン
index_merge_intersection	オン	オン
index_merge_sort_union	オン	オン
index_merge_union	オン	オン

設定	Aurora MySQL バージョン 2	Aurora MySQL バージョン 3
loosescan	オン	オン
materialization	オン	オン
mrr	オン	オン
mrr_cost_based	オン	オン
prefer_ordering_index	オン	オン
semijoin	オン	オン
skip_scan	–	オン
subquery_materialization_cost_based	オン	オン
subquery_to_derived	–	オフ
use_index_extensions	オン	オン
use_invisible_indexes	–	オフ

詳細については、MySQL ドキュメントの「切り替え可能な最適化 (MySQL 5.7)」と「切り替え可能な最適化 (MySQL 8.0)」を参照してください。

Amazon Aurora MySQL のリファレンス

このリファレンスには、Aurora MySQL パラメータ、ステータス可変、一般的な SQL 拡張に関する情報や、コミュニティ MySQL データベースエンジンとの相違点が含まれます。

トピック

Aurora MySQL 設定パラメータ

Amazon Aurora MySQL DB クラスターの管理には、他の Amazon RDS DB インスタンスを管理するのと同じ方法 DB パラメータグループのパラメータを使用して管理します。Amazon Aurora は、複数の DB インスタンスを含む DB クラスターを使用する点が、他の DB エンジンとは異なります。そのため、Aurora MySQL DB クラスターの管理に使用するパラメータの中には、クラスター全体に適用されるものがあります。それ以外のパラメータは、DB クラスターの特定の DB インスタンスにのみ適用されます。

クラスターレベルのパラメータを管理するには、DB クラスターのパラメータグループを使用します。インスタンスレベルのパラメータを管理するには、DB パラメータグループを使用します。Aurora MySQL DB クラスターの各 DB インスタンスは、MySQL データベースエンジンと互換性があります。ただし、クラスターレベルでは MySQL データベースエンジンのパラメータの一部を適用します。これらのパラメータは、DB クラスターのパラメータグループを使用して管理します。Aurora DB クラスター内のインスタンスの DB パラメータグループにクラスターレベルのパラメータでは見つけられません。クラスターレベルのパラメータの一覧は、このトピックの後半で紹介します。

クラスターレベルとインスタンスレベルのパラメータは、いずれも AWS Management Console、AWS CLI、または Amazon RDS API を使用して管理できます。クラスターレベルのパラメータとインスタンスレベルのパラメータの管理には、別々のコマンドを使用します。例えば、DB クラスターパラメータグループのクラスターレベルのパラメータを管理するには、CLI の DB クラ

スターパラメータグループを変更する コマンドを使用します。DB クラスターの DB インスタンスの DB パラメータグループのインスタンスレベルのパラメータを管理するには、CLI の modify-db-parameter-group コマンドを使用します。

クラスターレベルのパラメータとインスタンスレベルのパラメータはいずれも、コンソール、CLI、または RDS API を使用して表示できます。例えば、DB クラスターパラメータグループのクラスターレベルのパラメータを表示するには、AWS CLI の describe-db-cluster-parameters コマンドを使用します。DB クラスターの DB インスタンスの DB パラメータグループのインスタンスレベルのパラメータを表示するには、CLI の describe-db-parameters コマンドを使用します。

> **Note**
>
> 各デフォルトのパラメータグループには、パラメータグループ内のすべてのパラメータのデフォルト値が含まれます。パラメータのこの値に「エンジンのデフォルト」がある場合は、実際のデフォルト値については、バージョン固有の MySQL または PostgreSQL のドキュメントを参照してください。
> 特に明記されていない限り、次の表に記載されているパラメータは Aurora MySQL バージョン 2 および 3 で有効です。

DB パラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。Aurora Serverless v1 クラスターのルールおよび制限については、「Aurora Serverless v1 のパラメータグループ」を参照してください。

トピック

- クラスターレベルのパラメータ
- インスタンスレベルのパラメータ
- Aurora MySQL に適用されない MySQL パラメータ
- Aurora MySQL グローバルステータス変数
- Aurora MySQL に適応されない MySQL ステータス可変

クラスターレベルのパラメータ

次の表は、Aurora MySQL DB クラスター全体に適用されるすべてのパラメータを示しています。

パラメータ名	変更可能	コメント
aurora_binlog_read_buffer_size	はい	バイナリログ (binlog) レプリケーションを使用するクラスターにのみ影響します。バイナリログのレプリケーションの詳細については、「Aurora と MySQL との間、または Aurora と別の Aurora DB クラスターとの間のレプリケーション (バイナリログレプリケーション)」を参照してください。Aurora MySQL バージョン 3 から削除されました。
aurora_binlog_replication_max_yield_seconds	はい	バイナリログ (binlog) レプリケーションを使用するクラスターにのみ影響します。バイナリログのレプリケーションの詳細については、「Aurora と MySQL との間、または Aurora と別の Aurora DB クラスターとの間のレプリケーション (バイナリログレプリケーション)」を参照してください。
aurora_binlog_replication_sec_index_parallel_workers	はい	複数のセカンダリインデックスを持つ大きなテーブルのトランザクションを複製するときに、セカンダリインデックスの変更を適用できる並列スレッドの合計数を設定します。パラメータは、デフォルトで 0 (無効) に設定されています。 このパラメータは、Aurora MySQL バージョン 306 以降で使用できます。詳細については、「バイナリログのレプリケーションの最適化」を参照してください。
aurora_binlog_use_large_read_buffer	はい	バイナリログ (binlog) レプリケーションを使用するクラスターにのみ影響しま

パラメータ名	変更可能	コメント
		す。バイナリログのレプリケーションの詳細については、「Aurora と MySQL との間、または Aurora と別の Aurora DB クラスターとの間のレプリケーション (バイナリログレプリケーション)」を参照してください。Aurora MySQL バージョン 3 から削除されました。
aurora_disable_hash_join	はい	Aurora MySQL バージョン 2.09 以降でハッシュ結合最適化を無効にするには、このパラメータを ON に設定します。バージョン 3 ではサポートされていません。詳細については、「Amazon Aurora MySQL のパラレルクエリの使用」を参照してください。
aurora_enable_replica_log_compression	はい	詳細については、「Amazon Aurora MySQL レプリケーションのパフォーマンスに関する考慮事項」を参照してください。Aurora Global Database の一部であるクラスターには適用されません。Aurora MySQL バージョン 3 から削除されました。
aurora_enable_repl_bin_log_filtering	はい	詳細については、「Amazon Aurora MySQL レプリケーションのパフォーマンスに関する考慮事項」を参照してください。Aurora Global Database の一部であるクラスターには適用されません。Aurora MySQL バージョン 3 から削除されました。
aurora_enable_staggered_replica_restart	はい	この設定は、Aurora MySQL バージョン 3 で使用できますが、使用されていません。

パラメータ名	変更可能	コメント
aurora_enable_zdr	はい	Aurora MySQL 2.10 以降では、この設定はデフォルトでオンになっています。詳細については、「ダウンタイムのない再起動 (ZDR) (Amazon Aurora MySQL 用)」を参照してください。
aurora_enhanced_binlog	はい	このパラメータの値を 1 に設定すると、Aurora MySQL バージョン 3.03.1 以降で拡張バイナリログがオンになります。詳細については、「拡張バイナリログ記録の設定」を参照してください。
aurora_jemalloc_background_thread	はい	このパラメータを使用して、バックグラウンドスレッドがメモリメンテナンスオペレーションを実行できるようにします。指定できる値は 0 (無効) と 1 (有効) です。デフォルト値は 0 です。 このパラメータは、Aurora MySQL バージョン 3.05 以降に適用されます。
aurora_jemalloc_dirty_decay_ms	はい	このパラメータを使用して、解放されたメモリを一定時間 (ミリ秒単位) 保持します。メモリを保持すると、より迅速に再利用できます。指定できる値は 0 ~ 18446744073709551615 です。デフォルト値 (0) は、すべてのメモリを解放可能なメモリとしてオペレーティングシステムに返します。 このパラメータは、Aurora MySQL バージョン 3.05 以降に適用されます。

パラメータ名	変更可能	コメント
aurora_jemalloc_tcache_enabled	はい	このパラメータを使用して、スレッドローカルキャッシュ内の小さなメモリリクエスト (最大 32 KiB) を処理し、メモリアリーナをバイパスします。指定できる値は 0 (無効) と 1 (有効) です。デフォルト値は 1 です。 このパラメータは、Aurora MySQL バージョン 3.05 以降に適用されます。
aurora_load_from_s3_role	はい	詳細については、「Amazon S3 バケットのテキストファイルから Amazon Aurora MySQL DB クラスターへのデータのロード」を参照してください。現在、Aurora MySQL バージョン 3 では使用できません。aws_default_s3_role を使用します。
aurora_mask_password_hashes_type	はい	Aurora MySQL 2.11 以降では、この設定はデフォルトでオンになっています。 この設定を使用して、スロークエリログ、監査ログで Aurora MySQL パスワードハッシュをマスクします。許容されている値は 0 と 1 (デフォルト) です。1 に設定されている場合、パスワードは <secret> として記録されます。0 に設定されている場合、パスワードはハッシュ (#) 値として記録されます。

パラメータ名	変更可能	コメント
aurora_select_into_s3_role	はい	詳細については、「Amazon Aurora MySQL DB クラスターから Amazon S3 バケット内のテキストファイルへのデータの保存」を参照してください。現在、Aurora MySQL バージョン 3 では使用できません。aws_default_s3_role を使用します。
authentication_kerberos_caseins_cmp	はい	authentication_kerberos プラグインの大文字と小文字を区別しないユーザー名の比較を制御します。大文字と小文字を区別せずに比較するには、true に設定します。デフォルトでは、大文字と小文字を区別した比較が使用されます (false)。詳細については、「Aurora MySQL での Kerberos 認証の使用」を参照してください。 このパラメータは、Aurora MySQL バージョン 3.03 以降で使用できます。
auto_increment_increment	はい	
auto_increment_offset	はい	
aws_default_lambda_role	はい	詳細については、「Amazon Aurora MySQL DB クラスターからの Lambda 関数の呼び出し」を参照してください。

パラメータ名	変更可能	コメント
aws_default_s3_role	はい	DB クラスターから LOAD DATA FROM S3 ステートメント、LOAD XML FROM S3 ステートメント、または SELECT INTO OUTFILE S3 ステートメントを呼び出すときに使用します。 Aurora MySQL バージョン 2 では、該当するステートメントの aurora_load_from_s3_role または aurora_select_into_s3_role に IAM ロールが指定されていない場合、このパラメータで指定された IAM ロールが使用されます。 Aurora MySQL バージョン 3 では、このパラメータに指定した IAM ロールが常に使用されます。 詳細については、「IAM ロールと Amazon Aurora MySQL DB クラスターの関連付け」を参照してください。
binlog_backup	はい	このパラメータの値を 0 に設定すると、Aurora MySQL バージョン 3.03.1 以降で拡張バイナリログがオンになります。このパラメータは、拡張バイナリログを使用する場合にのみオフにできます。詳細については、「拡張バイナリログ記録の設定」を参照してください。

パラメータ名	変更可能	コメント
binlog_checksum	はい	このパラメータが設定されていない場合、AWS CLI および RDS API は None の値をレポートします。この場合、Aurora MySQL はエンジンのデフォルト値である CRC32 を使用します。これは、チェックサムを無効化する明示的な NONE の設定とは異なります。
binlog-do-db	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
binlog_format	はい	詳細については、「Aurora と MySQL との間、または Aurora と別の Aurora DB クラスターとの間のレプリケーション (バイナリログレプリケーション)」を参照してください。
binlog_group_commit_sync_delay	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
binlog_group_commit_sync_no_delay_count	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
binlog-ignore-db	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
binlog_replication_globaldb	はい	このパラメータの値を 0 に設定すると、Aurora MySQL バージョン 3.03.1 以降で拡張バイナリログがオンになります。このパラメータは、拡張バイナリログを使用する場合にのみオフにできます。詳細については、「拡張バイナリログ記録の設定」を参照してください。
binlog_row_image	いいえ	

パラメータ名	変更可能	コメント
binlog_row_metadata	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
binlog_row_value_options	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
binlog_rows_query_log_events	はい	
binlog_transaction_compression	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
binlog_transaction_compression_level_zstd	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
binlog_transaction_dependency_history_size	はい	このパラメータは、メモリに保持され、特定の行を最後に変更したトランザクションを検索するために使用される行ハッシュ数の上限を設定します。このハッシュ数に達すると、履歴はパージされます。 このパラメータは、Aurora MySQL バージョン 2.12 とバージョン 3 に適用されます。
binlog_transaction_dependency_tracking	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
character-set-client-handshake	はい	
character_set_client	はい	
character_set_connection	はい	
character_set_database	はい	

パラメータ名	変更可能	コメント
character_set_filesystem	はい	
character_set_results	はい	
character_set_server	はい	
collation_connection	はい	
collation_server	はい	
completion_type	はい	
default_storage_engine	いいえ	Aurora MySQL クラスターは、すべてのデータに対して InnoDB ストレージエンジンを使用します。
enforce_gtid_consistency	ときどき	Aurora MySQL バージョン 2 以降で変更可能です。
event_scheduler	はい	イベントスケジューラのステータスを示します。 Aurora MySQL バージョン 3 では、クラスターレベルでのみ変更できます。
gtid-mode	ときどき	Aurora MySQL バージョン 2 以降で変更可能です。
information_schema_stats_expiry	はい	MySQL データベースサーバーがストレージエンジンからデータを取得し、キャッシュ内のデータを置き換えるまでの秒数。指定できる値は `0` ~ `31536000` です。 このパラメータは、Aurora MySQL バージョン 3 に適用されます。

パラメータ名	変更可能	コメント
init_connect	はい	接続するクライアントごとにサーバーによって実行されるコマンド。設定では、接続障害を回避するため、二重引用符 ("") を使用します。次に例を示します。 `SET optimizer_switch="hash_join =off"` Aurora MySQL バージョン 3 では、CONNECTION_ADMIN 権限を持つユーザーにこのパラメータは適用されません。これには Aurora マスターユーザーが含まれます。詳細については、「ロールベースの特権モデル」を参照してください。
innodb_adaptive_hash_index	はい	このパラメータは、Aurora MySQL バージョン 2 および Aurora MySQL バージョン 3 の DB クラスターレベルで修正できます。 Adaptive Hash インデックスは Reader DB インスタンスではサポートされていません。

パラメータ名	変更可能	コメント
innodb_aurora_instant_alter_column_allowed	はい	INSTANT アルゴリズムをグローバルレベルでの ALTER COLUMN オペレーションに使用できるかどうかを制御します。許容値は以下のとおりです。 • 0 – INSTANT アルゴリズムは ALTER COLUMN オペレーション (OFF) には使用できません。他のアルゴリズムに戻します。 • 1 – INSTANT アルゴリズムは ALTER COLUMN オペレーション (ON) には使用できません。これは、デフォルト値です。 詳細については、MySQL ドキュメントの「列オペレーション」を参照してください。 このパラメータは、Aurora MySQL バージョン 3.05 以降に適用されます。
innodb_autoinc_lock_mode	はい	
innodb_checksums	いいえ	Aurora MySQL バージョン 3 から削除されました。
innodb_cmp_per_index_enabled	はい	
innodb_commit_concurrency	はい	
innodb_data_home_dir	いいえ	Aurora MySQL は、直接ファイルシステムにアクセスしないマネージドインスタンスを使用します。

パラメータ名	変更可能	コメント
innodb_deadlock_detect	はい	このオプションは、Aurora MySQL バージョン 2.11 以降とバージョン 3 でデッドロック検出を無効化するために使用されます。 高並行性システムでは、多数のスレッドが同じロックを待機すると、デッドロック検出によって速度が低下する可能性があります。MySQL パラメータの詳細については、MySQL のドキュメントを参照してください。
innodb_default_row_format	はい	このパラメータは、InnoDB テーブル (ユーザー作成 InnoDB 一時テーブルを含む) のデフォルトの行形式を定義します。Aurora MySQL バージョン 2 と 3 に適用されます。 値は DYNAMIC、COMPACT、または REDUNDANT。 になります。
innodb_file_per_table	はい	このパラメータは、テーブルストレージの編成方法に影響します。詳細については、「ストレージのスケーリング」を参照してください。

パラメータ名	変更可能	コメント
innodb_flush_log_at_trx_commit	はい	デフォルト値の 1 を使用することを強くお勧めします。 Aurora MySQL バージョン 3 では、このパラメータを 1 以外の値に設定する前に、innodb_trx_commit_allow_data_loss の値を 1 に設定する必要があります。 詳細については、「ログバッファをフラッシュする頻度の設定」を参照してください。
innodb_ft_max_token_size	はい	
innodb_ft_min_token_size	はい	
innodb_ft_num_word_optimize	はい	
innodb_ft_sort_pll_degree	はい	
innodb_online_alter_log_max_size	はい	
innodb_optimize_fulltext_only	はい	
innodb_page_size	いいえ	
innodb_print_all_deadlocks	はい	有効にすると、すべての InnoDB のデッドロックに関する情報が Aurora MySQL エラーログに記録されます。詳細については、「Aurora MySQL デッドロックの最小化とトラブルシューティング」を参照してください。

パラメータ名	変更可能	コメント
innodb_purge_batch_size	はい	
innodb_purge_threads	はい	
innodb_rollback_on_timeout	はい	
innodb_rollback_segments	はい	
innodb_spin_wait_delay	はい	
innodb_strict_mode	はい	
innodb_support_xa	はい	Aurora MySQL バージョン 3 から削除されました。
innodb_sync_array_size	はい	
innodb_sync_spin_loops	はい	
innodb_stats_include_delete_marked	はい	このパラメータが有効なとき、InnoDB はパーシステントオプティマイザ統計の計算時に削除マーク付きのレコードを含めます。 このパラメータは、Aurora MySQL バージョン 2.12 とバージョン 3 に適用されます。
innodb_table_locks	はい	

パラメータ名	変更可能	コメント
innodb_trx_commit_allow_data_loss	はい	Aurora MySQL バージョン 3 では、innodb_flush_log_at_trx_commit の値を変更できるように、このパラメータの値を 1 に設定します。 innodb_trx_commit_allow_data_loss の初期値は 0 です。 詳細については、「ログバッファをフラッシュする頻度の設定」を参照してください。
innodb_undo_directory	いいえ	Aurora MySQL は、直接ファイルシステムにアクセスしないマネージドインスタンスを使用します。
internal_tmp_disk_storage_engine	はい	どのインメモリストレージエンジンを内部一時テーブルに使用するかを制御します。指定できる値は INNODB と MYISAM です。 このパラメータは、Aurora MySQL バージョン 2 に適用されます。
internal_tmp_mem_storage_engine	はい	どのインメモリストレージエンジンを内部一時テーブルに使用するかを制御します。指定できる値は MEMORY と TempTable です。 このパラメータは、Aurora MySQL バージョン 3 に適用されます。
key_buffer_size	はい	MyISAM テーブルのキーキャッシュ。詳しい情報については、「keycache->cache_lock ミューテックス」を参照してください。

パラメータ名	変更可能	コメント
lc_time_names	はい	
log_error_suppression_list	はい	MySQL エラーログに記録されていないエラーコードのリストを指定します。これにより、重大でない特定のエラー条件を無視することで、エラーログをクリーンな状態に保つことができます。詳細については、MySQL ドキュメントの「log_error_suppression_list」を参照してください。 このパラメータは、Aurora MySQL バージョン 3.03 以降に適用されます。
low_priority_updates	はい	INSERT、UPDATE、DELETE、LOCK TABLE WRITE オペレーションは、保留中の SELECT オペレーションがなくなるまで待機します。このパラメータは、テーブルレベルのロック (MyISAM、MEMORY、MERGE) のみを使用するストレージエンジンにのみ影響します。 このパラメータは、Aurora MySQL バージョン 3 に適用されます。

パラメータ名	変更可能	コメント
lower_case_table_names	はい (Aurora MySQL バージョン 2) クラスター作成時のみ (Aurora MySQL バージョン 3)	Aurora MySQL バージョン 2.10 以降では、この設定を変更し、ライターインスタンスを再起動した後、すべてのリーダーインスタンスを再起動してください。詳細については、「読み取り可用性機能のある Aurora クラスターの再起動」を参照してください。 Aurora MySQL バージョン 3 では、 このパラメータ値はクラスターの作成時に永続的に設定されます。このオプションにデフォルト以外の値を使用する場合は、アップグレードする前に Aurora MySQL バージョン 3 カスタムパラメータグループを設定し、バージョン 3 クラスターを作成するスナップショットの復元操作中にパラメータグループを指定します。 Aurora MySQL に基づく Aurora グローバルデータベースでは、lower_case_table_names パラメータがオンの場合、Aurora MySQL バージョン 2 からバージョン 3 へのインプレースアップグレードを実行できません。使用できる方法の詳細については、「メジャーバージョンのアップグレード」を参照してください。
master-info-repository	はい	Aurora MySQL バージョン 3 から削除されました。

パラメータ名	変更可能	コメント
master_verify_checksum	はい	Aurora MySQL バージョン 2。source_verify_checksum を Aurora MySQL バージョン 3 で使用する。
max_delayed_threads	はい	INSERT DELAYED ステートメントを処理するスレッドの最大数を設定します。 このパラメータは、Aurora MySQL バージョン 3 に適用されます。
max_error_count	はい	表示用に保存するエラーメッセージ、警告メッセージ、およびメモメッセージの最大数。 このパラメータは、Aurora MySQL バージョン 3 に適用されます。
max_execution_time	はい	実行中の SELECT ステートメントのタイムアウトをミリ秒単位で表します。値は 0 ~ 18446744073709551615 の範囲で指定できます。0 に設定すると、タイムアウトは発生しません。 詳細については、MySQL ドキュメントの「max_execution_time」を参照してください。
min_examined_row_limit	はい	このパラメータを使用すると、指定した行数よりも少ない行数を調べるクエリがログに記録されないようにします。 このパラメータは、Aurora MySQL バージョン 3 に適用されます。

パラメータ名	変更可能	コメント
partial_revokes	いいえ	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
preload_buffer_size	はい	インデックスをプリロードするときに割り当てられるバッファのサイズ。 このパラメータは、Aurora MySQL バージョン 3 に適用されます。
query_cache_type	はい	Aurora MySQL バージョン 3 から削除されました。

パラメータ名	変更可能	コメント
read_only	はい	このパラメータがオンにされると、サーバーはレプリカスレッドによって実行される更新以外の更新を許可しません。 Aurora MySQL バージョン 2 の有効な値は以下のとおりです。 • 0 – OFF • 1 – ON • {TrueIfReplica} — リードレプリカの場合 ON。これは、デフォルト値です。 • {TrueIfClusterReplica} — クロスリージョンリードレプリカ、Aurora グローバルデータベースのセカンダリクラスター、ブルー/グリーンデプロイなどのレプリカクラスターの場合 ON。 Aurora MySQL バージョン 3 の有効な値は以下のとおりです。 • 0 – OFF。これは、デフォルト値です。 • 1 – ON • {TrueIfClusterReplica} — クロスリージョンリードレプリカ、Aurora グローバルデータベースのセカンダリクラスター、ブルー/グリーンデプロイなどのレプリカクラスターの場合 ON。

パラメータ名	変更可能	コメント
		Aurora MySQL バージョン 3 では、CONNECTION_ADMIN 権限を持つユーザーにこのパラメータは適用されません。これには Aurora マスターユーザーが含まれます。詳細については、「ロールベースの特権モデル」を参照してください。
relay-log-space-limit	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
replica_parallel_type	はい	このパラメータを使用すると、整合性を損なうことなく、すでに準備段階にあるコミットされていないすべてのスレッドのレプリカを並行して実行できます。Aurora MySQL バージョン 3 に適用されます。 Aurora MySQL バージョン 3.03.* 以前では、デフォルト値は DATABASE です。Aurora MySQL バージョン 3.04 以降では、デフォルト値は LOGICAL_CLOCK です。
replica_preserve_commit_order	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
replica_transaction_retries	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。

パラメータ名	変更可能	コメント
replica_type_conversions	はい	このパラメータは、レプリカで使用されるタイプ変換を決定します。指定できる値は、ALL_LOSSY 、ALL_NON_LOSSY 、ALL_SIGNED 、および ALL_UNSIGNED です。詳細については、MySQL ドキュメントの「ソースとレプリカのテーブル定義が異なるレプリケーション」を参照してください。 このパラメータは、Aurora MySQL バージョン 3 に適用されます。
replicate-do-db	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
replicate-do-table	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
replicate-ignore-db	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
replicate-ignore-table	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
replicate-wild-do-table	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
replicate-wild-ignore-table	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
require_secure_transport	はい	このパラメータは、Aurora MySQL バージョン 2 および 3 に適用されます。詳細については、「Aurora MySQL DB クラスターでの TLS の使用」を参照してください。

パラメータ名	変更可能	コメント
rpl_read_size	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
server_audit_events	はい	
server_audit_excl_users	はい	
server_audit_incl_users	はい	
server_audit_logging	はい	Amazon CloudWatch Logs への口グのアップロードの手順については、Amazon CloudWatch Logs への Amazon Aurora MySQL 口グの発行 を参照してください。
server_audit_logs_upload	はい	[高度な監査] を有効にし、このパラメータを 1 に設定することで、監査口グを CloudWatch Logs にパブリッシュできます。server_audit_logs_upload パラメータのデフォルト値は 0 です。 詳細については、「Amazon CloudWatch Logs への Amazon Aurora MySQL 口グの発行」を参照してください。
server_id	いいえ	
skip-character-set-client-handshake	はい	
skip_name_resolve	いいえ	
slave-skip-errors	はい	MySQL 5.7 の互換性を備えた Aurora MySQL バージョン 2 クラスターにのみ適用されます。

パラメータ名	変更可能	コメント
source_verify_checksum	はい	Aurora MySQL バージョン 3
sync_frm	はい	Aurora MySQL バージョン 3 から削除されました。
thread_cache_size	はい	キャッシュされるスレッドの数。このパラメータは、Aurora MySQL バージョン 2 および 3 に適用されます。
time_zone	はい	デフォルトでは、Aurora DB クラスターのタイムゾーンは協定世界時 (UTC) です。代わりに、DB クラスターのインスタンスのタイムゾーンをアプリケーションのローカルタイムゾーンに設定できます。詳細については、「Amazon Aurora DB クラスターのローカルタイムゾーン」を参照してください。
tls_version	はい	詳細については、「Aurora MySQL の TLS バージョン」を参照してください。

インスタンスレベルのパラメータ

次の表は、Aurora MySQL DB クラスターの特定の DB インスタンスに適用されるパラメータの一覧です。

パラメータ名	変更可能	コメント
activate_all_roles_on_login	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
allow-suspicious-udfs	いいえ	
aurora_disable_hash_join	はい	Aurora MySQL バージョン 2.09 以降でハッシュ結合最適化を無効にするに

パラメータ名	変更可能	コメント
		は、このパラメータを ON に設定します。バージョン 3 ではサポートされていません。詳細については、「Amazon Aurora MySQL のパラレルクエリの使用」を参照してください。
aurora_lab_mode	はい	詳細については、「Amazon Aurora MySQL ラボモード」を参照してください。Aurora MySQL バージョン 3 から削除されました。
aurora_oom_response	はい	このパラメータは、Aurora MySQL バージョン 2 および 3 でサポートされています。詳細については、「Aurora MySQL データベースのメモリ不足の問題のトラブルシューティング」を参照してください。
aurora_parallel_query	はい	Aurora MySQL バージョン 2.09 以降では、ON に設定してパラレルクエリを有効にします。これらのバージョンでは、古い aurora_pq パラメータは使用されません。詳細については、「Amazon Aurora MySQL のパラレルクエリの使用」を参照してください。
aurora_pq	はい	Aurora MySQL バージョン 2.09 より前の特定の DB インスタンスでは、パラレルクエリをオフにするには、OFF に設定します。バージョン 2.09 以降では、代わりに aurora_parallel_query を使用してパラレルクエリのオンとオフを切り替えます。詳細については、「Amazon Aurora MySQL のパラレルクエリの使用」を参照してください。

パラメータ名	変更可能	コメント
aurora_read_replica_read_committed	はい	Aurora レプリカの READ COMMITTED 分離レベルを有効化し、長時間実行クエリ中のパージラグを削減するように分離動作を変更します。動作の変更点および変更によるクエリ結果への影響を理解している場合にのみ、この設定を有効にしてください。たとえば、この設定では MySQL のデフォルトよりも厳密でない分離を使用します。Aurora はクエリ実行中にテーブルを再編成するため、これが有効なとき、長時間実行クエリには同じ行の複数のコピーが表示されることがあります。詳細については、「Aurora MySQL の分離レベル」を参照してください。
aurora_tmptable_enable_per_table_limit	はい	Aurora MySQL バージョン 3.04 以降で、TempTable ストレージエンジンによって作成されるメモリ内一時テーブルの最大サイズを tmp_table_size パラメータが制御するかどうかを決定します。 詳細については、「内部メモリ内一時テーブルのサイズを制限する」を参照してください。

パラメータ名	変更可能	コメント
aurora_use_vector_instructions	はい	このパラメータが有効なとき、Aurora MySQL は最新の CPU が提供する最適化されたベクトル処理命令を使用して、I/O 集約型ワークロードのパフォーマンスを向上させます。 Aurora MySQL version 3.05 以降では、この設定はデフォルトで有効になっています。
autocommit	はい	
automatic_sp_privileges	はい	
back_log	はい	
basedir	いいえ	Aurora MySQL は、直接ファイルシステムにアクセスしないマネージドインスタンスを使用します。
binlog_cache_size	はい	
binlog_max_flush_queue_time	はい	
binlog_order_commits	はい	
binlog_stmt_cache_size	はい	
binlog_transaction_compression	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
binlog_transaction_compression_level_zstd	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
bulk_insert_buffer_size	はい	
concurrent_insert	はい	

パラメータ名	変更可能	コメント
connect_timeout	はい	
core-file	いいえ	Aurora MySQL は、直接ファイルシステムにアクセスしないマネージドインスタンスを使用します。
datadir	いいえ	Aurora MySQL は、直接ファイルシステムにアクセスしないマネージドインスタンスを使用します。
default_authentication_plugin	いいえ	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
default_time_zone	いいえ	
default_tmp_storage_engine	はい	一時テーブルのデフォルトのストレージエンジン。
default_week_format	はい	
delay_key_write	はい	
delayed_insert_limit	はい	
delayed_insert_timeout	はい	
delayed_queue_size	はい	
div_precision_increment	はい	
end_markers_in_json	はい	
eq_range_index_dive_limit	はい	
event_scheduler	ときどき	イベントスケジューラのステータスを示します。 Aurora MySQL バージョン 3 では、クラスターレベルでのみ変更できます。

パラメータ名	変更可能	コメント
explicit_defaults_for_timestamp	はい	
flush	いいえ	
flush_time	はい	
ft_boolean_syntax	いいえ	
ft_max_word_len	はい	
ft_min_word_len	はい	
ft_query_expansion_limit	はい	
ft_stopword_file	はい	
general_log	はい	CloudWatch Logs へのログのアップロードの手順については、Amazon CloudWatch Logs への Amazon Aurora MySQL ログの発行 を参照してください。
general_log_file	いいえ	Aurora MySQL は、直接ファイルシステムにアクセスしないマネージドインスタンスを使用します。
group_concat_max_len	はい	
host_cache_size	はい	

パラメータ名	変更可能	コメント
init_connect	はい	接続するクライアントごとにサーバーによって実行されるコマンド。設定では、接続障害を回避するため、二重引用符 ("") を使用します。次に例を示します。 `SET optimizer_switch="hash_join =off"` Aurora MySQL バージョン 3 では、Aurora マスターユーザーなど、CONNECTION_ADMIN 権限を持つユーザーには、このパラメータは適用されません。詳細については、「ロールベースの特権モデル」を参照してください。
innodb_adaptive_hash_index	はい	このパラメータは、Aurora MySQL バージョン 2 の DB インスタンスレベルに適用されます。Aurora MySQL バージョン 3 では、DB クラスターレベルでのみ変更できます。 Adaptive Hash インデックスは Reader DB インスタンスではサポートされていません。
innodb_adaptive_max_sleep_delay	はい	Aurora では、innodb_thread_concurrency は常に 0 であるため、このパラメータを変更しても影響はありません。

パラメータ名	変更可能	コメント
innodb_aurora_max_partitions_for_range	はい	永続的な統計情報が得られない場合は、このパラメータを使用してパーティション分割テーブルの行数計算のパフォーマンスを向上させることができます。 この値は 0 ~ 8192 に設定できます。この値によって、行数の計算時にチェックするパーティションの数が決まります。デフォルト値は 0 で、MySQL のデフォルト動作と同じく、すべてのパーティションを使用していると推定されます。 このパラメータは、Aurora MySQL バージョン 3.03.1 以降で使用できます。
innodb_autoextend_increment	はい	
innodb_buffer_pool_dump_at_shutdown	いいえ	
innodb_buffer_pool_dump_now	いいえ	
innodb_buffer_pool_filename	いいえ	
innodb_buffer_pool_load_abort	いいえ	
innodb_buffer_pool_load_at_startup	いいえ	
innodb_buffer_pool_load_now	いいえ	

パラメータ名	変更可能	コメント
innodb_buffer_pool_size	はい	デフォルト値は式により表されます。式内で DBInstanceClassMemory 値がどのように計算されるかについては、「DB パラメータ式の変数」を参照してください。
innodb_change_buffer_max_size	いいえ	Aurora MySQL は、は InnoDB 変更バッファをまったく使用しません。
innodb_compression_failure_threshold_pct	はい	
innodb_compression_level	はい	
innodb_compression_pad_pct_max	はい	
innodb_concurrency_tickets	はい	Aurora では innodb_thread_concurrency が常に 0 であるため、このパラメータを変更しても影響はありません。
innodb_deadlock_detect	はい	このオプションは、Aurora MySQL バージョン 2.11 以降とバージョン 3 でデッドロック検出を無効化するために使用されます。 高並行性システムでは、多数のスレッドが同じロックを待機すると、デッドロック検出によって速度が低下する可能性があります。MySQL パラメータの詳細については、MySQL のドキュメントを参照してください。
innodb_file_format	はい	Aurora MySQL バージョン 3 から削除されました。

パラメータ名	変更可能	コメント
innodb_flushing_avg_loops	いいえ	
innodb_force_load_corrupted	いいえ	
innodb_ft_aux_table	はい	
innodb_ft_cache_size	はい	
innodb_ft_enable_stopword	はい	
innodb_ft_server_stopword_table	はい	
innodb_ft_user_stopword_table	はい	
innodb_large_prefix	はい	Aurora MySQL バージョン 3 から削除されました。
innodb_lock_wait_timeout	はい	
innodb_log_compressed_pages	いいえ	
innodb_lru_scan_depth	はい	
innodb_max_purge_lag	はい	
innodb_max_purge_lag_delay	はい	
innodb_monitor_disable	はい	
innodb_monitor_enable	はい	
innodb_monitor_reset	はい	
innodb_monitor_reset_all	はい	

パラメータ名	変更可能	コメント
innodb_old_blocks_pct	はい	
innodb_old_blocks_time	はい	
innodb_open_files	はい	
innodb_print_all_deadlocks	はい	有効にすると、すべての InnoDB のデッドロックに関する情報が Aurora MySQL エラーログに記録されます。詳細については、「Aurora MySQL デッドロックの最小化とトラブルシューティング」を参照してください。
innodb_random_read_ahead	はい	
innodb_read_ahead_threshold	はい	
innodb_read_io_threads	いいえ	
innodb_read_only	いいえ	Aurora MySQL は、クラスターの種類に基づき、DB インスタンスの読み取り専用と読み書きの状態を管理します。例えば、プロビジョンされたクラスターに読み書きの DB インスタンス (プライマリインスタンス) が 1 つあり、クラスターのそれ以外のインスタンスは読み取り専用 (Aurora レプリカ) です。
innodb_replication_delay	はい	
innodb_sort_buffer_size	はい	
innodb_stats_auto_recalc	はい	
innodb_stats_method	はい	
innodb_stats_on_metadata	はい	

パラメータ名	変更可能	コメント
innodb_stats_persistent	はい	
innodb_stats_persistent_sample_pages	はい	
innodb_stats_transient_sample_pages	はい	
innodb_thread_concurrency	いいえ	
innodb_thread_sleep_delay	はい	Aurora では、innodb_thread_concurrency は常に 0 であるため、このパラメータを変更しても影響はありません。
interactive_timeout	はい	Aurora は interactive_timeout と wait_timeout の最小値を評価します。次に、その最小値をタイムアウトとして使用して、対話型と非対話型の両方のアイドル状態のセッションをすべて終了します。
internal_tmp_disk_storage_engine	はい	どのインメモリストレージエンジンを内部一時テーブルに使用するかを制御します。指定できる値は INNODB と MYISAM です。 このパラメータは、Aurora MySQL バージョン 2 に適用されます。
internal_tmp_mem_storage_engine	はい	どのインメモリストレージエンジンを内部一時テーブルに使用するかを制御します。指定できる値は MEMORY と TempTable です。 このパラメータは、Aurora MySQL バージョン 3 に適用されます。

パラメータ名	変更可能	コメント
join_buffer_size	はい	
keep_files_on_create	はい	
key_buffer_size	はい	MyISAM テーブルのキーキャッシュ。詳しい情報については、「keycache->cache_lock ミューテックス」を参照してください。
key_cache_age_threshold	はい	
key_cache_block_size	はい	
key_cache_division_limit	はい	
local_infile	はい	
lock_wait_timeout	はい	
log-bin	いいえ	binlog_format を STATEMENT、MIXED、または ROW に設定すると、log-bin は自動的に ON に設定されます。binlog_format を OFF に設定すると、log-bin は自動的に OFF に設定されます。詳細については、「Aurora と MySQL との間、または Aurora と別の Aurora DB クラスターとの間のレプリケーション (バイナリログレプリケーション)」を参照してください。
log_bin_trust_function_creators	はい	
log_bin_use_v1_row_events	はい	Aurora MySQL バージョン 3 から削除されました。

パラメータ名	変更可能	コメント
log_error	いいえ	
log_error_suppression_list	はい	MySQL エラーログに記録されていないエラーコードのリストを指定します。これにより、重大でない特定のエラー条件を無視することで、エラーログをクリーンな状態に保つことができます。詳細については、MySQL ドキュメントの「log_error_suppression_list」を参照してください。 このパラメータは、Aurora MySQL バージョン 3.03 以降に適用されます。
log_output	はい	
log_queries_not_using_indexes	はい	
log_slave_updates	いいえ	Aurora MySQL バージョン 2。log_replica_updates を Aurora MySQL バージョン 3 で使用する。
log_replica_updates	いいえ	Aurora MySQL バージョン 3
log_throttle_queries_not_using_indexes	はい	
log_warnings	はい	Aurora MySQL バージョン 3 から削除されました。
long_query_time	はい	

パラメータ名	変更可能	コメント
low_priority_updates	はい	INSERT、UPDATE、DELETE、LOCK TABLE WRITE オペレーションは、保留中の SELECT オペレーションがなくなるまで待機します。このパラメータは、テーブルレベルのロック (MyISAM、MEMORY、MERGE) のみを使用するストレージエンジンにのみ影響します。 このパラメータは、Aurora MySQL バージョン 3 に適用されます。
max_allowed_packet	はい	
max_binlog_cache_size	はい	
max_binlog_size	いいえ	
max_binlog_stmt_cache_size	はい	
max_connect_errors	はい	
max_connections	はい	デフォルト値は式により表されます。式内で DBInstanceClassMemory 値がどのように計算されるかについては、「DB パラメータ式の変数」を参照してください。インスタンスクラスに応じたデフォルト値については、「Aurora MySQL DB インスタンスへの最大接続数」を参照してください。
max_delayed_threads	はい	INSERT DELAYED ステートメントを処理するスレッドの最大数を設定します。 このパラメータは、Aurora MySQL バージョン 3 に適用されます。

パラメータ名	変更可能	コメント
max_error_count	はい	表示用に保存するエラーメッセージ、警告メッセージ、およびメモメッセージの最大数。 このパラメータは、Aurora MySQL バージョン 3 に適用されます。
max_execution_time	はい	実行中の SELECT ステートメントのタイムアウトをミリ秒単位で表します。値は 0 ~ 18446744073709551615 の範囲で指定できます。0 に設定すると、タイムアウトは発生しません。 詳細については、MySQL ドキュメントの「max_execution_time」を参照してください。
max_heap_table_size	はい	
max_insert_delayed_threads	はい	
max_join_size	はい	
max_length_for_sort_data	はい	Aurora MySQL バージョン 3 から削除されました。
max_prepared_stmt_count	はい	
max_seeks_for_key	はい	
max_sort_length	はい	
max_sp_recursion_depth	はい	
max_tmp_tables	はい	Aurora MySQL バージョン 3 から削除されました。
max_user_connections	はい	

パラメータ名	変更可能	コメント
max_write_lock_count	はい	
metadata_locks_cache_size	はい	Aurora MySQL バージョン 3 から削除されました。
min_examined_row_limit	はい	このパラメータを使用すると、指定した行数よりも少ない行数を調べるクエリがログに記録されないようにします。 このパラメータは、Aurora MySQL バージョン 3 に適用されます。
myisam_data_pointer_size	はい	
myisam_max_sort_file_size	はい	
myisam_mmap_size	はい	
myisam_sort_buffer_size	はい	
myisam_stats_method	はい	
myisam_use_mmap	はい	
net_buffer_length	はい	
net_read_timeout	はい	
net_retry_count	はい	
net_write_timeout	はい	
old-style-user-limits	はい	
old_passwords	はい	Aurora MySQL バージョン 3 から削除されました。
optimizer_prune_level	はい	

パラメータ名	変更可能	コメント
optimizer_search_depth	はい	
optimizer_switch	はい	このスイッチを使用する Aurora MySQL 機能の詳細については、「Amazon Aurora MySQL を使用する際のベストプラクティス」を参照してください。
optimizer_trace	はい	
optimizer_trace_features	はい	
optimizer_trace_limit	はい	
optimizer_trace_max_mem_size	はい	
optimizer_trace_offset	はい	
performance-schema-consumer-events-waits-current	はい	
performance-schema-instrument	はい	
performance_schema	はい	
performance_schema_accounts_size	はい	
performance_schema_consumer_global_instrumentation	はい	
performance_schema_consumer_thread_instrumentation	はい	
performance_schema_consumer_events_stages_current	はい	

パラメータ名	変更可能	コメント
performance_schema_consumer_events_stages_history	はい	
performance_schema_consumer_events_stages_history_long	はい	
performance_schema_consumer_events_statements_current	はい	
performance_schema_consumer_events_statements_history	はい	
performance_schema_consumer_events_statements_history_long	はい	
performance_schema_consumer_events_waits_history	はい	
performance_schema_consumer_events_waits_history_long	はい	
performance_schema_consumer_statements_digest	はい	
performance_schema_digests_size	はい	
performance_schema_events_stages_history_long_size	はい	
performance_schema_events_stages_history_size	はい	

パラメータ名	変更可能	コメント
performance_schema_events_statements_history_long_size	はい	
performance_schema_events_statements_history_size	はい	
performance_schema_events_transactions_history_long_size	はい	
performance_schema_events_transactions_history_size	はい	
performance_schema_events_waits_history_long_size	はい	
performance_schema_events_waits_history_size	はい	
performance_schema_hosts_size	はい	
performance_schema_max_cond_classes	はい	
performance_schema_max_cond_instances	はい	
performance_schema_max_digest_length	はい	
performance_schema_max_file_classes	はい	
performance_schema_max_file_handles	はい	

パラメータ名	変更可能	コメント
performance_schema_max_file_instances	はい	
performance_schema_max_index_stat	はい	
performance_schema_max_memory_classes	はい	
performance_schema_max_metadata_locks	はい	
performance_schema_max_mutex_classes	はい	
performance_schema_max_mutex_instances	はい	
performance_schema_max_prepared_statements_instances	はい	
performance_schema_max_program_instances	はい	
performance_schema_max_rwlock_classes	はい	
performance_schema_max_rwlock_instances	はい	
performance_schema_max_socket_classes	はい	
performance_schema_max_socket_instances	はい	
performance_schema_max_sql_text_length	はい	

パラメータ名	変更可能	コメント
performance_schema_max_stage_classes	はい	
performance_schema_max_statement_classes	はい	
performance_schema_max_statement_stack	はい	
performance_schema_max_table_handles	はい	
performance_schema_max_table_instances	はい	
performance_schema_max_table_lock_stat	はい	
performance_schema_max_thread_classes	はい	
performance_schema_max_thread_instances	はい	
performance_schema_session_connect_attrs_size	はい	
performance_schema_setup_actors_size	はい	
performance_schema_setup_objects_size	はい	

パラメータ名	変更可能	コメント
performance_schema_show_processlist	はい	このパラメータは、使用する SHOW PROCESSLIST 実装を決定します。 • デフォルトの実装では、グローバルミューテックスを保持したまま、スレッドマネージャー内からアクティブなスレッドを反復処理します。これにより、特にビジーなシステムでは、パフォーマンスが低下する可能性があります。 • 代替の SHOW PROCESSLIST 実装は、パフォーマンススキーマ processlist テーブルに基づきます。この実装では、スレッドマネージャーではなくパフォーマンススキーマからアクティブなスレッドデータをクエリするため、ミューテックスは必要ありません。 このパラメータは、Aurora MySQL バージョン 2.12 とバージョン 3 に適用されます。
performance_schema_users_size	はい	
pid_file	いいえ	
plugin_dir	いいえ	Aurora MySQL は、直接ファイルシステムにアクセスしないマネージドインスタンスを使用します。

パラメータ名	変更可能	コメント
port	いいえ	Aurora MySQL は接続プロパティを管理し、クラスター内のすべての DB インスタンスに対して一貫した設定を適用します。
preload_buffer_size	はい	インデックスをプリロードするときに割り当てられるバッファのサイズ。 このパラメータは、Aurora MySQL バージョン 3 に適用されます。
profiling_history_size	はい	
query_alloc_block_size	はい	
query_cache_limit	はい	Aurora MySQL バージョン 3 から削除されました。
query_cache_min_res_unit	はい	Aurora MySQL バージョン 3 から削除されました。
query_cache_size	はい	デフォルト値は式により表されます。式内で DBInstanceClassMemory 値がどのように計算されるかについては、「DB パラメータ式の変数」を参照してください。 Aurora MySQL バージョン 3 から削除されました。
query_cache_type	はい	Aurora MySQL バージョン 3 から削除されました。
query_cache_wlock_invalidate	はい	Aurora MySQL バージョン 3 から削除されました。
query_prealloc_size	はい	

パラメータ名	変更可能	コメント
range_alloc_block_size	はい	
read_buffer_size	はい	

パラメータ名	変更可能	コメント
read_only	はい	このパラメータがオンにされると、サーバーはレプリカスレッドによって実行される更新以外の更新を許可しません。 Aurora MySQL バージョン 2 の有効な値は以下のとおりです。 • 0 – OFF • 1 – ON • {TrueIfReplica} — リードレプリカの場合 ON。これは、デフォルト値です。 • {TrueIfClusterReplica} — クロスリージョンリードレプリカ、Aurora グローバルデータベースのセカンダリクラスター、ブルー/グリーンデプロイなどのレプリカクラスターのインスタンスの場合 ON。 Aurora MySQL バージョン 2 の DB クラスターパラメータグループを使用して、フェイルオーバー時に read_only パラメータが新しいライターインスタンスに適用されていることを確認することをお勧めします。 **Note** Aurora MySQL はすべてのリーダーで innodb_read_only を 1 に設定しているため、リーダーインスタンスは常に読み取り専用です。したがっ

パラメータ名	変更可能	コメント
		て、read_only はリーダーインスタンスでは冗長になります。 Aurora MySQL バージョン 3 からインスタンスレベルで削除されました。
read_rnd_buffer_size	はい	
relay-log	いいえ	
relay_log_info_repository	はい	Aurora MySQL バージョン 3 から削除されました。
relay_log_recovery	いいえ	
replica_checkpoint_group	はい	Aurora MySQL バージョン 3
replica_checkpoint_period	はい	Aurora MySQL バージョン 3
replica_parallel_workers	はい	Aurora MySQL バージョン 3
replica_pending_jobs_size_max	はい	Aurora MySQL バージョン 3
replica_skip_errors	はい	Aurora MySQL バージョン 3
replica_sql_verify_checksum	はい	Aurora MySQL バージョン 3
safe-user-create	はい	

パラメータ名	変更可能	コメント
secure_auth	はい	Aurora MySQL バージョン 2 では、このパラメータは常にオンになっています。オフにしようとするとエラーが発生します。 Aurora MySQL バージョン 3 から削除されました。
secure_file_priv	いいえ	Aurora MySQL は、直接ファイルシステムにアクセスしないマネージドインスタンスを使用します。
show_create_table_verbosity	はい	この変数を有効にすると、SHOW_CREATE_TABLE は、デフォルトの形式であるかどうかに関係なく、ROW_FORMAT を表示します。 このパラメータは、Aurora MySQL バージョン 2.12 とバージョン 3 に適用されます。
skip-slave-start	いいえ	
skip_external_locking	いいえ	
skip_show_database	はい	
slave_checkpoint_group	はい	Aurora MySQL バージョン 2。replica_checkpoint_group を Aurora MySQL バージョン 3 で使用する。
slave_checkpoint_period	はい	Aurora MySQL バージョン 2。replica_checkpoint_period を Aurora MySQL バージョン 3 で使用する。

パラメータ名	変更可能	コメント
slave_parallel_workers	はい	Aurora MySQL バージョン 2。replica_parallel_workers を Aurora MySQL バージョン 3 で使用する。
slave_pending_jobs_size_max	はい	Aurora MySQL バージョン 2。replica_pending_jobs_size_max を Aurora MySQL バージョン 3 で使用する。
slave_sql_verify_checksum	はい	Aurora MySQL バージョン 2。replica_sql_verify_checksum を Aurora MySQL バージョン 3 で使用する。
slow_launch_time	はい	
slow_query_log	はい	CloudWatch Logs へのログのアップロードの手順については、Amazon CloudWatch Logs への Amazon Aurora MySQL ログの発行 を参照してください。
slow_query_log_file	いいえ	Aurora MySQL は、直接ファイルシステムにアクセスしないマネージドインスタンスを使用します。
socket	いいえ	
sort_buffer_size	はい	
sql_mode	はい	
sql_select_limit	はい	
stored_program_cache	はい	

パラメータ名	変更可能	コメント
sync_binlog	いいえ	
sync_master_info	はい	
sync_source_info	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。
sync_relay_log	はい	Aurora MySQL バージョン 3 から削除されました。
sync_relay_log_info	はい	
sysdate-is-now	はい	
table_cache_element_entry_ttl	いいえ	
table_definition_cache	はい	デフォルト値は式により表されます。式内で DBInstanceClassMemory 値がどのように計算されるかについては、「DB パラメータ式の変数」を参照してください。
table_open_cache	はい	デフォルト値は式により表されます。式内で DBInstanceClassMemory 値がどのように計算されるかについては、「DB パラメータ式の変数」を参照してください。
table_open_cache_instances	はい	
temp-pool	はい	Aurora MySQL バージョン 3 から削除されました。

パラメータ名	変更可能	コメント
temptable_max_mmap	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。詳細については、「Aurora MySQL バージョン 3 での新しい一時テーブルの動作」を参照してください。
temptable_max_ram	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。詳細については、「Aurora MySQL バージョン 3 での新しい一時テーブルの動作」を参照してください。
temptable_use_mmap	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。詳細については、「Aurora MySQL バージョン 3 での新しい一時テーブルの動作」を参照してください。
thread_cache_size	はい	キャッシュされるスレッドの数。このパラメータは、Aurora MySQL バージョン 2 および 3 に適用されます。
thread_handling	いいえ	
thread_stack	はい	
timed_mutexes	はい	

パラメータ名	変更可能	コメント
tmp_table_size	はい	Aurora MySQL バージョン 3 の MEMORY ストレージエンジンによって作成される内部メモリ内一時テーブルの最大サイズを定義します。 Aurora MySQL バージョン 3.04 以降で、aurora_tmptable_enable_per_table_limit が ON のときに TempTable ストレージエンジンによって作成される内部メモリ内一時テーブルの最大サイズを定義します。 詳細については、「内部メモリ内一時テーブルのサイズを制限する」を参照してください。
tmpdir	いいえ	Aurora MySQL は、直接ファイルシステムにアクセスしないマネージドインスタンスを使用します。
transaction_alloc_block_size	はい	
transaction_isolation	はい	このパラメータは、Aurora MySQL バージョン 3 に適用されます。tx_isolation はこれに置き換えられます。
transaction_prealloc_size	はい	
tx_isolation	はい	Aurora MySQL バージョン 3 から削除されました。これは transaction_isolation に置き換えられます。
updatable_views_with_limit	はい	

パラメータ名	変更可能	コメント
validate-password	いいえ	
validate_password_dictionary_file	いいえ	
validate_password_length	いいえ	
validate_password_mixed_case_count	いいえ	
validate_password_number_count	いいえ	
validate_password_policy	いいえ	
validate_password_special_char_count	いいえ	
wait_timeout	はい	Aurora は interactive_timeout と wait_timeout の最小値を評価します。次に、その最小値をタイムアウトとして使い、対話型と非対話型の両方のアイドル状態のセッションをすべて終了します。

Aurora MySQL に適用されない MySQL パラメータ

Aurora MySQL と MySQL ではアーキテクチャに違いがあるため、一部の MySQL パラメータは Aurora MySQL に適用されません。

次の MySQL パラメータは Aurora MySQL には適用されません。これはすべてを網羅したリストではありません。

- activate_all_roles_on_login – このパラメータは、Aurora MySQL バージョン 2 には適用されません。Aurora MySQL バージョン 3 で利用可能です。
- big_tables

- bind_address
- character_sets_dir
- innodb_adaptive_flushing
- innodb_adaptive_flushing_lwm
- innodb_buffer_pool_chunk_size
- innodb_buffer_pool_instances
- innodb_change_buffering
- innodb_checksum_algorithm
- innodb_data_file_path
- innodb_dedicated_server
- innodb_doublewrite
- innodb_flush_log_at_timeout – このパラメータは Aurora MySQL には適用されません。詳細については、「ログバッファをフラッシュする頻度の設定」を参照してください。
- innodb_flush_method
- innodb_flush_neighbors
- innodb_io_capacity
- innodb_io_capacity_max
- innodb_log_buffer_size
- innodb_log_file_size
- innodb_log_files_in_group
- innodb_log_spin_cpu_abs_lwm
- innodb_log_spin_cpu_pct_hwm
- innodb_log_writer_threads
- innodb_max_dirty_pages_pct
- innodb_numa_interleave
- innodb_page_size
- innodb_redo_log_capacity
- innodb_redo_log_encrypt
- innodb_undo_log_encrypt
- innodb_undo_log_truncate

- innodb_undo_logs
- innodb_undo_tablespaces
- innodb_use_native_aio
- innodb_write_io_threads

Aurora MySQL グローバルステータス変数

Aurora MySQL グローバルステータス変数の現在の値は、次のようなステートメントを使用して確認できます。

```
show global status like '%aurora%';
```

次の表では、Aurora MySQL が使用するグローバルステータス変数について説明します。

名前	説明
AuroraDb_commits	前回の再起動以降のコミットの総数。
AuroraDb_commit_latency	前回の再起動以降のコミットレイテンシーの合計。
AuroraDb_ddl_stmt_duration	前回の再起動以降の DDL レイテンシーの合計。
AuroraDb_select_stmt_duration	前回の再起動以降の SELECT ステートメントレイテンシーの合計。
AuroraDb_insert_stmt_duration	前回の再起動以降の INSERT ステートメントレイテンシーの合計。
AuroraDb_update_stmt_duration	前回の再起動以降の UPDATE ステートメントレイテンシーの合計。
AuroraDb_delete_stmt_duration	前回の再起動以降の DELETE ステートメントレイテンシーの合計。
Aurora_binlog_io_cache_allocated	バイナリログ I/O キャッシュに割り当てられたバイト数。

名前	説明
Aurora_binlog_io_cache_read_requests	バイナリログ I/O キャッシュに対して行われた読み取りリクエスト数。
Aurora_binlog_io_cache_reads	バイナリログ I/O キャッシュから処理された読み込みリクエスト数。
Aurora_enhanced_binlog	この DB インスタンスの拡張バイナリログが有効か無効かを示します。詳細については、「拡張バイナリログ記録の設定」を参照してください。
Aurora_external_connection_count	DB インスタンスへのデータベース接続の数。ただし、データベースヘルスチェックに使用される RDS サービス接続は除きます。
Aurora_fast_insert_cache_hits	キャッシュされたカーソルが正常に取得され検証されたときに増加するカウンター。高速挿入キャッシュの詳細については、「Amazon Aurora MySQL パフォーマンスの拡張」を参照してください。
Aurora_fast_insert_cache_misses	キャッシュされたカーソルが有効でなくなり、Aurora が通常のインデックストラバーサルを実行したときに増加するカウンター。高速挿入キャッシュの詳細については、「Amazon Aurora MySQL パフォーマンスの拡張」を参照してください。
Aurora_fwd_master_dml_stmt_count	このライター DB インスタンスに転送された DML ステートメントの合計数。この変数は、Aurora MySQL バージョン 2 に適用されます。
Aurora_fwd_master_dml_stmt_duration	このライター DB インスタンスに転送された DML ステートメントの合計期間。この変数は、Aurora MySQL バージョン 2 に適用されます。

名前	説明
Aurora_fwd_master_errors_rpc_timeout	転送された接続がライター上で確立されなかった回数。
Aurora_fwd_master_errors_session_limit	ライターで session full の理由で拒否された転送されたクエリの数。
Aurora_fwd_master_errors_session_timeout	ライターでのタイムアウトにより転送セッションが終了した回数。
Aurora_fwd_master_open_sessions	ライター DB インスタンスで転送されたセッションの数。この変数は、Aurora MySQL バージョン 2 に適用されます。
Aurora_fwd_master_select_stmt_count	このライター DB インスタンスに転送された SELECT ステートメントの総数。この変数は、Aurora MySQL バージョン 2 に適用されます。
Aurora_fwd_master_select_stmt_duration	このライター DB インスタンスに転送された SELECT ステートメントの合計期間。この変数は、Aurora MySQL バージョン 2 に適用されます。
Aurora_fwd_writer_dml_stmt_count	このライター DB インスタンスに転送された DML ステートメントの合計数。この変数は、Aurora MySQL バージョン 3 に適用されます。
Aurora_fwd_writer_dml_stmt_duration	このライター DB インスタンスに転送された DML ステートメントの合計期間。この変数は、Aurora MySQL バージョン 3 に適用されます。
Aurora_fwd_writer_errors_rpc_timeout	転送された接続がライター上で確立されなかった回数。
Aurora_fwd_writer_errors_session_limit	ライターで session full の理由で拒否された転送されたクエリの数。

名前	説明
Aurora_fwd_writer_errors_session_timeout	ライターでのタイムアウトにより転送セッションが終了した回数。
Aurora_fwd_writer_open_sessions	ライター DB インスタンスで転送されたセッションの数。この変数は、Aurora MySQL バージョン 3 に適用されます。
Aurora_fwd_writer_select_stmt_count	このライター DB インスタンスに転送された SELECT ステートメントの総数。この変数は、Aurora MySQL バージョン 3 に適用されます。
Aurora_fwd_writer_select_stmt_duration	このライター DB インスタンスに転送された SELECT ステートメントの合計期間。この変数は、Aurora MySQL バージョン 3 に適用されます。
Aurora_lockmgr_buffer_pool_memory_used	Aurora MySQL ロックマネージャーが使用しているバッファプールメモリの量 (バイト単位)。
Aurora_lockmgr_memory_used	Aurora MySQL ロックマネージャーが使用しているメモリの量 (バイト単位)。
Aurora_ml_actual_request_cnt	DB インスタンスのユーザーによって実行されたすべてのクエリで、Aurora MySQL が Aurora 機械学習サービスに対して行ったリクエストの集計カウント。詳細については、「Aurora MySQL で Amazon Aurora 機械学習を使用する」を参照してください。
Aurora_ml_actual_response_cnt	DB インスタンスのユーザーが実行するすべてのクエリで、Aurora MySQL が Aurora 機械学習サービスから受け取るレスポンスの集計カウント。詳細については、「Aurora MySQL で Amazon Aurora 機械学習を使用する」を参照してください。

名前	説明
Aurora_ml_cache_hit_cnt	DB インスタンスのユーザーが実行したすべてのクエリで、Aurora MySQL が Aurora 機械学習サービスから受け取る内部キャッシュヒットの集計カウント。詳細については、「Aurora MySQL で Amazon Aurora 機械学習を使用する」を参照してください。
Aurora_ml_logical_request_cnt	前回のステータスリセット以降、DB インスタンスが Aurora 機械学習サービスに送信されると評価した論理リクエストの数。バッチ処理が使用されているかどうかによっては、この値が Aurora_ml_actual_request_cnt より高くなることがあります。詳細については、「Aurora MySQL で Amazon Aurora 機械学習を使用する」を参照してください。
Aurora_ml_logical_response_cnt	DB インスタンスのユーザーが実行するすべてのクエリで、Aurora MySQL が Aurora 機械学習サービスから受け取るレスポンスの集計カウント。詳細については、「Aurora MySQL で Amazon Aurora 機械学習を使用する」を参照してください。
Aurora_ml_retry_request_cnt	前回のステータスリセット以降、DB インスタンスが Aurora 機械学習サービスに送信されると評価した再試行リクエストの数。詳細については、「Aurora MySQL で Amazon Aurora 機械学習を使用する」を参照してください。
Aurora_ml_single_request_cnt	DB インスタンスのユーザーが実行するすべてのクエリで、非バッチモードで評価される Aurora 機械学習の関数の集計カウント。詳細については、「Aurora MySQL で Amazon Aurora 機械学習を使用する」を参照してください。

名前	説明
Aurora_pq_bytes_returned	パラレルクエリ中にヘッドノードに送信されたタプルデータ構造のバイト数。Aurora_pq_pages_pushed_down と比較するために 16,384 で割ります。
Aurora_pq_max_concurrent_requests	この Aurora DB インスタンスで同時に実行できるパラレルクエリセッションの最大数。これは、AWS の DB インスタンスクラスによって異なる固定の数です。
Aurora_pq_pages_pushed_down	パラレルクエリがヘッドノードへのネットワーク送信を回避したデータページ数 (それぞれ 16 KiB の固定サイズ)。
Aurora_pq_request_attempted	リクエストされたパラレルクエリセッションの数。この値は、サブクエリや結合などの SQL 構成に応じて、クエリごとに複数のセッションを表す場合があります。
Aurora_pq_request_executed	パラレルクエリセッションの数は正常に実行されます。
Aurora_pq_request_failed	クライアントにエラーを戻したパラレルクエリセッションの数。場合によっては、例えば、ストレージレイヤーの問題のために、パラレルクエリのリクエストが失敗することがあります。このような場合、失敗したクエリ部分は、非パラレルクエリメカニズムを使用して再試行されます。再試行されたクエリも失敗すると、エラーがクライアントに返され、このカウンターが増分されます。

名前	説明
Aurora_pq_request_in_progress	現在進行中のパラレルクエリセッションの数。この数は、Aurora DB クラスター全体ではなく、接続している特定の Aurora DB インスタンスのものが適用されます。DB インスタンスが同時実行の制限に近いかどうかを調べるには、この値を Aurora_pq_max_concurrent_requests と比較します。
Aurora_pq_request_not_chosen	クエリを満たすためにパラレルクエリが選択されなかった回数。この値は、他のいくつかのより細かいカウンターの合計です。EXPLAIN ステートメントでは、クエリが実際に実行されていない場合でもこのカウンターは増加します。
Aurora_pq_request_not_chosen_below_min_rows	テーブル内の行数のためにパラレルクエリが選択されなかった回数。EXPLAIN ステートメントでは、クエリが実際に実行されていない場合でもこのカウンターは増加します。
Aurora_pq_request_not_chosen_column_bit	射影された列の中にサポートされていないデータ型があるために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。
Aurora_pq_request_not_chosen_column_geometry	テーブルに GEOMETRY データ型の列があるために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。この制限を解除する Aurora MySQL のバージョンについては、Aurora MySQL バージョン 3 への パラレルクエリクラスターのアップグレード を参照してください。

名前	説明
Aurora_pq_request_not_chosen_column_lob	LOB データタイプ、または宣言された長さのため外部に保存された VARCHAR カラムをテーブルが持っていることが原因で、非パラレルクエリの処理パスを使用したパラレルクエリのリクエスト数。この制限を解除する Aurora MySQL のバージョンについては、Aurora MySQL バージョン 3 への パラレルクエリクラスターのアップグレード を参照してください。
Aurora_pq_request_not_chosen_column_virtual	テーブルに仮想列があるために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。
Aurora_pq_request_not_chosen_custom_charset	テーブルにカスタム文字セットの列があるために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。
Aurora_pq_request_not_chosen_fast_ddl	テーブルが高速 DDL の ALTER ステートメントによって変更中であるために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。
Aurora_pq_request_not_chosen_ few_pages_outside_buffer_pool	パラレルクエリを価値のあるものにするためのバッファされていないテーブルデータが十分ないため、テーブルデータの 95 パーセント未満がバッファプールにあったにもかかわらず、パラレルクエリの回数は選択されませんでした。
Aurora_pq_request_not_chosen_full_text_index	テーブルに全文インデックスがあるために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。

名前	説明
Aurora_pq_request_not_chosen_high_buffer_pool_pct	テーブルデータの高パーセンテージ (現在は 95 パーセント以上) が既にバッファプールに入っていたため、パラレルクエリが選択されなかった回数。このような場合、オプティマイザは、バッファプールからのデータの読取りがより効率的であると判断します。EXPLAIN ステートメントでは、クエリが実際に実行されていない場合でもこのカウンターは増加します。
Aurora_pq_request_not_chosen_index_hint	クエリにインデックスヒントが含まれているために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。
Aurora_pq_request_not_chosen_innodb_table_format	テーブルが、サポートされていない InnoDB の行形式を使用しているために、パラレルクエリ以外の処理方法が適用されるパラレルクエリのリクエスト数。Aurora のパラレルクエリは、COMPACT、REDUNDANT ,および DYNAMIC の行形式にのみ適用されます。
Aurora_pq_request_not_chosen_long_trx	長時間実行トランザクション内でクエリがスタートされているために、非パラレルクエリ処理パスを使用したパラレルクエリリクエストの数。EXPLAIN ステートメントでは、クエリが実際に実行されていない場合でもこのカウンターは増加します。
Aurora_pq_request_not_chosen_no_where_clause	クエリに WHERE 句がないために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。
Aurora_pq_request_not_chosen_range_scan	インデックスの範囲スキャンを使用しているために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。

名前	説明
Aurora_pq_request_not_chosen_row_length_too_long	すべての列の合計長が長すぎるために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。
Aurora_pq_request_not_chosen_small_table	行数および平均行長によって決定される、テーブルの全体的なサイズのためにパラレルクエリが選択されなかった回数。EXPLAIN ステートメントでは、クエリが実際に実行されていない場合でもこのカウンターは増加します。
Aurora_pq_request_not_chosen_temporary_table	クエリでサポートされていない MyISAM または memory テーブルタイプを使用しているテンポラリテーブルを参照しているために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。
Aurora_pq_request_not_chosen_tx_isolation	クエリでサポートされていないトランザクション分離レベルを使用しているために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。リーダー DB インスタンスでは、パラレルクエリは REPEATABLE READ および READ COMMITTED 分離レベルにのみ適用されます。
Aurora_pq_request_not_chosen_update_delete_stmts	クエリが UPDATE または DELETE ステートメントの一部であるために、パラレルクエリ以外の処理方法が使用されるパラレルクエリのリクエスト数。
Aurora_pq_request_not_chosen_unsupported_access	WHERE 句がパラレルクエリの基準を満たしていないために、非パラレルクエリ処理パスを使用するパラレルクエリリクエストの数。この結果は、クエリがデータ集約型スキャンを必要としない場合、またはクエリが DELETE または UPDATE ステートメントである場合に発生します。

名前	説明
Aurora_pq_request_not_chosen_unsupported_storage_type	Aurora MySQL DB クラスターがサポートされている Aurora クラスターストレージ設定を使用していないために非並列クエリ処理パスを使用する並列クエリリクエストの数。詳細については、「制限事項」を参照してください。 このパラメータは、Aurora MySQL バージョン 3.04 以降に適用されます。
Aurora_pq_request_throttled	特定の Aurora DB インスタンスで既に実行されている同時パラレルクエリの最大数のために、パラレルクエリが選択されなかった回数。
Aurora_repl_bytes_received	前回再起動してから Aurora MySQL リーダーデータベースインスタンスに複製されたバイト数。詳細については、「Amazon Aurora MySQL でのレプリケーション」を参照してください。
Aurora_reserved_mem_exceeded_incidents	前回の再起動以降、エンジンが予約メモリの制限を超えた回数。aurora_oom_response が設定されている場合、このしきい値は、メモリ不足 (OOM) 回避アクティビティがトリガーされるタイミングを定義します。Aurora MySQL OOM レスポンスの詳細については、「Aurora MySQL データベースのメモリ不足の問題のトラブルシューティング」を参照してください。
Aurora_thread_pool_thread_count	Aurora スレッドプール内の現在のスレッド数。Aurora MySQL OOM レスポンスの詳細については、「スレッドプール」を参照してください。

名前	説明
`Aurora_tmz_version`	DB クラスターで使用されているタイムゾーン情報の現在のバージョンを示します。値は IANA (Internet Assigned Numbers Authority) 形式 `YYYYsuffix` に従います。例えば、`2022a` および `2023c` です。 このパラメータは、Aurora MySQL バージョン 2.12 以降とバージョン 3.04 以降に適用されます。
`server_aurora_das_running`	この DB インスタンスでデータベースアクティビティストリーム (DAS) が有効か無効かを示します。詳細については、「データベースアクティビティストリームを使用した Amazon Aurora のモニタリング」を参照してください。

Aurora MySQL に適応されない MySQL ステータス可変

Aurora MySQL と MySQL ではアーキテクチャに違いがあるため、一部の MySQL パラメータのステータス可変は Aurora MySQL に適用されません。

以下の MySQL ステータス可変は Aurora MySQL には適用されません。これはすべてを網羅したリストではありません。

- `innodb_buffer_pool_bytes_dirty`
- `innodb_buffer_pool_pages_dirty`
- `innodb_buffer_pool_pages_flushed`

Aurora MySQL バージョン 3 は、Aurora MySQL バージョン 2 にあった次のステータス可変を削除します。

- `AuroraDb_lockmgr_bitmaps0_in_use`
- `AuroraDb_lockmgr_bitmaps1_in_use`
- `AuroraDb_lockmgr_bitmaps_mem_used`

- AuroraDb_thread_deadlocks
- available_alter_table_log_entries
- Aurora_lockmgr_memory_used
- Aurora_missing_history_on_replica_incidents
- Aurora_new_lock_manager_lock_release_cnt
- Aurora_new_lock_manager_lock_release_total_duration_micro
- Aurora_new_lock_manager_lock_timeout_cnt
- Aurora_total_op_memory
- Aurora_total_op_temp_space
- Aurora_used_alter_table_log_entries
- Aurora_using_new_lock_manager
- Aurora_volume_bytes_allocated
- Aurora_volume_bytes_left_extent
- Aurora_volume_bytes_left_total
- Com_alter_db_upgrade
- Compression
- External_threads_connected
- Innodb_available_undo_logs
- Last_query_cost
- Last_query_partial_plans
- Slave_heartbeat_period
- Slave_last_heartbeat
- Slave_received_heartbeats
- Slave_retried_transactions
- Slave_running
- Time_since_zero_connections

これらの MySQL ステータス変数は、Aurora MySQL バージョン 2 で使用できますが、Aurora MySQL バージョン 3 では使用できません。

- Innodb_redo_log_enabled

- Innodb_undo_tablespaces_total
- Innodb_undo_tablespaces_implicit
- Innodb_undo_tablespaces_explicit
- Innodb_undo_tablespaces_active

Aurora MySQL の待機イベント

以下は、Aurora MySQL の代表的な待機イベントです。

> **Note**
>
> 待機イベントを使用して Aurora MySQL のパフォーマンスを調整する方法については、「待機イベントを使用した Aurora MySQL のチューニング」を参照してください。
> MySQL の待機イベントで使用される命名規則については、MySQL ドキュメントの「Performance Schema インストゥルメント命名規則」を参照してください。

cpu

実行可能なアクティブな接続の数が vCPUs 数よりも一貫して多くなります。詳細については、「cpu」を参照してください。

io/aurora_redo_log_flush

セッションは Aurora ストレージにデータを保持しています。通常、この待機イベントは Aurora MySQL の書き込み I/O オペレーション用です。詳細については、「io/aurora_redo_log_flush」を参照してください。

io/aurora_respond_to_client

Aurora MySQL バージョン 2.10.2 以降の 2.10 バージョン、2.09.3 以降の 2.09 バージョン、2.07.7 以降の 2.07 バージョンの場合、クエリ処理が完了すると、結果がアプリケーションクライアントに返されます。DB インスタンスクラスのネットワーク帯域幅と、返される結果セットのサイズを比較します。また、クライアント側の応答時間もチェックしてください。クライアントが応答せず、TCP パケットを処理できない場合、パケットドロップと TCP 再送信が発生する可能性があります。この状況は、ネットワーク帯域幅に悪影響を及ぼします。2.10.2、2.09.3、2.09.3、および 2.07.7 より前のバージョンでは、待機イベントにアイドル時間が誤って含まれます。この待機が目立つときにデータベースをチューニングする方法については、io/aurora_respond_to_client を参照してください。

io/file/csv/data

スレッドは Comma Separated Value (CSV) 形式でテーブルに書き込みます。CSV テーブルの使用状況を確認してください。通常、このイベントはテーブルでの `log_output` の設定に伴って発生します。

io/file/sql/binlog

スレッドは、ディスクに書き込まれるバイナリログ (binlog) ファイルを待っています。

io/redo_log_flush

セッションは Aurora ストレージにデータを保持しています。通常、この待機イベントは Aurora MySQL の書き込み I/O オペレーション用です。詳細については、「io/redo_log_flush」を参照してください。

io/socket/sql/client_connection

`mysqld` プログラムは受信する新規クライアント接続を処理するためのスレッド作成でビジー状態です。詳細については、「io/socket/sql/client_connection」を参照してください。

io/table/sql/handler

エンジンは、テーブルへのアクセスを待っています。このイベントは、データがバッファプールにキャッシュされているか、ディスク上でアクセスされているかにかかわらず、発生します。詳細については、「io/table/sql/handler」を参照してください。

lock/table/sql/handler

この待機イベントは、テーブルロック待機イベントハンドラです。Performance Schema の原始的イベントと分子的イベントの詳細については、MySQL ドキュメントの Performance Schema の原子的および分子的イベントを参照してください。

synch/cond/innodb/row_lock_wait

複数のデータ操作言語 (DML) ステートメントが同じデータベース行に同時にアクセスしようとしています。詳細については、「synch/cond/innodb/row_lock_wait」を参照してください。

synch/cond/innodb/row_lock_wait_cond

複数の DML ステートメントが同じデータベース行に同時にアクセスしようとしています。詳細については、「synch/cond/innodb/row_lock_wait_cond」を参照してください。

synch/cond/sql/MDL_context::COND_wait_status

スレッドは、テーブルのメタデータロックを待っています。エンジンは、このタイプのロックを使用して、データベーススキーマへの同時アクセスを管理し、データの整合性を確保しま

す。詳細については、MySQL ドキュメントの「ロック操作の最適化」を参照してください。この待機が目立つときにデータベースをチューニングする方法については、synch/cond/sql/MDL_context::COND_wait_status を参照してください。

synch/cond/sql/MYSQL_BIN_LOG::COND_done

バイナリログを有効にしている。高いコミットスループット、多数のトランザクションがコミットされている、またはバイナリログを読み取るレプリカがある可能性があります。複数行ステートメントを使用するか、ステートメントを 1 つのトランザクションにバンドルすることを検討してください。Aurora では、バイナリログレプリケーションや `aurora_binlog_*` パラメータの代わりにグローバルデータベースを使用します。

synch/mutex/innodb/aurora_lock_thread_slot_futex

複数の DML ステートメントが同じデータベース行に同時にアクセスしようとしています。詳細については、「synch/mutex/innodb/aurora_lock_thread_slot_futex」を参照してください。

synch/mutex/innodb/buf_pool_mutex

バッファプールが、ワーキング データ セットを保持するのに十分な大きさではありません。または、ワークロードが特定のテーブルからページにアクセスし、バッファプール内で競合が発生します。詳細については、「synch/mutex/innodb/buf_pool_mutex」を参照してください。

synch/mutex/innodb/fil_system_mutex

プロセスは、テーブルスペースのメモリキャッシュへのアクセスを待っています。詳細については、「synch/mutex/innodb/fil_system_mutex」を参照してください。

synch/mutex/innodb/trx_sys_mutex

オペレーションは、一貫した、または制御された方法で InnoDB 内のトランザクション ID をチェック、更新、削除、または追加しています。これらの操作は `trx_sys` mutex 呼び出しを必要とします。これはパフォーマンススキーマインストルメンテーションによって追跡されます。操作には、データベースの起動時またはシャットダウン時のトランザクションシステムの管理、ロールバック、クリーンアップの取り消し、行の読み取りアクセス、およびバッファプールのロードが含まれます。トランザクション数が多い高データベース ロードは、この待機イベントの頻繁な発生につながります。詳細については、「synch/mutex/innodb/trx_sys_mutex」を参照してください。

synch/mutex/mysys/key_cache:: cache_lock

`keycache->cache_lock` mutex は MyISAM テーブルのキーキャッシュへのアクセスを制御します。Aurora MySQL では MyISAM テーブルを使用して永続データを保存することはでき

ませんが、内部一時テーブルの保存に使用されます。状況によっては、一時テーブルがメモリに収まらなくなるとディスクに書き込まれることがあるため、`created_tmp_tables` または `created_tmp_disk_tables` ステータスカウンターを確認することを検討してください。

synch/mutex/sql/file_as_table:: LOCK_offsets

エンジンは、テーブルメタデータファイルを開いたり作成したりするときに、このミューテックスを取得します。この待機イベントが過剰な頻度で発生すると、作成またはオープンされているテーブルの数が急増しています。

synch/mutex/sql/FILE_AS_TABLE::LOCK_shim_lists

エンジンは、`reset_size`、`detach_contents`、または `add_contents` のようなオペレーションを、オープンテーブルを追跡する内部構造上で実行しながら、このミューテックスを取得します。ミューテックスは、リストの内容へのアクセスを同期します。この待機イベントが高頻度で発生すると、以前にアクセスされたテーブルのセットが突然変化したことを示します。エンジンは、新しいテーブルにアクセスするか、以前にアクセスしたテーブルに関連するコンテキストを解放する必要があります。

synch/mutex/sql/LOCK_open

セッションが開いているテーブルの数が、テーブル定義キャッシュまたはテーブルオープンキャッシュのサイズを超えています。これらのキャッシュのサイズを増やします。詳細については、「MySQL でのテーブルのオープンとクローズの方法」を参照してください。

synch/mutex/sql/LOCK_table_cache

セッションが開いているテーブルの数が、テーブル定義キャッシュまたはテーブルオープンキャッシュのサイズを超えています。これらのキャッシュのサイズを増やします。詳細については、「MySQL でのテーブルのオープンとクローズの方法」を参照してください。

synch/mutex/sql/LOG

この待機イベントの場合、ログロックの待機中のスレッドがあります。例えば、スレッドは、スロークエリログファイルに書き込むためにロックを待っている場合があります。

synch/mutex/sql/MYSQL_BIN_LOG::LOCK_commit

この待機イベントの場合、バイナリログにコミットする目的でロック取得を待機中のスレッドがあります。変化率が非常に高いデータベースではバイナリログ記録の競合が発生する場合がありえます。使用している MySQL のバージョンによっては、バイナリログの整合性と耐久性の高さを保護するために使用される特定のロックがあります。RDS for MySQL の場合、バイナリログはレプリケーションと自動バックアッププロセスに使用されます。Aurora MySQL の場合、バ

イナリログはネイティブのレプリケーションやバックアップに必要ありません。バイナリログはデフォルトで無効になっていますが、これを有効にして外部のレプリケーションや変更データのキャプチャに使用できます。詳細については、MySQL ドキュメントの「バイナリログ」を参照してください。

sync/mutex/sql/MYSQL_BIN_LOG:: LOCK_dump_thread_metrics_collection

バイナリログ記録がオンの場合、エンジンはアクティブなダンプスレッドのメトリクスをエンジンエラーログと内部オペレーションマップに出力する際、このミューテックスを取得します。

sync/mutex/sql/MYSQL_BIN_LOG:: LOCK_inactive_binlogs_map

バイナリログ記録がオンの場合、エンジンは最新のバイナリログファイルのリストに追加したり、そこから削除したり、検索したりする際に、このミューテックスを取得します。

sync/mutex/sql/MYSQL_BIN_LOG:: LOCK_io_cache

バイナリログがオンの場合、エンジンは Aurora binlog IO キャッシュ操作(割り当て、サイズ変更、解放、書き込み、読み取り、パージ、およびキャッシュ情報へのアクセス) 中にこのミューテックスを取得します。このイベントが頻繁に発生する場合、エンジンは binlog イベントが格納されているキャッシュにアクセスしています。待機時間を短縮するには、コミットを減らします。複数のステートメントを 1 つのトランザクションにグループ化してみてください。

synch/mutex/sql/MYSQL_BIN_LOG::LOCK_log

バイナリログを有効にしている。高いコミットスループット、多数のコミットしているトランザクション、またはバイナリログを読み取るレプリカがある可能性があります。複数行ステートメントを使用するか、ステートメントを 1 つのトランザクションにバンドルすることを検討してください。Aurora では、バイナリログ レプリケーションや `aurora_binlog_*` パラメータの代わりにグローバルデータベースを使用します。

synch/mutex/sql/server_thread:: LOCK_sync

ミューテックス `SERVER_THREAD::LOCK_sync` はファイル書き込みスレッドのスケジューリング、処理、または起動中に取得されます。この待機イベントが過剰に発生すると、データベース内の書き込みアクティビティが増加していることを示します。

synch/mutex/sql/TABLESPACES:lock

エンジンは、作成、削除、トランケート、および拡張のテーブルスペース オペレーション中に `TABLESPACES:lock` ミューテックスを取得します。この待機イベントが過剰に発生すると、テーブルスペース操作の頻度が高いことを示します。例として、大量のデータをデータベースにロードしています。

synch/rwlock/innodb/dict

この待機イベントの場合、InnoDB データディクショナリに保持されている rwlock を待機中のスレッドがあります。

synch/rwlock/innodb/dict_operation_lock

この待機イベントの場合、InnoDB データディクショナリのオペレーションの口ックを保持しているスレッドがあります。

synch/rwlock/innodb/dict sys RW lock

データ定義言語コード (DDLs) 内の多数の同時データ制御言語ステートメント (DCLs) が同時にトリガーされます。通常のアプリケーションアクティビティ中に、アプリケーションの DDL への依存度を下げます。

synch/rwlock/innodb/index_tree_rw_lock

複数の類似したデータ操作言語 (DML) ステートメントが同じデータベースオブジェクトに同時にアクセスしようとしています。複数行ステートメントを使用してみてください。また異なるデータベースオブジェクトにワークロードを分散します。例えば、パーティショニングを実装します。

synch/sxlock/innodb/dict_operation_lock

データ定義言語コード (DDL) 内の多数の同時データ制御言語ステートメント (DCL) が同時にトリガーされます。通常のアプリケーションアクティビティ中に、アプリケーションの DDLs への依存度を下げます。

synch/sxlock/innodb/dict_sys_lock

データ定義言語コード (DDL) 内の多数の同時データ制御言語ステートメント (DCL) が同時にトリガーされます。通常のアプリケーションアクティビティ中に、アプリケーションの DDLs への依存度を下げます。

synch/sxlock/innodb/hash_table_locks

セッションは、バッファプール内のページを見つけることができませんでした。エンジンは、ファイルを読み取るか、バッファプールの最も長い時間使われていない (LRU) リストを変更する必要があります。バッファキャッシュのサイズを増やし、関連するクエリのアクセスパスを改善することを検討してください。

synch/sxlock/innodb/index_tree_rw_lock

複数の類似したデータ操作言語 (DML) ステートメントが同じデータベースオブジェクトに同時にアクセスしようとしています。複数行ステートメントを使用してみてください。また異なるデー

タベースオブジェクトにワークロードを分散します。例えば、パーティショニングを実装します。

同期待機イベントのトラブルシューティングの詳細は、「Performance Insights で SYNCH 待機イベントを待機しているアクティブセッションの数が多いことが MySQL DB インスタンスに表示されているのはなぜですか?」を参照してください。

Aurora MySQL スレッド状態

以下は、Aurora MySQL の代表的な待機イベントです。

許可の確認

スレッドは、サーバーがステートメントを実行するために必要な権限を持っているかどうかをチェックしています。

クエリキャッシュのクエリのチェック

サーバーは、現在のクエリがクエリキャッシュに存在するかどうかをチェックしています。

クリーンアップ

これは、作業は完了しているがクライアントによってまだ閉じられていない接続の最終状態です。最善の解決策は、コード内の接続を明示的に閉じることです。または、パラメータグループの `wait_timeout` に、より低い値を設定することもできます。

テーブルを閉じる

スレッドは、変更されたテーブルデータをディスクにフラッシュし、使用されているテーブルを閉じています。高速操作ではない場合は、インスタンスクラスのネットワーク帯域幅に対するネットワーク帯域幅消費メトリックを確認します。また `table_open_cache` と `table_definition_cache` パラメータのパラメータ値が、十分なテーブルを同時に開ける状態かをチェックし、エンジンが頻繁にテーブルを開いたり閉じたりする必要がないようにします。これらのパラメータは、インスタンスのメモリ消費量に影響します。

HEAP を MyISAM に変換する

クエリは、一時テーブルをインメモリからディスク上に変換しています。この変換は、クエリ処理の中間ステップで MySQL によって作成された一時テーブルがメモリに対して大きすぎるため、必要になります。`tmp_table_size` と `max_heap_table_size` の値をチェックします。それ以降のバージョンでは、このスレッド状態名は `converting HEAP to ondisk` です。

HEAP をオンディスクに変換する

スレッドは、内部一時テーブルをインメモリテーブルからディスク上のテーブルに変換しています。

tmp テーブルにコピーする

スレッドが ALTER TABLE ステートメントを処理中です。この状態は、新しい構造を持つテーブルが作成された後、ローがそのテーブルにコピーされる前に発生します。この状態のスレッドの場合、パフォーマンススキーマを使用して、コピー操作の進行状況に関する情報を取得できます。

ソートインデックスの作成

Aurora MySQL は、既存のインデックスを使用して クエリの ORDER BY および GROUP BY 句を満たすことができないため、ソートを実行しています。詳細については、「ソートインデックスの作成」を参照してください。

テーブルの作成

スレッドは、永続テーブルまたは一時テーブルを作成しています。

遅延コミット ok 完了

Aurora MySQL の非同期コミットが承認を受け取り、完了しました。

遅延コミット ok スタートしました

Aurora MySQL スレッドは非同期コミットプロセスをスタートしましたが、確認を待っています。これは通常、トランザクションの本物のコミット時間です。

遅延送信 ok 完了

接続に関連付けられている Aurora MySQL ワーカースレッドは、応答がクライアントに送信されている間に解放できます。スレッドは他の作業をスタートできます。ステータス delayed send okは クライアントへの非同期確認が完了したことを示します。

遅延送信 「ok」 をスタートしました

Aurora MySQL ワーカースレッドがクライアントに非同期で応答を送信し、他の接続で自由に作業できるようになりました。トランザクションは、まだ確認されていない非同期コミットプロセスをスタートしました。

実行中

スレッドはステートメントの実行をスタートしました。

アイテムを解放する

スレッドがコマンドを実行しました。この状態中に行われた項目の解放には、クエリキャッシュが含まれます。この状態は通常、クリーンアップが続きます。

初期化

この状態は、ALTER TABLE、DELETE、INSERT、SELECT、または UPDATE ステートメントの初期化前に発生します。この状態のアクションには、バイナリログまたは InnoDB ログのフラッシュ、クエリキャッシュのクリーンアップが含まれます。

マスターがすべてのバイナリログをスレーブに送信しました

プライマリノードはレプリケーションの一部を終了しました。スレッドは、バイナリログ (binlog) に書き込むことができるように、より多くのクエリが実行されるのを待っています。

テーブルを開く

スレッドがテーブルを開こうとしている。ALTER TABLE または LOCK TABLE ステートメントを終了する必要がある場合や、 table_open_cache の値を超える場合を除いて、このオペレーションは早いです。

最適化

サーバーはクエリの初期最適化を実行しています。

準備

この状態は、クエリの最適化中に発生します。

クエリ終了

この状態は、クエリの処理後、アイテム解放状態の前に発生します。

重複を除去する

Aurora MySQL は クエリの初期段階でDISTINCT オペレーションを最適化できませんでした。Aurora MySQL は、結果をクライアントに送信する前に、重複した行をすべて削除する必要があります。

更新の行の検索

スレッドは更新する前に一致する行をすべて検索しています。この段階は、エンジンが行の検索に使用するインデックスを UPDATE が変更している時に必要です。

バイナリログイベントをスレーブに送信する

スレッドはバイナリログからイベントを読み取り、それをレプリカに送信しています。

キャッシュされた結果をクライアントに送信する

サーバーは、クエリキャッシュからクエリの結果を取得し、それをクライアントに送信しています。

データの送信

スレッドは、SELECT ステートメントの行を読み取り、処理していますが、クライアントへのデータの送信はまだスタートされていません。このプロセスでは、クエリを満たすために必要な結果が含まれているページを特定します。詳細については、「データの送信」を参照してください。

クライアントに送信する

サーバーはクライアントにパケットを書き込んでいます。以前のバージョンの MySQL では、この待機イベントは `writing to net` としてラベル付けされていました。

スタート中

これは、ステートメントの実行スタートの初期の段階です。

統計

サーバーは統計を計算して、クエリ実行プランを開発しています。スレッドが長時間この状態にある場合、サーバーは他の作業の実行中にディスクバインドされている可能性があります。

クエリキャッシュに結果を格納する

サーバーは、クエリの結果をクエリキャッシュに格納しています。

システムロック

スレッドは `mysql_lock_tables` を呼び出しましたが、呼び出し以降、スレッドの状態は更新されていません。この一般的な状態は多くの理由で発生します。

更新

スレッドはテーブルの更新をスタートする準備をしています。

更新中

スレッドは行を検索し、更新中です。

ユーザーロック

スレッドは GET_LOCK を発行しました。スレッドがアドバイザリロックを要求して待っているか、リクエストを計画しています。

さらに更新を待っている

プライマリノードはレプリケーションの一部を終了しました。スレッドは、バイナリログ (binlog) に書き込むことができるように、より多くのクエリが実行されるのを待っています。

スキーマメタデータのロックを待っています

これは、メタデータのロックを待ちます。

ストアド関数のメタデータのロックを待っています

これは、メタデータのロックを待ちます。

ストアドプロシージャのメタデータのロックを待っています。

これは、メタデータのロックを待ちます。

テーブルフラッシュを待っています。

スレッドが実行中ですFLUSH TABLESそして、すべてのスレッドがテーブルを閉じるのを待っています。または、スレッドは、テーブルの基になる構造が変更されたという通知を受信したため、新しい構造を取得するにはテーブルを再度開く必要があります。テーブルを再度開くには、スレッドは他のすべてのスレッドがテーブルを閉じるまで待機する必要があります。この通知は、別のスレッドがテーブルで次のいずれかのステートメントを使用している場合に発生します。FLUSH TABLES、ALTER TABLE、RENAME TABLE、REPAIR TABLE、ANALYZE TABLE、またはOPTIMIZE TABLE。

テーブルレベルのロックを待っています。

あるセッションがテーブルのロックを保持し、別のセッションが同じテーブルで同じロックを取得しようとします。

テーブルメタデータのロックを待っています。

Aurora MySQL は、メタデータロックを使用して、データベースオブジェクトへの同時アクセスを管理し、データの整合性を確保します。この待機イベントでは、あるセッションがテーブルのメタデータロックを保持し、別のセッションが同じテーブルで同じロックを取得しようとします。パフォーマンススキーマが有効になっている場合、このスレッドの状態は待機イベント synch/cond/sql/MDL_context::COND_wait_status として報告されます。

ネットへの書き込み

サーバーはネットワークにパケットを書き込んでいます。それ以降のバージョンの MySQL では、この待機イベントには Sending to client というラベルが付けられます。

Aurora MySQL の分離レベル

Aurora MySQL クラスターの DB インスタンスが分離のデータベースプロパティを実装する方法について説明します。このトピックでは、Aurora MySQL のデフォルトの動作で厳密な一貫性と高いパフォーマンスのバランスがどのように取られるかを説明します。この情報を使用して、ワークロードの特性に基づいて、デフォルト設定を変更するタイミングを判断することができます。

ライターインスタンスで使用可能な分離レベル

Aurora MySQL DB クラスターのプライマリインスタンスでは、分離レベル `REPEATABLE READ`、`READ COMMITTED`、`READ UNCOMMITTED`、および `SERIALIZABLE` を使用できます。これらの分離レベルは、Aurora MySQL と RDS for MySQL で同じように機能します。

リーダーインスタンスでの REPEATABLE READ の分離レベル

デフォルトでは、読み取り専用の Aurora レプリカとして設定された Aurora MySQL DB インスタンスは、常に `REPEATABLE READ` 分離レベルを使用します。これらの DB インスタンスは、`SET TRANSACTION ISOLATION LEVEL` ステートメントを無視し、引き続き `REPEATABLE READ` 分離レベルを使用します。

DB パラメータまたは DB クラスターパラメータを使用して Reader DB インスタンスの分離レベルを設定することはできません。

リーダーインスタンスでの READ COMMITTED の分離レベル

アプリケーションのプライマリインスタンスに書き込み集中型のワークロードが含まれ、Aurora レプリカに長時間実行されるクエリが含まれている場合、かなりのパージラグが発生する可能性があります。パージラグは、長時間実行されるクエリによって内部ガベージコレクションがブロックされた場合に発生します。確認できる症状は、`SHOW ENGINE INNODB STATUS` コマンドからの出力で `history list length` の値が高いことです。この値をモニタリングするには、CloudWatch の `RollbackSegmentHistoryListLength` メトリクスを使用します。大幅なパージラグが発生すると、セカンダリインデックスの有効性が低下し、全体的なクエリパフォーマンスの低下とストレージ領域の浪費が起きる可能性があります。

このような問題が発生した場合は、Aurora レプリカで `READ COMMITTED` 分離レベルを使用するように、Aurora MySQL セッションレベルの構成設定 `aurora_read_replica_read_committed` を設定します。この設定を適用すると、テーブルを変更するトランザクションと同時に長時間実行されるクエリを実行した場合に発生する可能性のある、速度低下と領域の浪費を軽減できます。

この設定を使用するには、Aurora MySQL の READ COMMITTED 分離に固有の動作を理解しておくことをお勧めします。Aurora レプリカでの READ COMMITTED の動作は、ANSI SQL スタンダードに準拠しています。ただし、分離は一般的な MySQL の READ COMMITTED の存知の動作ほど厳密ではありません。そのため、Aurora MySQL のリードレプリカでの READ COMMITTED は、Aurora MySQL のプライマリインスタンスや RDS for MySQL での READ COMMITTED の同じクエリとは異なるクエリ結果になる場合があります。非常に大規模なデータベースをスキャンする包括的なレポートなどの場合には、aurora_read_replica_read_committed 設定の使用を検討してください。逆に、精度と再現性が重要な、結果セットが小さいショートクエリでは、回避した方がよい場合があります。

READ COMMITTED 独立性レベルは、書き込み転送機能を使用する Aurora Global Database 内のセカンダリクラスター内のセッションでは使用できません。書き込み転送の詳細については、「Amazon Aurora Global Database の書き込み転送を使用する」を参照してください。

リーダーでの READ COMMITTED の使用

Aurora レプリカで READ COMMITTED 分離レベルを使用するには、aurora_read_replica_read_committed 構成設定を ON に設定します。特定の Aurora レプリカに接続しているときに、セッションレベルでこの設定を使用します。有効にするには、以下の SQL コマンドを実行します。

```
set session aurora_read_replica_read_committed = ON;
set session transaction isolation level read committed;
```

この構成設定を一時的に有効にして、インタラクティブなアドホック (1 回限りの) クエリを実行することもできます。また、READ COMMITTED 分離レベルのメリットを活かせるレポートまたはデータ分析アプリケーションを実行して、他のアプリケーションについては、デフォルト設定のままにしておくこともできます。

aurora_read_replica_read_committed 設定が有効になっているときに、SET TRANSACTION ISOLATION LEVEL コマンドを使用して、該当するトランザクションの分離レベルを指定します。

```
set transaction isolation level read committed;
```

Aurora レプリカでの READ COMMITTED の動作の違い

aurora_read_replica_read_committed 設定は、Aurora レプリカで READ COMMITTED 分離レベルを使用可能にし、長時間実行されるトランザクション用に最適化された一貫性動作を実現します。Aurora レプリカでの READ COMMITTED 分離レベルの厳密性は、Aurora プライマリインスタン

スでの厳密性より劣ります。そのため、クエリが特定のタイプの一貫性のない結果を受け入れられることがわかっている Aurora レプリカでのみ、この設定を有効にしてください。

aurora_read_replica_read_committed 設定がオンのときに、クエリで特定の種類の読み取り異常が発生する可能性があります。アプリケーションコードについて理解して処理するためには、2 種類の異常が特に重要です。クエリの実行中に別のトランザクションがコミットすると、繰り返し不可のリードが発生します。長時間実行されるクエリでは、クエリのスタート時に、終了時に表示されるデータとは異なるデータが表示されることがあります。ファントムリードは、クエリの実行中に他のトランザクションによって既存の行が再編成され、1 つ以上の行がクエリによって 2 回読み取られると発生します。

ファントムリードの結果として、クエリで一貫性のない行カウントが実行される場合があります。繰り返し不可のリードが原因で、クエリから不完全または一貫性のない結果が返されることもあります。例えば、結合操作が SQL ステートメント (INSERT、DELETE など) によって同時に変更されるテーブルを参照するとします。この場合、結合クエリは、あるテーブルの行を読み取りますが、別のテーブルの対応する行は読み取らない可能性があります。

ANSI SQL スタンダードでは、READ COMMITTED 分離レベルでこれらの両方の動作が許可されます。ただしこれらの動作は、READ COMMITTED の一般的な MySQL 実装とは異なります。そのため、aurora_read_replica_read_committed 設定を有効にする前に、既存の SQL コードを調べて、よりあいまいな整合性モデルで期待どおりに動作するかどうかを確認します。

この設定が有効になっている場合、READ COMMITTED 分離レベルでは、行カウントとその他の結果の一貫性があまりない場合があります。したがって通常は、大量のデータを集計し、絶対的な精度を必要としない分析クエリを実行しているときにのみ、設定を有効にします。これらの種類の長時間実行クエリを、書き込み集中型のワークロードと組み合わせて使用しない場合、aurora_read_replica_read_committed 設定は不要である可能性があります。長時間実行されるクエリと書き込み集中型のワークロードの組み合わせがなければ、履歴リストの長さに問題が発生することはほとんどありません。

Example Aurora レプリカでの READ COMMITTED の分離動作を示すクエリ

次の例は、トランザクションが関連するテーブルを同時に変更した場合に、Aurora レプリカで実行された READ COMMITTED クエリが繰り返し不可能な結果を返す方法を示しています。テーブル BIG_TABLE には、クエリスタートの前に 100 万行が含まれています。他のデータ操作言語 (DML) ステートメントは、実行中に行を追加、削除、または変更します。

Aurora プライマリインスタンスにおける READ COMMITTED 分離レベルでのクエリは、予測可能な結果を生成します。ただし、長時間実行されるすべてのクエリのライフタイムにわたって一貫した読

み取りビューを維持するオーバーヘッドにより、後で高コストのガベージコレクションが発生する可能性があります。

Aurora レプリカにおける READ COMMITTED 分離レベルでのクエリは、このガベージコレクションのオーバーヘッドを最小限に抑えるように最適化されます。トレードオフとして、クエリの実行中にコミットされるトランザクションによって追加、削除、または再編成された行をクエリが取得するかどうかによって結果が異なる場合があります。クエリではこれらの行を考慮することができますが、必須ではありません。デモンストレーションの目的で、クエリは COUNT(*) 関数を使用してテーブル内の行数のみをチェックします。

時間	Aurora プライマリインスタンスでの DML ステートメント	Aurora プライマリインスタンスでの READ COMMITTED を使用したクエリ	Aurora レプリカでの READ COMMITTED を使用したクエリ
T1	`INSERT INTO big_table SELECT * FROM other_table LIMIT 1000000; COMMIT;`		
T2		Q1: `SELECT COUNT(*) FROM big_table;`	Q2: `SELECT COUNT(*) FROM big_table;`
T3	`INSERT INTO big_table (c1, c2) VALUES (1, 'one more row'); COMMIT;`		
T4		Q1 が終了すると、結果は 1,000,000 になります。	Q2 が終了すると、結果は 1,000,000 または 1,000,001 になります。

時間	Aurora プライマリインスタンスでの DML ステートメント	Aurora プライマリインスタンスでの READ COMMITTED を使用したクエリ	Aurora レプリカでの READ COMMITTED を使用したクエリ
T5	`DELETE FROM big_table LIMIT 2; COMMIT;`		
T6		Q1 が終了すると、結果は 1,000,000 になります。	Q2 が終了すると、結果は 1,000,000、1,000,001、999,999、999,998 のいずれかになります。
T7	`UPDATE big_table SET c2 = CONCAT(c2,c2,c2); COMMIT;`		
T8		Q1 が終了すると、結果は 1,000,000 になります。	Q2 が終了すると、結果は 1,000,000、1,000,001、999,999、またはそれより大きい値になります。
T9		Q3: `SELECT COUNT(*) FROM big_table;`	Q4: `SELECT COUNT(*) FROM big_table;`
T10		Q3 が終了すると、結果は 999,999 になります。	Q4 が終了すると、結果は 999,999 になります。

時間	Aurora プライマリインスタンスでの DML ステートメント	Aurora プライマリインスタンスでの READ COMMITTED を使用したクエリ	Aurora レプリカでの READ COMMITTED を使用したクエリ
T11		Q5: `SELECT COUNT(*) FROM parent_table p JOIN child_table c ON (p.id = c.id) WHERE p.id = 1000;`	Q6: `SELECT COUNT(*) FROM parent_table p JOIN child_table c ON (p.id = c.id) WHERE p.id = 1000;`
T12	`INSERT INTO parent_table (id, s) VALUES (1000, 'hello'); INSERT INTO child_table (id, s) VALUES (1000, 'world'); COMMIT;`		
T13		Q5 が終了すると、結果は 0 になります。	Q6 終了すると、結果は 0 または 1 になります。

他のトランザクションが DML ステートメントを実行してコミットする前にクエリが迅速に終了する場合、結果は予測可能であり、プライマリインスタンスでも Aurora レプリカでも同じ結果になります。最初のクエリから始めて、動作の違いを詳しく調べてみましょう。

プライマリインスタンスでの `READ COMMITTED` は `REPEATABLE READ` 分離レベルと同様の強力な整合性モデルを使用するため、Q1 の結果の予測可能性が非常に高くなります。

Q2 の結果は、クエリの実行中にコミットされるトランザクションに応じて異なる場合があります。例えば、他のトランザクションが DML ステートメントを実行し、クエリの実行中にコミットするとします。この場合、Aurora レプリカでの `READ COMMITTED` 分離レベルを使用したクエリでは、変

更が考慮される場合と考慮されない場合があります。行数は、REPEATABLE READ 分離レベルの場合と同じ方法では予測できません。さらに、プライマリインスタンスまたは RDS for MySQL インスタンスにおいて READ COMMITTED 分離レベルで実行されるクエリほど予測可能ではありません。

T7 の UPDATE ステートメントは、実際にはテーブル内の行数を変更しません。ただし、可変長列の長さを変更すると、このステートメントによって行が内部的に再編成される可能性があります。長時間実行される READ COMMITTED トランザクションでは、古いバージョンの行が表示され、同じクエリ内で後から同じ行の新しいバージョンが表示される場合があります。クエリでは、行の古いバージョンと新しいバージョンの両方をスキップすることもできます。そのため、行カウントが予想と異なる場合があります。

Q5 と Q6 の結果は、同じである場合と、わずかに異なる場合があります。Aurora レプリカにおける READ COMMITTED でのクエリ Q6 は、クエリの実行中にコミットされた新しい行を表示できますが、表示する必要はありません。また、一方のテーブルの行は表示され、もう一方のテーブルの行は表示されないこともあります。結合クエリは、両方のテーブルで一致する行を見つけられない場合、ゼロのカウントを返します。クエリは、PARENT_TABLE と CHILD_TABLE の両方で新しい行を検出した場合、カウント 1 を返します。長時間実行されるクエリでは、結合されたテーブルからの検索が行われる時間に大きな分離が生じる可能性があります。

> **Note**
>
> これらの動作の違いは、トランザクションがコミットされるタイミングと、基になるテーブル行をクエリが処理するタイミングによって異なります。したがって、このような違いが見られる可能性が最も高くなるのは、数分または数時間かかるレポートクエリ、および OLTP トランザクションを同時に処理する Aurora クラスターで実行されるレポートクエリです。これらは、Aurora レプリカでの READ COMMITTED 分離レベルのメリットを最も享受する種類の混合ワークロードです。

Aurora MySQL のヒント

Aurora MySQL クエリで SQL ヒントを使用して、パフォーマンスを微調整できます。ヒントを使用して、重要なクエリの実行計画が予測不可能な条件のために変更されないようにすることもできます。

> **Tip**
>
> ヒントがクエリに与える影響を確認するには、EXPLAIN ステートメントによって生成されるクエリプランを調べます。ヒントの有無にかかわらず、クエリプランを比較します。

Aurora MySQL バージョン 3 では、MySQL Community Edition 8.0 で利用可能なすべてのヒントを使用できます。これらのヒントの詳細については、MySQL リファレンスマニュアルの「オプティマイザヒント」を参照してください。

これらのヒントは、Aurora MySQL バージョン 2 で使用可能です。これらのヒントは、Aurora MySQL バージョン 2 のハッシュ結合特徴を使用するクエリ、特にパラレルクエリ最適化を使用するクエリに適用されます。

PQ、NO_PQ

オプティマイザが、テーブル単位またはクエリ単位、どちらのパラレルクエリを使用するかを指定します。

PQ は、指定されたテーブルまたはクエリ全体 (ブロック) に対して、オプティマイザがパラレルクエリを使用するよう強制します。NO_PQ は、指定されたテーブルまたはクエリ全体 (ブロック) に対して、オプティマイザがパラレルクエリを使用しないようにします。

このヒントは、Aurora MySQL バージョン 2.11 以降で使用可能です。以下の例では、このヒントの使用方法を示します。

> **Note**
>
> テーブル名を指定した場合、オプティマイザは選択したテーブルにのみ PQ/NO_PQ ヒントを適用するようになります。テーブル名を指定しない場合、クエリブロックの影響を受けるすべてのテーブルに PQ/NO_PQ ヒントが強制されます。

```
EXPLAIN SELECT /*+ PQ() */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ PQ(t1) */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ PQ(t1,t2) */ f1, f2
```

```
    FROM num1 t1, num1 t2 WHERE t1.f1 = t2.f21;

EXPLAIN SELECT /*+ NO_PQ() */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ NO_PQ(t1) */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ NO_PQ(t1,t2) */ f1, f2
    FROM num1 t1, num1 t2 WHERE t1.f1 = t2.f21;
```

HASH_JOIN、NO_HASH_JOIN

クエリにハッシュ結合最適化メソッドを使用するかどうかを選択するパラレルクエリオプティマイザの機能をオンまたはオフにします。HASH_JOIN は、そのメカニズムがより効率的である場合、オプティマイザがハッシュ結合を使用できるようにします。NO_HASH_JOIN は、オプティマイザがクエリにハッシュ結合を使用できないようにします。このヒントは、Aurora MySQL バージョン 2.08 以降で使用可能です。Aurora MySQL バージョン 3 では効果がありません。

以下の例では、このヒントの使用方法を示します。

```
EXPLAIN SELECT/*+ HASH_JOIN(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ NO_HASH_JOIN(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;
```

HASH_JOIN_PROBING、NO_HASH_JOIN_PROBING

ハッシュ結合クエリで、結合のプローブ側に指定されたテーブルを使用するかどうかを指定します。クエリは、プローブテーブルの内容全体を読み取る代わりに、構築テーブルの列値がプローブテーブルに存在するかどうかをテストします。HASH_JOIN_PROBING および HASH_JOIN_BUILDING を使用して、クエリテキスト内のテーブルを並べ替えることなく、ハッシュ結合クエリの処理方法を指定できます。このヒントは、Aurora MySQL バージョン 2.08 以降で使用可能です。Aurora MySQL バージョン 3 では効果がありません。

以下の例では、このヒントの使用方法を示します。テーブル HASH_JOIN_PROBING の T2 ヒントを指定することは、テーブル NO_HASH_JOIN_PROBING に T1 を指定するのと同じ効果があります。

```
EXPLAIN SELECT /*+ HASH_JOIN(t2) HASH_JOIN_PROBING(t2) */ f1, f2
```

```
  FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ HASH_JOIN(t2) NO_HASH_JOIN_PROBING(t1) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;
```

HASH_JOIN_BUILDING、NO_HASH_JOIN_BUILDING

ハッシュ結合クエリで、結合の構築側に指定されたテーブルを使用するかどうかを指定します。クエリは、このテーブルのすべての行を処理し、他のテーブルと相互参照する列値のリストを作成します。HASH_JOIN_PROBING および HASH_JOIN_BUILDING を使用して、クエリテキスト内のテーブルを並べ替えることなく、ハッシュ結合クエリの処理方法を指定できます。このヒントは、Aurora MySQL バージョン 2.08 以降で使用可能です。Aurora MySQL バージョン 3 では効果がありません。

以下の例では、このヒントの使用方法を示します。テーブル HASH_JOIN_BUILDING の T2 ヒントを指定することは、テーブル NO_HASH_JOIN_BUILDING に T1 を指定するのと同じ効果があります。

```
EXPLAIN SELECT /*+ HASH_JOIN(t2) HASH_JOIN_BUILDING(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ HASH_JOIN(t2) NO_HASH_JOIN_BUILDING(t1) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;
```

JOIN_FIXED_ORDER

クエリ内のテーブルが、クエリにリストされている順序に基づいて結合されるように指定します。これは、3 つ以上のテーブルを含むクエリで特に便利です。これは、MySQL STRAIGHT_JOIN の代替となることを目的とし、MySQL JOIN_FIXED_ORDER ヒントに相当します。このヒントは、Aurora MySQL バージョン 2.08 以降で使用可能です。

以下の例では、このヒントの使用方法を示します。

```
EXPLAIN SELECT /*+ JOIN_FIXED_ORDER() */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_ORDER

クエリ内のテーブルの結合順序を指定します。これは、3 つ以上のテーブルを含むクエリで特に便利です。MySQL JOIN_ORDER ヒントに相当します。このヒントは、Aurora MySQL バージョン 2.08 以降で使用可能です。

以下の例では、このヒントの使用方法を示します。

```
EXPLAIN SELECT /*+ JOIN_ORDER (t4, t2, t1, t3) */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_PREFIX

結合順序の先頭に配置するテーブルを指定します。これは、3 つ以上のテーブルを含むクエリで特に便利です。MySQL JOIN_PREFIX ヒントに相当します。このヒントは、Aurora MySQL バージョン 2.08 以降で使用可能です。

以下の例では、このヒントの使用方法を示します。

```
EXPLAIN SELECT /*+ JOIN_PREFIX (t4, t2) */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_SUFFIX

結合順で最後に配置するテーブルを指定します。これは、3 つ以上のテーブルを含むクエリで特に便利です。MySQL JOIN_SUFFIX ヒントに相当します。このヒントは、Aurora MySQL バージョン 2.08 以降で使用可能です。

以下の例では、このヒントの使用方法を示します。

```
EXPLAIN SELECT /*+ JOIN_SUFFIX (t1) */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

ハッシュ結合クエリの使用方法については、「ハッシュ結合を使用した大規模な Aurora MySQL 結合クエリの最適化」を参照してください。

Aurora MySQL ストアドプロシージャ

Aurora MySQL DB クラスターを管理するには、組み込みストアドプロシージャを呼び出します。

トピック

- 設定
- セッションやクエリの終了
- ログ記録

- Global Status History の管理
- レプリケーション

設定

次のストアドプロシージャは、バイナリログファイルの保存などの設定パラメータを設定および表示します。

トピック

- mysql.rds_set_configuration
- mysql.rds_show_configuration

mysql.rds_set_configuration

バイナリログを保持する時間数またはレプリケーションを遅延させる秒数を指定します。

構文

```
CALL mysql.rds_set_configuration(name,value);
```

パラメータ

name

設定する設定パラメータの名前。

#

設定パラメータの値。

使用に関する注意事項

`mysql.rds_set_configuration` プロシージャは、以下の設定パラメータをサポートしています。

- バイナリログの保持時間

設定パラメータは永続的に保存され、DB インスタンスの再起動やフェイルオーバー後も存続します。

バイナリログの保持時間

`binlog retention hours` パラメータは、バイナリログファイルを保持する時間数を指定するために使用されます。Amazon Aurora では、通常、バイナリログは可能な限りすみやかに消去されますが、Aurora の外部にある MySQL データベースでのレプリケーションのためにバイナリログが必要になる場合があります。

`binlog retention hours` の初期値は `NULL` です。Aurora MySQL の場合、`NULL` はバイナリログが遅延してクリーンアップされることを意味します。Aurora MySQL バイナリログは、一定期間 (通常は 1 日以内)、システムに残ることがあります。

DB クラスターのバイナリログを保持する時間数を指定するには、`mysql.rds_set_configuration` ストアドプロシージャを使用して、次の例のように、レプリケーションを実行するのに十分な期間を指定します。

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

> **Note**
>
> `binlog retention hours` には、値 `0` は使用できません。

Aurora MySQL バージョン 2.11.0 以降、およびバージョン 3 DB クラスターの場合、最大 `binlog retention hours` 値は 2160 (90 日) です。

保持期間を設定したら、DB インスタンスのストレージ使用状況をモニタリングして、保持されたバイナリログに必要以上の容量が使用されないようにします。

mysql.rds_show_configuration

バイナリログを保持する時間数。

構文

```
CALL mysql.rds_show_configuration;
```

使用に関する注意事項

Amazon RDS がバイナリログを保持する時間数を確認するには、`mysql.rds_show_configuration` ストアドプロシージャを使用します。

例

以下の例では、保持期間を表示しています。

```
call mysql.rds_show_configuration;
                name                          value     description
                binlog retention hours        24        binlog retention hours specifies
 the duration in hours before binary logs are automatically deleted.
```

セッションやクエリの終了

次のストアドプロシージャは、セッションまたはクエリを終了します。

トピック

- mysql.rds_kill
- mysql.rds_kill_query

mysql.rds_kill

MySQL サーバーへの接続を終了します。

構文

```
CALL mysql.rds_kill(processID);
```

パラメータ

processID

終了する接続スレッドの識別子。

使用に関する注意事項

MySQL サーバーへの個々の接続は別々のスレッドで実行されます。接続を終了するには、mysql.rds_kill プロシージャを使用して、その接続のスレッド ID を渡します。スレッド ID を取得するには、MySQL の PROCESSLIST コマンドを使用します。

例

次の例では、4243 のスレッド ID を持つ接続を終了します。

```
CALL mysql.rds_kill(4243);
```

mysql.rds_kill_query

MySQL サーバーに対して実行中のクエリを終了します。

構文

```
CALL mysql.rds_kill_query(processID);
```

パラメータ

processID

終了するクエリを実行しているプロセスまたはスレッドの ID。

使用に関する注意事項

MySQL サーバーに対して実行しているクエリを終了するには、mysql_rds_kill_query プロシージャを使用して、クエリを実行しているスレッドの接続 ID を渡します。これにより、プロシージャは接続を終了します。

ID を取得するには、MySQL INFORMATION_SCHEMA PROCESSLIST テーブルまたは MySQL SHOW PROCESSLIST コマンドを使用します。SHOW PROCESSLIST または SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST の ID 列の値は *processID* です。

例

次の例では、クエリスレッド ID が 230040 のクエリを停止します。

```
CALL mysql.rds_kill_query(230040);
```

ログ記録

以下のストアドプロシージャは、MySQL ログをバックアップテーブルにローテーションします。詳細については、「Aurora MySQL データベースのログファイル」を参照してください。

トピック

- mysql.rds_rotate_general_log
- mysql.rds_rotate_slow_log

mysql.rds_rotate_general_log

mysql.general_log テーブルをバックアップテーブルとしてローテーションさせます。

構文

```
CALL mysql.rds_rotate_general_log;
```

使用に関する注意事項

mysql.general_log テーブルのバックアップテーブルとしてのローテーションは、mysql.rds_rotate_general_log プロシージャを呼び出すことで実行できます。ログテーブルのローテーションが実行されると、現在のログテーブルがバックアップのログテーブルにコピーされ、現在のログテーブル内にあるエントリは削除されます。バックアップのログテーブルがすでに存在する場合は、現在のログテーブルをバックアップにコピーする前に、削除されます。バックアップのログテーブルは、必要に応じて照会することができます。mysql.general_log テーブルに対するバックアップのログテーブルは、mysql.general_log_backup という名前になります。

log_output パラメータが TABLE に設定されている場合にのみ、このプロシージャを実行できます。

mysql.rds_rotate_slow_log

mysql.slow_log テーブルをバックアップテーブルとしてローテーションさせます。

構文

```
CALL mysql.rds_rotate_slow_log;
```

使用に関する注意事項

mysql.slow_log テーブルのバックアップテーブルとしてのローテーション
は、mysql.rds_rotate_slow_log プロシージャを呼び出すことで実行できます。ログテーブル
のローテーションが実行されると、現在のログテーブルがバックアップのログテーブルにコピーさ
れ、現在のログテーブル内にあるエントリは削除されます。バックアップのログテーブルがすでに存
在する場合は、現在のログテーブルをバックアップにコピーする前に、削除されます。

バックアップのログテーブルは、必要に応じて照会することができます。mysql.slow_log テーブ
ルに対するバックアップのログテーブルは、mysql.slow_log_backup という名前になります。

Global Status History の管理

Amazon RDS は、ステータス変数の値のスナップショットを掲示的に取得し、前回のスナップショット後の変化とともにテーブルに書き込む一連のプロシージャを提供します。このインフラストラクチャは Global Status History (GoSH) と呼ばれます。詳細については、「Managing the Global Status History」(Global Status History の管理) を参照してください。

以下のストアドプロシージャは、Global Status History (GoSH) の収集方法と保守方法を管理します。

トピック

- mysql.rds_collect_global_status_history
- mysql.rds_disable_gsh_collector
- mysql.rds_disable_gsh_rotation
- mysql.rds_enable_gsh_collector
- mysql.rds_enable_gsh_rotation
- mysql.rds_rotate_global_status_history
- mysql.rds_set_gsh_collector
- mysql.rds_set_gsh_rotation

mysql.rds_collect_global_status_history

Global Status History (GoSH) のスナップショットをオンデマンドで作成します。

構文

```
CALL mysql.rds_collect_global_status_history;
```

mysql.rds_disable_gsh_collector

Global Status History (GoSH) によって作成されたスナップショットを無効にします。

構文

```
CALL mysql.rds_disable_gsh_collector;
```

mysql.rds_disable_gsh_rotation

mysql.global_status_history テーブルのローテーションをオフにします。

構文

```
CALL mysql.rds_disable_gsh_rotation;
```

mysql.rds_enable_gsh_collector

Global Status History (GoSH) をオンにして、rds_set_gsh_collector で指定された間隔でデフォルトのスナップショットを作成します。

構文

```
CALL mysql.rds_enable_gsh_collector;
```

mysql.rds_enable_gsh_rotation

rds_set_gsh_rotation で指定された間隔での mysql.global_status_history テーブルのコンテンツの mysql.global_status_history_old へのローテーションをオンにします。

構文

```
CALL mysql.rds_enable_gsh_rotation;
```

mysql.rds_rotate_global_status_history

mysql.global_status_history テーブルのコンテンツを mysql.global_status_history_old にオンデマンドでローテーションします。

構文

```
CALL mysql.rds_rotate_global_status_history;
```

mysql.rds_set_gsh_collector

Global Status History (GoSH) によって作成されたスナップショットの間隔を分単位で指定します。

構文

```
CALL mysql.rds_set_gsh_collector(intervalPeriod);
```

パラメータ

intervalPeriod

スナップショット作成の間隔 (分単位)。デフォルト値は 5 です。

mysql.rds_set_gsh_rotation

mysql.global_status_history テーブルのローテーション間隔を日単位で指定します。

構文

```
CALL mysql.rds_set_gsh_rotation(intervalPeriod);
```

パラメータ

intervalPeriod

テーブルのローテーション間隔 (日単位)。デフォルト値は 7 です。

レプリケーション

Aurora MySQL クラスターのプライマリインスタンスに接続している間に、次のストアドプロシージャを呼び出すことができます。これらのプロシージャでは、トランザクションが外部データベースから Aurora MySQL、または Aurora MySQL から外部データベースに複製される方法を管理します。Aurora MySQL を使用して、グローバルトランザクション ID (GTIDs) に基づきレプリケーションを使用する方法については、「GTID ベースレプリケーションを使用する」を参照してください。

トピック

- mysql.rds_assign_gtids_to_anonymous_transactions (Aurora MySQL バージョン 3)
- mysql.rds_disable_session_binlog (Aurora MySQL バージョン 2)
- mysql.rds_enable_session_binlog (Aurora MySQL バージョン 2)
- mysql.rds_gtid_purged (Aurora MySQL バージョン 3)
- mysql.rds_import_binlog_ssl_material
- mysql.rds_next_master_log (Aurora MySQL バージョン 2)
- mysql.rds_next_source_log (Aurora MySQL バージョン 3)
- mysql.rds_remove_binlog_ssl_material
- mysql.rds_reset_external_master (Aurora MySQL バージョン 2)
- mysql.rds_reset_external_source (Aurora MySQL バージョン 3)
- mysql.rds_set_binlog_source_ssl (Aurora MySQL バージョン 3)
- mysql.rds_set_external_master (Aurora MySQL バージョン 2)
- mysql.rds_set_external_master_with_auto_position (Aurora MySQL バージョン 2)
- mysql.rds_set_external_source (Aurora MySQL バージョン 3)
- mysql.rds_set_external_source_with_auto_position (Aurora MySQL バージョン 3)
- mysql.rds_set_master_auto_position (Aurora MySQL バージョン 2)
- mysql.rds_set_read_only (Aurora MySQL バージョン 3)
- mysql.rds_set_session_binlog_format (Aurora MySQL バージョン 2)
- mysql.rds_set_source_auto_position (Aurora MySQL バージョン 3)
- mysql.rds_skip_transaction_with_gtid (Aurora MySQL バージョン 2 および 3)
- mysql.rds_skip_repl_error
- mysql.rds_start_replication

- mysql.rds_start_replication_until (Aurora MySQL バージョン 3)
- mysql.rds_start_replication_until_gtid (Aurora MySQL バージョン 3)
- mysql.rds_stop_replication

mysql.rds_assign_gtids_to_anonymous_transactions (Aurora MySQL バージョン 3)

を設定します。ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONSのオプションCHANGE REPLICATION SOURCE TO表示されます。これにより、レプリケーションチャネルが GTID を持たないレプリケートされたトランザクションに割り当てられます。このように、GTID ベースのレプリケーションを使用しないソースから、使用するレプリカに対してバイナリログレプリケーションを実行できます。詳細については、「CHANGE REPLICATION SOURCE TO Statement」を参照してください。そしてGTIDs のないソースから GTIDs を使用したレプリカへのレプリケーションのMySQL リファレンスマニュアルを参照してください。

Syntax

```
CALL mysql.rds_assign_gtids_to_anonymous_transactions(gtid_option);
```

パラメータ

gtid_option

文字列値。指定できる値は次のとおりです。OFF,LOCAL、または指定された UUID。

使用に関する注意事項

このプロシージャは、コミュニティMySQLでステートメントの発行と同じ効果がありますCHANGE REPLICATION SOURCE TO ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS = *gtid_option*。

GTID をONに変える必要があります*####gtid_option*または設定されるLOCALまたは特定の UUID に設定されます。

デフォルトは OFF であり、この機能が使用されないことを意味します。

LOCALレプリカ独自の UUID を含む GTID を割り当てます (server_uuid設定)。

UUID であるパラメータを渡すと、指定された UUID を含む GTID が割り当てられます。server_uuidレプリケーションソースサーバーの設定。

例

この特徴をオフにするには、次の手順を実行します:

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('OFF');
+-------------------------------------------------------------+
| Message  |
+-------------------------------------------------------------+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: OFF |
+-------------------------------------------------------------+
1 row in set (0.07 sec)
```

レプリカ独自の UUID を使用するには、次の手順を実行します:

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('LOCAL');
+------------------------------------------------------------+
| Message  |
+------------------------------------------------------------+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: LOCAL |
+------------------------------------------------------------+
1 row in set (0.07 sec)
```

指定した UUID を使用するには、次の手順を実行します:

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('317a4760-
f3dd-3b74-8e45-0615ed29de0e');
+-----------------------------------------------------------------------------------------------------
+
| Message |
+-----------------------------------------------------------------------------------------------------
+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: 317a4760-
f3dd-3b74-8e45-0615ed29de0e |
+-----------------------------------------------------------------------------------------------------
+
1 row in set (0.07 sec)
```

mysql.rds_disable_session_binlog (Aurora MySQL バージョン 2)

sql_log_bin 変数を OFF に設定することにより、現在のセッションのバイナリログをオフにします。

Syntax

```
CALL mysql.rds_disable_session_binlog;
```

パラメータ

なし

使用に関する注意事項

Aurora MySQL DB クラスターでは、プライマリインスタンスに接続されている間に、このストアドプロシージャを呼び出します。

Aurora では、このプロシージャは Aurora MySQL バージョン 2.12 以降の MySQL 5.7 互換バージョンでサポートされています。

> **Note**
>
> Aurora MySQL バージョン 3 では、SESSION_VARIABLES_ADMIN 権限を持っている場合、次のコマンドを使用して現在のセッションのバイナリログ記録を無効にできます。
>
> ```
> SET SESSION sql_log_bin = OFF;
> ```

mysql.rds_enable_session_binlog (Aurora MySQL バージョン 2)

sql_log_bin 変数を ON に設定することにより、現在のセッションのバイナリログをオンにします。

Syntax

```
CALL mysql.rds_enable_session_binlog;
```

パラメータ

なし

使用に関する注意事項

Aurora MySQL DB クラスターでは、プライマリインスタンスに接続されている間に、このストアドプロシージャを呼び出します。

Aurora では、このプロシージャは Aurora MySQL バージョン 2.12 以降の MySQL 5.7 互換バージョンでサポートされています。

> **Note**
>
> Aurora MySQL バージョン 3 では、SESSION_VARIABLES_ADMIN 権限を持っている場合、次のコマンドを使用して現在のセッションのバイナリログ記録を有効にできます。
>
> ```
> SET SESSION sql_log_bin = ON;
> ```

mysql.rds_gtid_purged (Aurora MySQL バージョン 3)

システム変数 gtid_purged のグローバル値を特定のグローバルトランザクション識別子 (GTID) セットに設定します。gtid_purged システム変数は、サーバー上でコミットされたが、サーバー上のどのバイナリログファイルにも存在しないすべてのトランザクションの GTID で構成される GTID セットです。

MySQL 8.0 との互換性を保つため、gtid_purged の値を設定する方法が 2 つあります。

- gtid_purged の値を、指定した GTID セットに置き換えます。
- 指定した GTID セットを gtid_purged が既に含んでいる GTID セットに追加します。

Syntax

gtid_purged の値を、指定した GTID セットに置き換えるには

```
CALL mysql.rds_gtid_purged (gtid_set);
```

gtid_purged の値を、指定した GTID セットに追加するには

```
CALL mysql.rds_gtid_purged (+gtid_set);
```

パラメータ

gtid_set

gtid_set の値は、gtid_purged の現在値のスーパーセットでなければならず、gtid_subtract(gtid_executed,gtid_purged) と交差することはできません。つま

り、新しい GTID セットには、gtid_purged に既に存在していた GTID がすべて含まれている必要がありますが、まだパージされていなかった gtid_executed 内の GTID を含むことはできません。また、*gtid_set* パラメータは、グローバル gtid_owned セットにある GTID、サーバー上で現在処理中のトランザクションの GTID を含むことはできません。

使用に関する注意事項

マスターユーザーが mysql.rds_gtid_purged を実行する必要があります。

この手順は Aurora MySQL バージョン 3.04 以降でサポートされています。

例

次の例では、GTID 3E11FA47-71CA-11E1-9E33-C80AA9429562:23 を gtid_purged グローバル変数に代入します。

```
CALL mysql.rds_gtid_purged('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

mysql.rds_import_binlog_ssl_material

認証局証明書、クライアント証明書、およびクライアントキーを Aurora MySQL DB クラスターにインポートします。この情報は SSL 通信および暗号化レプリケーションに必要です。

> **Note**
>
> 現在、この手順は、Aurora MySQL バージョン 2 (2.09.2、2.10.0、2.10.1、2.11.0) とバージョン 3 (3.01.1 以降) でサポートされています。

Syntax

```
CALL mysql.rds_import_binlog_ssl_material (
  ssl_material
);
```

パラメータ

ssl_material

MySQL クライアント用の以下の .pem 形式のコンテンツを含む JSON ペイロード。

- "ssl_ca":"######"
- "ssl_cert":"#########"
- "ssl_key":"########"

使用に関する注意事項

この手順を実行する前に、暗号化レプリケーションを準備します。

- 外部の MySQL 出典データベースインスタンスで有効になった SSL がなく、またクライアントキーおよびクライアント証明書が準備されていない場合、MySQL データベースサーバーで SSL を有効にし、必要なクライアントキーおよびクライアントの証明書を生成します。
- SSL が外部出典データベースインスタンスで有効になっている場合は、Aurora MySQL DB クラスターにクライアントキーおよび証明書を提供します。これらがない場合は、Aurora MySQL DB クラスター用に新しいキーと証明書を生成します。クライアント証明書に署名するには、外部の MySQL 出典データベースインスタンスで SSL の設定に使用した認証局キーが必要です。

詳細については、MySQL ドキュメントの「Creating SSL Certificates and Keys Using openssl」を参照してください。

> **Important**
>
> 暗号化レプリケーションをじゅんびしたら、SSL 接続を使用してこの手順を実行します。クライアントのキーは、安全ではない接続で転送するべきではありません。

この手順では、外部の MySQL データベースから SSL 情報を Aurora MySQL DB クラスターにインポートします。SSL 情報は、Aurora MySQL DB クラスター用の SSL 情報が含まれる .pem 形式のファイルにあります。暗号化のレプリケーション中、Aurora MySQL DB クラスターはクライアントとして MySQL データベースサーバーに動作します。Aurora MySQL 用の証明書およびキーは、.pem 形式のファイルにあります。

上記のファイルからこの情報を正しい JSON ペイロードで `ssl_material` パラメータにコピーできます。暗号化レプリケーションをサポートするために、この SSL 情報を Aurora MySQL DB クラスターにインポートします。

JSON ペイロードは、次のようになります。

```
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
ssl_ca_pem_body_code
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
ssl_cert_pem_body_code
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
ssl_key_pem_body_code
-----END RSA PRIVATE KEY-----\n"}'
```

例

次の例では、SSL 情報を Aurora MySQL にインポートします。.pem 形式ファイルでは、通常の場合、コード本文に例に示されるコード本文より長くなっています。

```
call mysql.rds_import_binlog_ssl_material(
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

mysql.rds_next_master_log (Aurora MySQL バージョン 2)

出典データベースインスタンスのログの位置を、出典データベースインスタンスの次のバイナリログの先頭に変更します。このプロシージャは、リードレプリカでレプリケーション I/O エラー 1236 が発生した場合にのみ使用してください。

Syntax

```
CALL mysql.rds_next_master_log(
curr_master_log
);
```

パラメータ

curr_master_log

現在のマスターログファイルのインデックス。例えば、現在のファイルが mysql-bin-changelog.012345 という名前の場合は、インデックスは 12345 になります。現在のマスターログファイルの名前を調べるには、SHOW REPLICA STATUS コマンドを実行し、Master_Log_File フィールドを確認します。

> **Note**
>
> MySQL の旧バージョンは、SHOW SLAVE STATUS ではなく SHOW REPLICA STATUS を使用していました。8.0.23 より前の MySQL バージョンを使用している場合は、SHOW SLAVE STATUS を使用します。

使用に関する注意事項

マスターユーザーが mysql.rds_next_master_log を実行する必要があります。

> **Warning**
>
> mysql.rds_next_master_log は、レプリケーション出典であるマルチ AZ DB インスタンスのフェイルオーバーの後でレプリケーションが失敗し、Last_IO_Errno の SHOW REPLICA STATUS フィールドが I/O エラー 1236 を示している場合にのみ呼び出してください。
>
> mysql.rds_next_master_log を呼び出すと、フェイルオーバーイベントが発生する前にソースインスタンスのトランザクションがディスク上のバイナリログに書き込まれなかった場合、リードレプリカのデータが失われる可能性があります。

例

Aurora MySQL リードレプリカでレプリケーションが失敗すると仮定します。リードレプリカで SHOW REPLICA STATUS\G を実行すると、次の結果が返されます。

```
*************************** 1. row ***************************
             Replica_IO_State:
                  Source_Host: myhost.XXXXXXXXXXXXXX.rr-rrrr-1.rds.amazonaws.com
                  Source_User: MasterUser
                  Source_Port: 3306
                Connect_Retry: 10
              Source_Log_File: mysql-bin-changelog.012345
          Read_Source_Log_Pos: 1219393
               Relay_Log_File: relaylog.012340
                Relay_Log_Pos: 30223388
        Relay_Source_Log_File: mysql-bin-changelog.012345
           Replica_IO_Running: No
          Replica_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Source_Log_Pos: 30223232
              Relay_Log_Space: 5248928866
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Source_SSL_Allowed: No
           Source_SSL_CA_File:
           Source_SSL_CA_Path:
              Source_SSL_Cert:
            Source_SSL_Cipher:
               Source_SSL_Key:
        Seconds_Behind_Master: NULL
Source_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 1236
                Last_IO_Error: Got fatal error 1236 from master when reading data from
 binary log: 'Client requested master to start replication from impossible position;
 the first event 'mysql-bin-changelog.013406' at 1219393, the last event read from
 '/rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4, the last byte read from '/
rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4.'
               Last_SQL_Errno: 0
               Last_SQL_Error:
```

```
  Replicate_Ignore_Server_Ids:
             Source_Server_Id: 67285976
```

Last_IO_Errno フィールドはインスタンスが I/O エラー 1236 を受け取ったことを示します。Master_Log_File フィールドは、ファイル名が mysql-bin-changelog.012345 であることを示しています。これは、ログファイルのインデックスが 12345 であることを表しています。エラーを解決するには、次のパラメータを使用して mysql.rds_next_master_log を呼び出すことができます。

```
CALL mysql.rds_next_master_log(12345);
```

> **Note**
>
> MySQL の旧バージョンは、SHOW SLAVE STATUS ではなく SHOW REPLICA STATUS を使用していました。8.0.23 より前の MySQL バージョンを使用している場合は、SHOW SLAVE STATUS を使用します。

mysql.rds_next_source_log (Aurora MySQL バージョン 3)

出典データベースインスタンスのログの位置を、出典データベースインスタンスの次のバイナリログの先頭に変更します。このプロシージャは、リードレプリカでレプリケーション I/O エラー 1236 が発生した場合にのみ使用してください。

Syntax

```
CALL mysql.rds_next_source_log(
curr_source_log
);
```

パラメータ

curr_source_log

現在のソースログファイルのインデックス。例えば、現在のファイルが mysql-bin-changelog.012345 という名前の場合は、インデックスは 12345 になります。現在のソースログファイルの名前を調べるには、SHOW REPLICA STATUS コマンドを実行し、Source_Log_File フィールドを確認します。

使用に関する注意事項

マスターユーザーが mysql.rds_next_source_log を実行する必要があります。

> **Warning**
>
> mysql.rds_next_source_log は、レプリケーション出典であるマルチ AZ DB インスタンスのフェイルオーバーの後でレプリケーションが失敗し、Last_IO_Errno の SHOW REPLICA STATUS フィールドが I/O エラー 1236 を示している場合にのみ呼び出してください。
>
> mysql.rds_next_source_log を呼び出すと、フェイルオーバーイベントが発生する前にソースインスタンスのトランザクションがディスク上のバイナリログに書き込まれなかった場合、リードレプリカのデータが失われる可能性があります。

例

Aurora MySQL リードレプリカでレプリケーションが失敗すると仮定します。リードレプリカで SHOW REPLICA STATUS\G を実行すると、次の結果が返されます。

```
*************************** 1. row ***************************
             Replica_IO_State:
                  Source_Host: myhost.XXXXXXXXXXXXXX.rr-rrrr-1.rds.amazonaws.com
                  Source_User: MasterUser
                  Source_Port: 3306
                Connect_Retry: 10
              Source_Log_File: mysql-bin-changelog.012345
          Read_Source_Log_Pos: 1219393
               Relay_Log_File: relaylog.012340
                Relay_Log_Pos: 30223388
        Relay_Source_Log_File: mysql-bin-changelog.012345
           Replica_IO_Running: No
          Replica_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
```

```
                  Skip_Counter: 0
          Exec_Source_Log_Pos: 30223232
              Relay_Log_Space: 5248928866
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Source_SSL_Allowed: No
           Source_SSL_CA_File:
           Source_SSL_CA_Path:
              Source_SSL_Cert:
            Source_SSL_Cipher:
               Source_SSL_Key:
        Seconds_Behind_Source: NULL
Source_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 1236
                Last_IO_Error: Got fatal error 1236 from source when reading data from
 binary log: 'Client requested source to start replication from impossible position;
 the first event 'mysql-bin-changelog.013406' at 1219393, the last event read from
 '/rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4, the last byte read from '/
rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4.'
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Source_Server_Id: 67285976
```

Last_IO_Errno フィールドはインスタンスが I/O エラー 1236 を受け取ったことを示します。Source_Log_File フィールドは、ファイル名が mysql-bin-changelog.012345 であることを示しています。これは、ログファイルのインデックスが 12345 であることを表しています。エラーを解決するには、次のパラメータを使用して mysql.rds_next_source_log を呼び出すことができます。

```
CALL mysql.rds_next_source_log(12345);
```

mysql.rds_remove_binlog_ssl_material

SSL 通信と暗号化レプリケーション用の認証局証明書、クライアント証明書およびクライアントキーを削除します。mysql.rds_import_binlog_ssl_material を使用してこの情報をインポートします。

Syntax

```
CALL mysql.rds_remove_binlog_ssl_material;
```

mysql.rds_reset_external_master (Aurora MySQL バージョン 2)

Aurora MySQL DB インスタンスを、Amazon RDS の外部で実行している MySQL インスタンスのリードレプリカとして使用しないように再設定します。

> **Important**
>
> この手順を実行するには、`autocommit` を有効にする必要があります。これを有効にするには、`autocommit` パラメータを 1 に設定します。パラメータの変更については、「DB パラメータグループのパラメータの変更」を参照してください。

Syntax

```
CALL mysql.rds_reset_external_master;
```

使用に関する注意事項

マスターユーザーが `mysql.rds_reset_external_master` を実行する必要があります。このプロシージャは、Amazon RDS の外部で動作する MySQL インスタンスのリードレプリカとしての設定が解除される MySQL DB インスタンス上で実行する必要があります。

> **Note**
>
> これらのストアドプロシージャは、主に Amazon RDS 外部で実行されている MySQL インスタンスのレプリケーションの有効化のために提供されています。可能な場合は、Aurora MySQL DB クラスター内のレプリケーションを管理するために、Aurora レプリカを使用することをお勧めします。Aurora MySQL DB クラスターでのレプリカの管理については、「Aurora レプリカの使用」を参照してください。

レプリケーションを使用して、Aurora MySQL の外部で実行している MySQL インスタンスからデータをインポートする方法の詳細については、「Aurora と MySQL との間、または Aurora と別の Aurora DB クラスターとの間のレプリケーション (バイナリログレプリケーション)」を参照してください。

mysql.rds_reset_external_source (Aurora MySQL バージョン 3)

Aurora MySQL DB インスタンスを、Amazon RDS の外部で実行している MySQL インスタンスのリードレプリカとして使用しないように再設定します。

> **Important**
>
> この手順を実行するには、`autocommit` を有効にする必要があります。これを有効にするには、`autocommit` パラメータを `1` に設定します。パラメータの変更については、「DB パラメータグループのパラメータの変更」を参照してください。

Syntax

```
CALL mysql.rds_reset_external_source;
```

使用に関する注意事項

マスターユーザーが `mysql.rds_reset_external_source` を実行する必要があります。このプロシージャは、Amazon RDS の外部で動作する MySQL インスタンスのリードレプリカとしての設定が解除される MySQL DB インスタンス上で実行する必要があります。

> **Note**
>
> これらのストアドプロシージャは、主に Amazon RDS 外部で実行されている MySQL インスタンスのレプリケーションの有効化のために提供されています。可能な場合は、Aurora MySQL DB クラスター内のレプリケーションを管理するために、Aurora レプリカを使用することをお勧めします。Aurora MySQL DB クラスターでのレプリカの管理については、「Aurora レプリカの使用」を参照してください。

mysql.rds_set_binlog_source_ssl (Aurora MySQL バージョン 3)

バイナリログレプリケーションの `SOURCE_SSL` 暗号化を有効にします。詳細については、MySQL ドキュメントの「ステートメントに対してレプリケーションソースを変更する」を参照してください。

Syntax

```
CALL mysql.rds_set_binlog_source_ssl(mode);
```

パラメータ

###

SOURCE_SSL 暗号化が有効になっているかを示す次の値:

- 0 — SOURCE_SSL 暗号化は無効です。デフォルト: 0。
- 1 — SOURCE_SSL 暗号化は有効です。SSL または TLS を使用して暗号化を設定できます。

使用に関する注意事項

この手順は Aurora MySQL バージョン 3.06 以降でサポートされています。

mysql.rds_set_external_master (Aurora MySQL バージョン 2)

Aurora MySQL DB インスタンスを、Amazon RDS の外部で実行している MySQL インスタンスのリードレプリカとして使用するように設定します。

mysql.rds_set_external_master プロシージャは廃止されており、将来のリリースでは削除されます。代わりに mysql.rds_set_external_source を使用します。

> **Important**
>
> この手順を実行するには、autocommit を有効にする必要があります。これを有効にするには、autocommit パラメータを 1 に設定します。パラメータの変更については、「DB パラメータグループのパラメータの変更」を参照してください。

Syntax

```
CALL mysql.rds_set_external_master (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
```

```
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);
```

パラメータ

host_name

出典データベースインスタンスとなる、Amazon RDS の外部で動作する MySQL インスタンスのホスト名または IP アドレス。

host_port

Amazon RDS の外部で動作する MySQL インスタンス (出典データベースインスタンス) で使用されるポート。ポート番号を変換する Secure Shell (SSH) ポートのレプリケーションがネットワーク設定に含まれる場合、SSH によって発行されるポート番号を指定します。

replication_user_name

Amazon RDS の外部で実行される MySQL インスタンスの `REPLICATION CLIENT` および `REPLICATION SLAVE` のアクセス許可を持つユーザーの ID。外部インスタンスのレプリケーション専用のアカウントを使用することをお勧めします。

replication_user_password

`replication_user_name` で指定されたユーザー ID のパスワード。

mysql_binary_log_file_name

レプリケーション情報を含む出典データベースインスタンス上のバイナリログの名前。

mysql_binary_log_file_location

`mysql_binary_log_file_name` バイナリログ内の場所。レプリケーションでは、この場所からレプリケーション情報の読み取りをスタートします。

binlog ファイルの名前と場所は、`SHOW MASTER STATUS`出典データベースインスタンス上で実行することによって決定できます。

ssl_encryption

レプリケーション接続で Secure Socket Layer (SSL) 暗号化を使用するかどうかを指定する値。1 は SSL 暗号化を使用することを指定し、0 は暗号化を使用しないことを指定します。デフォルトは 0 です。

> **Note**
>
> MASTER_SSL_VERIFY_SERVER_CERT オプションはサポートされていません。このオプションは 0 に設定されます。つまり、接続は暗号化されますが、証明書は検証されません。

使用に関する注意事項

マスターユーザーが mysql.rds_set_external_master を実行する必要があります。このプロシージャは、Amazon RDS の外部で動作する MySQL インスタンスのリードレプリカとして設定される MySQL DB インスタンス上で実行する必要があります。

mysql.rds_set_external_master を実行する前に、Amazon RDS の外部で動作する MySQL インスタンスを出典データベースインスタンスとして必ず設定してください。Amazon RDS の外部で動作する MySQL インスタンスに接続するには、MySQL の外部インスタンスの replication_user_name および replication_user_password アクセス権限を持つレプリケーションユーザーを示す REPLICATION CLIENT および REPLICATION SLAVE の値を指定する必要があります。

MySQL の外部インスタンスを出典データベースインスタンスとして設定するには

1. 選択した MySQL クライアントを使用して、MySQL の外部インスタンスに接続し、レプリケーションに使用されるユーザーアカウントを作成します。次に例を示します。

   MySQL 5.7

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
   ```

   MySQL 8.0

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED WITH mysql_native_password BY
    'password';
   ```

   > **Note**
   >
   > セキュリティ上のベストプラクティスとして、ここに示されているプロンプト以外のパスワードを指定してください。

2. MySQL の外部インスタンスで、REPLICATION CLIENT と REPLICATION SLAVE の権限をレプリケーションユーザーに付与します。次の例では、ドメインの「repl_user」ユーザーに、すべてのデータベースの REPLICATION CLIENT および REPLICATION SLAVE 権限を付与します。

   MySQL 5.7

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
    IDENTIFIED BY 'password';
   ```

   MySQL 8.0

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
   ```

暗号化されたレプリケーションを使用するには、SSL 接続を使用するようにソースデータベースインスタンスを設定します。また、mysql.rds_import_binlog_ssl_material 手順を使用して、認証局証明書、クライアント証明書、クライアントキーを DB インスタンスまたは DB クラスターにインポートします。

> **Note**
>
> これらのストアドプロシージャは、主に Amazon RDS 外部で実行されている MySQL インスタンスのレプリケーションの有効化のために提供されています。可能な場合は、Aurora MySQL DB クラスター内のレプリケーションを管理するために、Aurora レプリカを使用することをお勧めします。Aurora MySQL DB クラスターでのレプリカの管理については、「Aurora レプリカの使用」を参照してください。

mysql.rds_set_external_master を呼び出して Amazon RDS DB インスタンスをリードレプリカとして設定した後で、このリードレプリカで mysql.rds_start_replication を呼び出してレプリケーションプロセスをスタートできます。mysql.rds_reset_external_master (Aurora MySQL バージョン 2) を呼び出して、リードレプリカの設定を削除することもできます。

mysql.rds_set_external_master が呼び出されると、Amazon RDS では、時刻、ユーザー、set master アクションが mysql.rds_history テーブルと mysql.rds_replication_status テーブルに記録されます。

例

MySQL DB インスタンス上で実行すると、DB インスタンスが Amazon RDS の外部で動作する MySQL インスタンスのリードレプリカとして設定されます。次に例を示します。

```
call mysql.rds_set_external_master(
  'Externaldb.some.com',
  3306,
  'repl_user',
  'password',
  'mysql-bin-changelog.0777',
  120,
  0);
```

mysql.rds_set_external_master_with_auto_position (Aurora MySQL バージョン 2)

外部の MySQL インスタンスからの受信レプリケーションを受け入れるように Aurora MySQL プライマリインスタンスを設定します。このプロシージャでも、グローバルトランザクション識別子 (GTIDs) に基づき、レプリケーションが設定されます。

Aurora MySQL は遅延レプリケーションをサポートしていないため、この手順では遅延レプリケーションは設定されません。

Syntax

```
CALL mysql.rds_set_external_master_with_auto_position (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , ssl_encryption
);
```

パラメータ

host_name

レプリケーションマスターとなる、Aurora の外部で動作する MySQL インスタンスのホスト名または IP アドレス。

host_port

Aurora の外部で動作する MySQL インスタンス (レプリケーションマスター) で使用されるポート。ポート番号を変換する Secure Shell (SSH) ポートのレプリケーションがネットワーク設定に含まれる場合、SSH によって発行されるポート番号を指定します。

replication_user_name

Aurora の外部で実行される MySQL インスタンスの REPLICATION CLIENT および REPLICATION SLAVE のアクセス許可を持つユーザーの ID。外部インスタンスのレプリケーション専用のアカウントを使用することをお勧めします。

replication_user_password

replication_user_name で指定されたユーザー ID のパスワード。

ssl_encryption

このオプションは、現在実装されていません。デフォルトは 0 です。

使用に関する注意事項

Aurora MySQL DB クラスターでは、プライマリインスタンスに接続されている間に、このストアドプロシージャを呼び出します。

マスターユーザーが mysql.rds_set_external_master_with_auto_position を実行する必要があります。マスターユーザーは、レプリケーションターゲットとして機能する Aurora MySQL DB クラスターのプライマリインスタンスでこのプロシージャを実行します。例えば、外部の MySQL DB インスタンスまたは Aurora MySQL DB クラスターのレプリケーションターゲットになります。

この手順は Aurora MySQL バージョン 2 でサポートされています。Aurora MySQL バージョン 3 の場合は、mysql.rds_set_external_source_with_auto_position (Aurora MySQL バージョン 3)代わりにプロシージャを使用します。

mysql.rds_set_external_master_with_auto_position を実行する前に、外部の MySQL DB インスタンスがレプリケーションマスターになるように設定します。外部の MySQL インスタンスに接続するには、replication_user_name および replication_user_password の値を指定します。これらの値を使用して、MySQL の外部インスタンスで REPLICATION CLIENT および REPLICATION SLAVE アクセス許可を持つレプリケーションユーザーを示す必要があります。

外部の MySQL インスタンスをレプリケーションマスターとして設定するには

1. 選択した MySQL クライアントを使用して、外部の MySQL インスタンスに接続し、レプリケーションに使用されるユーザーアカウントを作成します。次に例を示します。

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
```

2. 外部の MySQL インスタンスで、REPLICATION CLIENT と REPLICATION SLAVE の特権をレプリケーションユーザーに付与します。次の例では、ドメインの REPLICATION CLIENT ユーザーに、すべてのデータベースの REPLICATION SLAVE および 'repl_user' 権限を付与します。

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
IDENTIFIED BY 'SomePassW0rd'
```

mysql.rds_set_external_master_with_auto_position を呼び出すと、Amazon RDS によって特定の情報が記録されます。この情報には、時刻、ユーザー、"set master" の mysql.rds_history テーブルの mysql.rds_replication_status のアクションがあります。

問題の原因となることがわかっている特定の GTID ベースのトランザクションをスキップするには、mysql.rds_skip_transaction_with_gtid ストアドプロシージャを使用します。GTID ベースのレプリケーションの使用に関する詳細については、「GTID ベースレプリケーションを使用する」を参照してください。

例

Aurora プライマリインスタンス上で実行すると、以下の例では、Aurora の外部で動作する MySQL のインスタンスのリードレプリカとして動作するように Aurora クラスターを設定します。

```
call mysql.rds_set_external_master_with_auto_position(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'SomePassW0rd');
```

mysql.rds_set_external_source (Aurora MySQL バージョン 3)

MySQL DB インスタンスを、Amazon RDS の外部で実行している MySQL インスタンスのリードレプリカとして設定します。

> **Important**
>
> この手順を実行するには、autocommit を有効にする必要があります。これを有効にするには、autocommit パラメータを 1 に設定します。パラメータの変更については、「DB パラメータグループのパラメータの変更」を参照してください。

Syntax

```
CALL mysql.rds_set_external_source (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);
```

パラメータ

host_name

出典データベースインスタンスとなる、Amazon RDS の外部で動作する MySQL インスタンスのホスト名または IP アドレス。

host_port

Amazon RDS の外部で動作する MySQL インスタンス (出典データベースインスタンス) で使用されるポート。ポート番号を変換する Secure Shell (SSH) ポートのレプリケーションがネットワーク設定に含まれる場合、SSH によって発行されるポート番号を指定します。

replication_user_name

Amazon RDS の外部で実行される MySQL インスタンスの REPLICATION CLIENT および REPLICATION SLAVE のアクセス許可を持つユーザーの ID。外部インスタンスのレプリケーション専用のアカウントを使用することをお勧めします。

replication_user_password

replication_user_name で指定されたユーザー ID のパスワード。

mysql_binary_log_file_name

レプリケーション情報を含む出典データベースインスタンス上のバイナリログの名前。

mysql_binary_log_file_location

mysql_binary_log_file_name バイナリログ内の場所。レプリケーションでは、この場所からレプリケーション情報の読み取りをスタートします。

binlog ファイルの名前と場所は、SHOW MASTER STATUS出典データベースインスタンス上で実行することによって決定できます。

ssl_encryption

レプリケーション接続で Secure Socket Layer (SSL) 暗号化を使用するかどうかを指定する値。1 は SSL 暗号化を使用することを指定し、0 は暗号化を使用しないことを指定します。デフォルトは 0 です。

> **Note**
>
> このオプションを有効にするには、mysql.rds_import_binlog_ssl_material を使用してカスタム SSL 証明書をインポートしておく必要があります。カスタム SSL 証明書をインポートしていない場合は、このパラメータを 0 に設定し、mysql.rds_set_binlog_source_ssl (Aurora MySQL バージョン 3) を使用してバイナリログのレプリケーション用の SSL を有効にします。
> MASTER_SSL_VERIFY_SERVER_CERT オプションはサポートされていません。このオプションは 0 に設定されます。つまり、接続は暗号化されますが、証明書は検証されません。

使用に関する注意事項

マスターユーザーが mysql.rds_set_external_source を実行する必要があります。このプロシージャは、Amazon RDS の外部で実行している MySQL インスタンスのリードレプリカとして設定される MySQL DB インスタンス上で実行する必要があります。

mysql.rds_set_external_source を実行する前に、Amazon RDS の外部で動作する MySQL インスタンスを出典データベースインスタンスとして必ず設定してください。Amazon RDS の外部で動作する MySQL インスタンスに接続するには、MySQL の外部インスタンスの replication_user_name および replication_user_password アクセス権限を持つレプリ

ケーションユーザーを示す REPLICATION CLIENT および REPLICATION SLAVE の値を指定する必要があります。

MySQL の外部インスタンスを出典データベースインスタンスとして設定するには

1. 選択した MySQL クライアントを使用して、MySQL の外部インスタンスに接続し、レプリケーションに使用されるユーザーアカウントを作成します。次に例を示します。

   MySQL 5.7

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
   ```

   MySQL 8.0

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED WITH mysql_native_password BY
    'password';
   ```

   > **Note**
   >
   > セキュリティ上のベストプラクティスとして、ここに示されているプロンプト以外のパスワードを指定してください。

2. MySQL の外部インスタンスで、REPLICATION CLIENT と REPLICATION SLAVE の権限をレプリケーションユーザーに付与します。次の例では、ドメインの「repl_user」ユーザーに、すべてのデータベースの REPLICATION CLIENT および REPLICATION SLAVE 権限を付与します。

   MySQL 5.7

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
    IDENTIFIED BY 'password';
   ```

   MySQL 8.0

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
   ```

暗号化されたレプリケーションを使用するには、SSL 接続を使用するようにソースデータベースインスタンスを設定します。また、mysql.rds_import_binlog_ssl_material プロシージャを使用し

て、DB インスタンスまたは DB クラスターに認証局証明書、クライアント証明書、およびクライアントキーをインポートします。

> **Note**
>
> これらのストアドプロシージャは、主に Amazon RDS 外部で実行されている MySQL インスタンスのレプリケーションの有効化のために提供されています。可能な場合は、Aurora MySQL DB クラスター内のレプリケーションを管理するために、Aurora レプリカを使用することをお勧めします。Aurora MySQL DB クラスターでのレプリカの管理については、「Aurora レプリカの使用」を参照してください。

`mysql.rds_set_external_source` を呼び出して、Aurora MySQL DB インスタンスをリードレプリカとして設定した後、このリードレプリカに対して mysql.rds_start_replication を呼び出して、レプリケーションプロセスを開始できます。mysql.rds_reset_external_source を呼び出して、リードレプリカ設定を削除できます。

`mysql.rds_set_external_source` が呼び出されると、Amazon RDS では、時刻、ユーザー、`set master` アクションが `mysql.rds_history` テーブルと `mysql.rds_replication_status` テーブルに記録されます。

例

Aurora MySQL DB インスタンスに対して実行すると、次の例の示されているように、DB インスタンスが Amazon RDS の外部で実行している MySQL インスタンスのリードレプリカとして設定されます。

```
call mysql.rds_set_external_source(
  'Externaldb.some.com',
  3306,
  'repl_user',
  'password',
  'mysql-bin-changelog.0777',
  120,
  0);
```

mysql.rds_set_external_source_with_auto_position (Aurora MySQL バージョン 3)

外部の MySQL インスタンスからの受信レプリケーションを受け入れるように Aurora MySQL プライマリインスタンスを設定します。このプロシージャでも、グローバルトランザクション識別子 (GTIDs) に基づき、レプリケーションが設定されます。

Syntax

```
CALL mysql.rds_set_external_source_with_auto_position (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , ssl_encryption
);
```

パラメータ

host_name

レプリケーションソースとなる、Aurora の外部で動作する MySQL インスタンスのホスト名または IP アドレス。

host_port

Aurora の外部で動作する MySQL インスタンス (レプリケーションソース) で使用されるポート。ポート番号を変換する Secure Shell (SSH) ポートのレプリケーションがネットワーク設定に含まれる場合、SSH によって発行されるポート番号を指定します。

replication_user_name

Aurora の外部で実行される MySQL インスタンスの `REPLICATION CLIENT` および `REPLICATION SLAVE` のアクセス許可を持つユーザーの ID。外部インスタンスのレプリケーション専用のアカウントを使用することをお勧めします。

replication_user_password

`replication_user_name` で指定されたユーザー ID のパスワード。

ssl_encryption

このオプションは、現在実装されていません。デフォルトは 0 です。

> **Note**
>
> mysql.rds_set_binlog_source_ssl (Aurora MySQL バージョン 3) を使用して、バイナリログレプリケーション用に SSL を有効にします。

使用に関する注意事項

Aurora MySQL DB クラスターでは、プライマリインスタンスに接続されている間に、このストアドプロシージャを呼び出します。

管理ユーザーは、mysql.rds_set_external_source_with_auto_positionプロシージャを実行しなければなりません。管理ユーザーは、レプリケーションターゲットとして機能する Aurora MySQL DB クラスターのプライマリインスタンスでこのプロシージャを実行します。例えば、外部の MySQL DB インスタンスまたは Aurora MySQL DB クラスターのレプリケーションターゲットになります。

このプロシージャは Aurora MySQL バージョン 3 でサポートされています。Aurora MySQL は遅延レプリケーションをサポートしていないため、この手順では遅延レプリケーションは設定されません。

mysql.rds_set_external_source_with_auto_position を実行する前に、外部の MySQL DB インスタンスがレプリケーションソースになるように設定します。外部の MySQL インスタンスに接続するには、replication_user_name および replication_user_password の値を指定します。これらの値を使用して、MySQL の外部インスタンスで REPLICATION CLIENT および REPLICATION SLAVE アクセス許可を持つレプリケーションユーザーを示す必要があります。

外部の MySQL インスタンスをレプリケーションソースとして設定するには

1. 選択した MySQL クライアントを使用して、外部の MySQL インスタンスに接続し、レプリケーションに使用されるユーザーアカウントを作成します。次に例を示します。

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
   ```

2. 外部の MySQL インスタンスで、REPLICATION CLIENT と REPLICATION SLAVE の特権をレプリケーションユーザーに付与します。次の例では、ドメインの REPLICATION CLIENT ユーザーに、すべてのデータベースの REPLICATION SLAVE および 'repl_user' 権限を付与します。

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
   ```

```
IDENTIFIED BY 'SomePassW0rd'
```

mysql.rds_set_external_source_with_auto_position を呼び出すと、Amazon RDS によって特定の情報が記録されます。この情報には、時刻、ユーザー、"set master" の mysql.rds_history テーブルの mysql.rds_replication_status のアクションがあります。

問題の原因となることがわかっている特定の GTID ベースのトランザクションをスキップするには、mysql.rds_skip_transaction_with_gtid/> ストアドプロシージャを使用します。GTID ベースのレプリケーションの使用に関する詳細については、「GTID ベースレプリケーションを使用する」を参照してください。

例

Aurora プライマリインスタンス上で実行すると、以下の例では、Aurora の外部で動作する MySQL のインスタンスのリードレプリカとして動作するように Aurora クラスターを設定します。

```
call mysql.rds_set_external_source_with_auto_position(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'SomePassW0rd');
```

mysql.rds_set_master_auto_position (Aurora MySQL バージョン 2)

バイナリログファイルの位置、または、グローバルトランザクション識別子 (GTID) ベースのレプリケーションモードを設定します。

Syntax

```
CALL mysql.rds_set_master_auto_position (
auto_position_mode
);
```

パラメータ

auto_position_mode

ファイルの位置に基づくレプリケーション、または GTID ベースのレプリケーションかどうかを示す値:

- `0` - バイナリログファイルの位置に基づくレプリケーション方法を使用します。デフォルト: `0`。
- `1` - GTID ベースのレプリケーション方法を使用します。

使用に関する注意事項

マスターユーザーが `mysql.rds_set_master_auto_position` を実行する必要があります。

この手順は Aurora MySQL バージョン 2 でサポートされています。

mysql.rds_set_read_only (Aurora MySQL バージョン 3)

DB インスタンスの `read_only` モードをグローバルにオンまたはオフにします。

Syntax

```
CALL mysql.rds_set_read_only(mode);
```

パラメータ

###

DB インスタンスの `read_only` モードがグローバルにオンまたはオフになっているかを示す値。

- `0` – OFF。デフォルト値は `0` です。
- `1` – ON

使用に関する注意事項

`mysql.rds_set_read_only` ストアドプロシージャは `read_only` パラメータのみを変更します。`innodb_read_only` パラメータはリーダー DB インスタンスでは変更できません。

`read_only` パラメータの変更は再起動しても持続しません。`read_only` に永続的な変更を加えるには、`read_only` DB クラスターパラメータを使用する必要があります。

この手順は Aurora MySQL バージョン 3.06 以降でサポートされています。

mysql.rds_set_session_binlog_format (Aurora MySQL バージョン 2)

現在のセッションのバイナリログ形式を設定します。

Syntax

```
CALL mysql.rds_set_session_binlog_format(format);
```

パラメータ

format

現在のセッションのバイナリログ形式を示す値:

- STATEMENT — レプリケーションソースは、SQL ステートメントに基づいてバイナリログにイベントを書き込みます。
- ROW — レプリケーションソースは、個々のテーブル行の変更を示すイベントをバイナリログに書き込みます。
- MIXED — ロギングは通常、SQL ステートメントに基づきますが、特定の条件下では行に切り替わります。詳細については、MySQL ドキュメントの「混合バイナリログ形式」を参照してください。

使用に関する注意事項

Aurora MySQL DB クラスターでは、プライマリインスタンスに接続されている間に、このストアドプロシージャを呼び出します。

このストアドプロシージャを使用するには、現在のセッションに対してバイナリログが設定されている必要があります。

Aurora では、このプロシージャは Aurora MySQL バージョン 2.12 以降の MySQL 5.7 互換バージョンでサポートされています。

mysql.rds_set_source_auto_position (Aurora MySQL バージョン 3)

バイナリログファイルの位置、または、グローバルトランザクション識別子 (GTID) ベースのレプリケーションモードを設定します。

構文

```
CALL mysql.rds_set_source_auto_position (auto_position_mode);
```

パラメータ

auto_position_mode

ファイルの位置に基づくレプリケーション、または GTID ベースのレプリケーションかどうかを示す値:

- `0` - バイナリログファイルの位置に基づくレプリケーション方法を使用します。デフォルト: `0`。
- 1 - GTID ベースのレプリケーション方法を使用します。

使用に関する注意事項

Aurora MySQL DB クラスターでは、プライマリインスタンスに接続されている間に、このストアドプロシージャを呼び出します。

管理ユーザーは、`mysql.rds_set_source_auto_position`プロシージャを実行すべきです。

mysql.rds_skip_transaction_with_gtid (Aurora MySQL バージョン 2 および 3)

Aurora プライマリインスタンスで、指定されたグローバルトランザクション識別子 (GTID) のあるトランザクションのレプリケーションをスキップします。

特定の GTID トランザクションが問題の原因となることが知られている場合、障害復旧のためにこのプロシージャを使用できます。このストアドプロシージャを使用して、問題となるトランザクションをスキップします。問題のあるトランザクションの例には、レプリケーションを無効にしたり、重要なデータを削除したり、DB インスタンスを利用不可にするトランザクションが含まれます。

Syntax

```
CALL mysql.rds_skip_transaction_with_gtid (
gtid_to_skip
);
```

パラメータ

gtid_to_skip

スキップするレプリケーショントランザクションの GTID。

使用に関する注意事項

マスターユーザーが mysql.rds_skip_transaction_with_gtid を実行する必要があります。

この手順は Aurora MySQL バージョン 2 および 3 でサポートされています。

例

次の例では、GTID 3E11FA47-71CA-11E1-9E33-C80AA9429562:23 を使用したトランザクションのレプリケーションをスキップします。

```
CALL mysql.rds_skip_transaction_with_gtid('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

mysql.rds_skip_repl_error

MySQL DB リードレプリカのレプリケーションエラーをスキップして、削除します。

Syntax

```
CALL mysql.rds_skip_repl_error;
```

使用に関する注意事項

マスターユーザーはリードレプリカに対して mysql.rds_skip_repl_error プロシージャを実行する必要があります。この手順の詳細については、「Skipping the current replication error」(現在のレプリケーションエラーをスキップする) を参照してください。

MySQL の SHOW REPLICA STATUS\G コマンドを実行して、エラーがあるかどうかを判断します。レプリケーションエラーが重要ではない場合、mysql.rds_skip_repl_error を実行して、エラーをスキップすることができます。複数のエラーがある場合、mysql.rds_skip_repl_error では、初期のエラーを削除してから、他のエラーが存在することを警告します。その後で SHOW REPLICA STATUS\G を使用して、次のエラーに対処するための適切な対応方法を判断することができます。戻り値の詳細については、MySQL ドキュメントの「SHOW REPLICA STATUS statement」(SHOW REPLICA STATUS ステートメント) を参照してください。

> **Note**
>
> MySQL の旧バージョンは、SHOW REPLICA STATUS ではなく SHOW SLAVE STATUS を使用していました。8.0.23 より前の MySQL バージョンを使用している場合は、SHOW SLAVE STATUS を使用します。

Aurora MySQL でのレプリケーションエラーの対処方法の詳細については、「MySQL のリードレプリケーションのエラーの診断と解決」を参照してください。

レプリケーション停止エラー

mysql.rds_skip_repl_error プロシージャを呼び出すと、レプリカがダウンしているか無効であることを示すエラーメッセージが表示されることがあります。

このエラーメッセージは、リードレプリカではなくプライマリインスタンスでプロシージャを実行した場合に表示されます。このプロシージャを機能させるには、リードレプリカに対してこのプロシージャを実行する必要があります。

このエラーメッセージは、リードレプリカに対してプロシージャを実行したが、レプリケーションを正常に再開できない場合にも表示されることがあります。

多数のエラーをスキップする必要がある場合は、レプリケーションの遅延により、バイナリログ (binlog) ファイルがデフォルトの保持期間を超えて増大する場合があります。この場合、バイナリログファイルがリードレプリカで再生される前に破棄されるため、致命的なエラーが発生することがあります。この破棄によりレプリケーションが停止し、mysql.rds_skip_repl_error コマンドを呼び出してレプリケーションエラーをスキップすることができなくなります。

この問題は、出典データベースインスタンスでバイナリログファイルの保持時間を増加させることで軽減できます。バイナリログ保持時間を長くすると、レプリケーションを再開し、必要に応じて mysql.rds_skip_repl_error コマンドを使用できるようになります。

バイナリログの保持期間を設定するには、「mysql.rds_set_configuration」の手順を使用して、DB クラスターのバイナリログファイルの保持期間に合わせて、'binlog retention hours' の設定パラメータを指定します。以下の例では、バイナリログファイルの保持期間を 48 時間に設定しています。

```
CALL mysql.rds_set_configuration('binlog retention hours', 48);
```

mysql.rds_start_replication

Aurora MySQL DB クラスターからのレプリケーションを開始します。

> **Note**
>
> mysql.rds_start_replication_until (Aurora MySQL バージョン 3) または mysql.rds_start_replication_until_gtid (Aurora MySQL バージョン 3) ストアドプロシージャを

使用して、Aurora MySQL DB インスタンスからのレプリケーションを開始し、指定したバイナリログファイルの場所でレプリケーションを停止できます。

Syntax

```
CALL mysql.rds_start_replication;
```

使用に関する注意事項

マスターユーザーが mysql.rds_start_replication を実行する必要があります。

Amazon RDS の外部の MySQL インスタンスからデータをインポートするには、mysql.rds_set_external_master または mysql.rds_set_external_source を呼び出してレプリケーション設定を構築した後、リードレプリカに対して mysql.rds_start_replication を呼び出して、レプリケーションプロセスを開始します。詳細については、「Aurora と MySQL との間、または Aurora と別の Aurora DB クラスターとの間のレプリケーション (バイナリログレプリケーション)」を参照してください。

Amazon RDS の外部の MySQL インスタンスにデータをエクスポートするには、リードレプリカに対して mysql.rds_start_replication と mysql.rds_stop_replication を呼び出して、バイナリログの消去などのレプリケーションアクションを制御します。詳細については、「Aurora と MySQL との間、または Aurora と別の Aurora DB クラスターとの間のレプリケーション (バイナリログレプリケーション)」を参照してください。

リードレプリカで mysql.rds_start_replication を呼び出すことで、mysql.rds_stop_replication の呼び出しによって前に停止したレプリケーションプロセスを再開することもできます。詳細については、「レプリケーション停止エラー」を参照してください。

mysql.rds_start_replication_until (Aurora MySQL バージョン 3)

Aurora MySQL DB クラスターからレプリケーションを開始し、指定したバイナリログファイルの場所でレプリケーションを停止します。

Syntax

```
CALL mysql.rds_start_replication_until (
```

```
replication_log_file
  , replication_stop_point
);
```

パラメータ

replication_log_file

レプリケーション情報を含む出典データベースインスタンス上のバイナリログの名前。

replication_stop_point

replication_log_file バイナリログ内でレプリケーションが停止する場所。

使用に関する注意事項

マスターユーザーが mysql.rds_start_replication_until を実行する必要があります。

この手順は Aurora MySQL バージョン 3.04 以降でサポートされています。

mysql.rds_start_replication_until ストアドプロシージャは、以下を含むマネージドレプリケーションではサポートされていません。

- AWS リージョン 間での Amazon Aurora MySQL DB クラスターのレプリケーション
- Aurora リードレプリカを使用した、RDS for MySQL DB インスタンスから Amazon Aurora MySQL DB クラスターへのデータの移行

replication_log_file パラメータに指定するファイル名は、出典データベースインスタンスの binlog ファイル名と一致する必要があります。

replication_stop_pointパラメータで指定した停止場所が過去の時点である場合、レプリケーションは即座に停止します。

例

次の例では、レプリケーションをスタートし、120 バイナリログファイルの場所 mysql-bin-changelog.000777 に達するまで変更をレプリケートします。

```
call mysql.rds_start_replication_until(
  'mysql-bin-changelog.000777',
  120);
```

mysql.rds_start_replication_until_gtid (Aurora MySQL バージョン 3)

Aurora MySQL DB クラスターからのレプリケーションを開始し、指定したグローバルトランザクション識別子 (GTID) の後でレプリケーションを停止します。

Syntax

```
CALL mysql.rds_start_replication_until_gtid(gtid);
```

パラメータ

gtid

レプリケーションがその後で停止する GTID。

使用に関する注意事項

マスターユーザーが `mysql.rds_start_replication_until_gtid` を実行する必要があります。

この手順は Aurora MySQL バージョン 3.04 以降でサポートされています。

`mysql.rds_start_replication_until_gtid` ストアドプロシージャは、以下を含むマネージドレプリケーションではサポートされていません。

- AWS リージョン 間での Amazon Aurora MySQL DB クラスターのレプリケーション
- Aurora リードレプリカを使用した、RDS for MySQL DB インスタンスから Amazon Aurora MySQL DB クラスターへのデータの移行

gtidパラメータで指定したトランザクションがレプリカによって既に実行されている場合、レプリケーションは即座に停止します。

例

次の例では、レプリケーションをスタートし、GTID `3E11FA47-71CA-11E1-9E33-C80AA9429562:23` に達するまで変更をレプリケートします。

```
call mysql.rds_start_replication_until_gtid('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

mysql.rds_stop_replication

MySQL DB インスタンスからのレプリケーションを停止します。

Syntax

```
CALL mysql.rds_stop_replication;
```

使用に関する注意事項

マスターユーザーが `mysql.rds_stop_replication` を実行する必要があります。

Amazon RDS の外部で動作する MySQL インスタンスからデータをインポートするようにレプリケーションを設定している場合は、リードレプリカで `mysql.rds_stop_replication` を呼び出して、インポートが完了した後でレプリケーションプロセスを停止することができます。詳細については、「Aurora と MySQL との間、または Aurora と別の Aurora DB クラスターとの間のレプリケーション (バイナリログレプリケーション)」を参照してください。

Amazon RDS の外部にある MySQL インスタンスにデータをエクスポートするようにレプリケーションを設定している場合は、リードレプリカで `mysql.rds_start_replication` と `mysql.rds_stop_replication` を呼び出して、バイナリログの消去などのレプリケーションアクションを制御できます。詳細については、「Aurora と MySQL との間、または Aurora と別の Aurora DB クラスターとの間のレプリケーション (バイナリログレプリケーション)」を参照してください。

`mysql.rds_stop_replication` ストアドプロシージャは、以下を含むマネージドレプリケーションではサポートされていません。

- AWS リージョン 間での Amazon Aurora MySQL DB クラスターのレプリケーション
- Aurora リードレプリカを使用した、RDS for MySQL DB インスタンスから Amazon Aurora MySQL DB クラスターへのデータの移行

Aurora MySQL — 固有の information_schema テーブル

Aurora MySQL には、Aurora に固有の特定の `information_schema` テーブルがあります。

information_schema.aurora_global_db_instance_status

`information_schema.aurora_global_db_instance_status` テーブルには、グローバルデータベースのプライマリ DB クラスターとセカンダリ DB クラスター内のすべての DB インスタンスの

ステータスに関する情報が含まれています。使用できる列を次の表に示します。残りの列は Aurora 内部でのみ使用されます。

> **Note**
>
> この情報スキーマテーブルは、Aurora MySQL バージョン 3.04.0 以降のグローバルデータベースでのみ使用できます。

Column	データ型	説明
SERVER_ID	varchar(100)	DB インスタンスの ID。
SESSION_ID	varchar(100)	現在のセッションの一意識別子。`MASTER_SESSION_ID` の値は、Writer (プライマリ) DB インスタンスを識別します。
AWS_REGION	varchar(100)	このグローバルデータベースインスタンスが実行する AWS リージョン。リージョンのリストについては、「利用可能なリージョン」を参照してください。
DURABLE_LSN	bigint unsigned	ストレージで耐久性のあるログシーケンス番号 (LSN)。ログシーケンス番号 (LSN) は、データベーストランザクションログ内のレコードを識別する一意の連続番号です。LSN は、より大きな LSN が後のトランザクションを表すように順序付けられます。

Column	データ型	説明
HIGHEST_LSN_RCVD	bigint unsigned	ライター DB インスタンスから DB インスタンスが受信した最も高い LSN。
OLDEST_READ_VIEW_TRX_ID	bigint unsigned	ライター DB インスタンスがパージできる最も古いトランザクションの ID。
OLDEST_READ_VIEW_LSN	bigint unsigned	ストレージから読み取るために DB インスタンスが使用した最も古い LSN。
VISIBILITY_LAG_IN_MSEC	float(10,0) unsigned	プライマリ DB クラスターのリーダーの場合、この DB インスタンスがライター DB インスタンスよりどれだけ遅れているか (ミリ秒単位)。セカンダリ DB クラスターのリーダーの場合、この DB インスタンスがセカンダリボリュームからどれだけ遅れているかをミリ秒単位で示します。

information_schema.aurora_global_db_status

`information_schema.aurora_global_db_status` テーブルには、Aurora グローバルデータベースの遅延のさまざまな側面に関する情報が含まれています。具体的には、基盤となる Aurora ストレージの遅延 (いわゆる耐久性の遅延) と目標復旧時点 (RPO) 間の遅延などです。使用できる列を次の表に示します。残りの列は Aurora 内部でのみ使用されます。

> **Note**
>
> この情報スキーマテーブルは、Aurora MySQL バージョン 3.04.0 以降のグローバルデータベースでのみ使用できます。

Column	データ型	説明
AWS_REGION	varchar(100)	このグローバルデータベースインスタンスが実行する AWS リージョン。リージョンのリストについては、「利用可能なリージョン」を参照してください。
HIGHEST_LSN_WRITTEN	bigint unsigned	この DB クラスターに現在存在するログシーケンス番号 (LSN) の最大値。ログシーケンス番号 (LSN) は、データベーストランザクションログ内のレコードを識別する一意の連続番号です。LSN は、より大きな LSN が後のトランザクションを表すように順序付けられます。
DURABILITY_LAG_IN_MILLISECONDS	float(10,0) unsigned	セカンダリ DB クラスターの `HIGHEST_LSN_WRITTEN` とプライマリ DB クラスターの `HIGHEST_LSN_WRITTEN` とのタイムスタンプ値の差。この値は、Aurora グローバルデータベースのプライマリ DB クラスターでは常に 0 です。
RPO_LAG_IN_MILLISECONDS	float(10,0) unsigned	目標復旧時点 (RPO) の遅延。RPO 遅延とは、最近のユーザートランザクション COMMIT が、Aurora グローバルデータベースのプライマリ DB クラスターに保存された後、セカンダリ DB クラ

Column	データ型	説明
		スターに保存されるまでにかかる時間です。この値は、Aurora グローバルデータベースのプライマリ DB クラスターでは常に 0 です。 簡単に言うと、このメトリクスは、Aurora グローバルデータベース内の各 Aurora MySQL DB クラスターの目標復旧時点、つまり、障害が発生した場合に失われる可能性のあるデータの量を計算します。遅延と同様に、RPO は時間単位で測定されます。
LAST_LAG_CALCULATION_TIMESTAMP	datetime	`DURABILITY_LAG_IN_MILLISECONDS` と `RPO_LAG_IN_MILLISECONDS` について値が最後に計算された時刻を指定するタイムスタンプ。`1970-01-01 00:00:00+00` のような時間値は、これがプライマリ DB クラスターであることを意味します。
OLDEST_READ_VIEW_TRX_ID	bigint unsigned	ライター DB インスタンスがパージできる最も古いトランザクションの ID。

information_schema.replica_host_status

`information_schema.replica_host_status` テーブルにはレプリケーション情報が含まれています。使用できる列を以下のテーブルに示します。残りの列は Aurora 内部でのみ使用されます。

Column	データ型	説明
CPU	double	レプリカホストの CPU 使用率。
IS_CURRENT	tinyint	レプリカが最新かどうか。
LAST_UPDATE_TIMESTAMP	datetime(6)	前回の更新が行われた時刻。レコードが古くなっているかどうかを判断するために使用されます。
REPLICA_LAG_IN_MILLISECONDS	double	ミリ秒単位でのレプリカ遅延。
SERVER_ID	varchar(100)	データベースサーバーの ID。
SESSION_ID	varchar(100)	データベースセッションの ID。DB インスタンスがライターインスタンスかリーダーインスタンスかを判断するために使用されます。

> **Note**
>
> レプリカインスタンスが遅延すると、その `information_schema.replica_host_status` テーブルからクエリされる情報が古くなることがあります。このような状況では、代わりにライターインスタンスからクエリを実行することをお勧めします。
> `mysql.ro_replica_status` テーブルにも同様の情報がありますが、使用することは推奨されません。

information_schema.aurora_forwarding_processlist

`information_schema.aurora_forwarding_processlist` テーブルには、書き込み転送に関係するプロセスについての情報が含まれています。

このテーブルの内容は、グローバルまたはクラスター内書き込み転送がオンになっている DB クラスターのライター DB インスタンスでのみ表示されます。リーダー DB インスタンスでは、空の結果セットが返されます。

フィールド	データ型	説明
ID	bigint	ライター DB インスタンス上の接続の識別子。この識別子は、`SHOW PROCESSLIST` ステートメントの `Id` 列に表示され、スレッド内の `CONNECTION_ID()` 関数によって返される値と同じです。
USER	varCHAR(32)	ステートメントを発行した MySQL ユーザー。
HOST	varCHAR(255)	ステートメントを発行した MySQL クライアント。転送されたステートメントの場合、このフィールドには、転送側リーダー DB インスタンスで接続を確立したアプリケーションクライアントのホストアドレスが表示されます。
DB	varCHAR(64)	スレッドのデフォルトデータベース。
コマンド	varCHAR(16)	スレッドがクライアントに代わって実行しているコマンドのタイプ、またはセッションがアイドル状態の場合は `Sleep`。スレッドコマンドの説明については、MySQL ドキュメントの「スレッドコマンド値」を参照してください。
TIME	整数	スレッドが現在の状態になっていた時間 (秒単位)。
STATE	varCHAR(64)	スレッドが何をしているのかを示すアクション、イベント、または状態。状態値の説明については、MySQL ドキュメントの「一般的なスレッドの状態」を参照してください。

フィールド	データ型	説明
INFO	longtext	スレッドが実行しているステートメント、またはステートメントを実行していない場合は NULL。ステートメントは、サーバーに送信されるステートメントかもしれず、ステートメントが他のステートメントを実行する場合は最も内側のステートメントかもしれません。
IS_FORWARDED	bigint	スレッドがリーダー DB インスタンスから転送されるかどうかを示します。
REPLICA_SESSION_ID	bigint	Aurora レプリカの接続識別子。この識別子は、転送側の Aurora リーダー DB インスタンスで SHOW PROCESSLIST ステートメントの Id 列に表示されるのと同じ値です。
REPLICA_INSTANCE_IDENTIFIER	varCHAR(64)	転送側スレッドの DB インスタンス識別子。
REPLICA_CLUSTER_NAME	varCHAR(64)	転送側スレッドの DB クラスター識別子。クラスター内書き込み転送の場合、この識別子は、ライター DB インスタンスと同じ DB クラスターです。
REPLICA_REGION	varCHAR(64)	転送側スレッドの送信元の AWS リージョン。クラスター内書き込み転送の場合、このリージョンは、ライター DB インスタンスと同じ AWS リージョン です。

Amazon Aurora MySQL のデータベースエンジンの更新

Amazon Aurora は定期的に更新をリリースします。更新はシステムメンテナンスの時間中に Aurora DB クラスターに適用されます。更新が適用されるタイミングは、リージョンや DB クラスターのメンテナンスウィンドウの設定、および更新のタイプによって異なります。

Amazon Aurora のリリースは、数日のうちに、すべての AWS リージョンで使用可能になります。一部のリージョンでは、別のリージョンではまだ使用できないエンジンバージョンが一時的に表示されることがあります。

更新は、DB クラスター内のすべてのインスタンスに同時に適用されます。更新では、DB クラスター内のすべてのインスタンスでデータベースを再起動する必要があるため、20 ~ 30 秒のダウンタイムが発生します。その後、DB クラスターの使用を再開できます。AWS Management Console でメンテナンスウィンドウの設定を表示または変更できます。

Amazon Aurora でサポートされている Aurora MySQL バージョンの詳細については、Aurora MySQL のリリースノートを参照してください。

次に、クラスターに適した Aurora MySQL バージョンを選択する方法、クラスターの作成時またはアップグレード時にバージョンを指定する方法、最小限の中断でクラスターを特定のバージョンから別のバージョンにアップグレードする手順を確認できます。

トピック

- Aurora MySQL のバージョン番号と特殊バージョン
- Amazon Aurora MySQL 互換エディションバージョン 2 の標準サポート終了に向けて準備する
- Amazon Aurora MySQL 互換エディションバージョン 1 のサポート終了に向けて準備する
- Amazon Aurora MySQL DB クラスターのアップグレード
- Amazon Aurora MySQL に関するデータベースエンジンの更新と修正

Aurora MySQL のバージョン番号と特殊バージョン

Aurora MySQL 互換エディション は MySQL データベースエンジンと互換性がありますが、Aurora MySQL には Aurora MySQL のバージョン別に固有の機能とバグ修正が含まれています。アプリケーションデベロッパーは、SQL を使用して、アプリケーションの Aurora MySQL バージョンを確認できます。データベース管理者は、Aurora MySQL DB クラスターと DB インスタンスの作成時またはアップグレード時に、Aurora MySQL のバージョンを確認および指定できます。

トピック

AWS による Aurora MySQL エンジンバージョンの確認または指定

AWS Management Console、AWS CLI、あるいは RDS API のいずれかを使用して管理タスクを実行する際に、わかりやすい英数字形式で Aurora MySQL バージョンを指定します。

Aurora MySQL 2 から、Aurora エンジンバージョンの構文は次のとおりです。

```
mysql-major-version.mysql_aurora.aurora-mysql-version
```

mysql-major-version- 部分は 5.7 または 8.0 です。この値は、クライアントプロトコルのバージョンと、対応する Aurora MySQL バージョンでの一般的なレベルの MySQL 機能のサポートを表します。

aurora-mysql-version は、Aurora MySQL メジャーバージョン、Aurora MySQL マイナーバージョン、パッチレベルの 3 つのパートからなるドット付きの値です。メジャーバージョンは 2 または 3 です。これらの値は、MySQL 5.7 または 8.0 と互換性がある Aurora MySQL をそれぞれ表しています。マイナーバージョンは、2.x または 3.x シリーズ内の機能リリースを表します。パッチレベルは、各マイナーバージョンで 0 から始まり、マイナーバージョンに適用される以降の一連のバグ修正を表します。まれに、新しい機能がマイナーバージョンに組み込まれ、すぐには表示されないことがあります。このような場合、この機能はチューニングされ、後のパッチレベルで公開されます。

すべての 2.x Aurora MySQL エンジンバージョンは、コミュニティ MySQL 5.7.12 とワイヤー互換性があります。すべての 3.x Aurora MySQL エンジンバージョンは、MySQL 8.0.23 とワイヤー互換性があります。特定の 3.x バージョンのリリースノートを参照して、対応する MySQL 互換バージョンを検索することができます。

例えば、Aurora MySQL 3.02.0 と 2.11.2 のエンジンバージョンは、次のとおりです。

```
8.0.mysql_aurora.3.02.0
5.7.mysql_aurora.2.11.2
```

> **Note**
>
> コミュニティ MySQL バージョンと Aurora MySQL 2.x バージョンとの間には、1 対 1 の対応はありません。Aurora MySQL バージョン 3 では、より直接的なマッピングがあります。特定の Aurora MySQL リリースにどのバグ修正と新しい機能があるかを確認するには、Aurora MySQL のリリースノートの「Amazon Aurora MySQL バージョン 3 のデータベースエンジンの更新」と「Amazon Aurora MySQL バージョン 2 のデータベースエンジンの更新」を参照してください。新機能とリリースの時系列リストについては、「ドキュメント履歴」を参照してください。セキュリティ関連の修正に必要な最小バージョンを確認するには、Aurora MySQL のリリースノートの「Aurora MySQL で修正されたセキュリティの脆弱性」を参照してください。

Aurora MySQL エンジンバージョンは、AWS CLI コマンドと RDS API オペレーションで指定します。例えば、`--engine-version` オプションは、AWS CLI の create-db-cluster コマンドと modify-db-cluster コマンドを実行するときに指定します。`EngineVersion` パラメータは、RDS API の CreateDBCluster オペレーションと ModifyDBCluster オペレーションを実行するときに指定します。

Aurora MySQL バージョン 2 以降では、AWS Management Console のエンジンバージョンに Aurora バージョンも含まれます。クラスターのアップグレードに伴って、表示される値が変わります。この変更により、クラスターに接続したり、SQL コマンドを実行せずに、Aurora MySQL の正確なバージョンを指定および確認できます。

> **Tip**
>
> AWS CloudFormation を通じて管理される Aurora クラスターの場合、この `EngineVersion` 設定の変更に伴って、AWS CloudFormation によるアクションがトリガーされる場合があります。AWS CloudFormation 設定の変更を `EngineVersion` で処理する方法については、AWS CloudFormation のドキュメントを参照してください。

SQL を使用した Aurora MySQL バージョンの確認

SQL クエリを使用してアプリケーションで取得できる Aurora バージョン番号は、形式として *`<major version>`*`.`*`<minor version>`*`.`*`<patch version>`* を使用します 。`AURORA_VERSION` システム可変をクエリすることで、Aurora MySQL クラスター内の任意の DB インスタンスについて、このバージョン番号を取得できます。このバージョン番号を取得するには、次のいずれかのクエリを使用します。

```
select aurora_version();
select @@aurora_version;
```

これらのクエリは、次のような出力を生成します。

```
mysql> select aurora_version(), @@aurora_version;
+------------------+------------------+
| aurora_version() | @@aurora_version |
+------------------+------------------+
| 2.11.1           | 2.11.1           |
+------------------+------------------+
```

「AWS による Aurora MySQL エンジンバージョンの確認または指定」で説明している手法でコンソール、CLI、および RDS API によって返されるバージョン番号は、通常、より分かりやすいものです。

Aurora MySQL 長期サポート (LTS) リリース

新しい Aurora MySQL バージョンは、それぞれ DB クラスターを作成またはアップグレードする際に一定期間使用できます。この期間後は、そのバージョンを使用するためにクラスターをアップグレードする必要があります。サポート期間終了の際にクラスターを手動でアップグレード、または Aurora MySQL バージョンがサポートされなくなると同時に Aurora は自動的にアップグレードできます。

Aurora は、特定の Aurora MySQL バージョンを長期サポート (LTS) のリリースとして指定します。LTS リリースを使用する DB クラスターは、非 LTS リリースを使用するクラスターよりも同じバージョンに長くとどまるので、アップグレードサイクルの数は少なくなります。Aurora は、そのリリースが利用可能になった後、少なくとも 3 年間は各 LTS のリリースをサポートします。LTS リリースにある DB クラスターをアップグレードする必要がある場合、Aurora は次の LTS リリースにアップグレードされます。これで、クラスターを長時間再度アップグレードする必要はありません。

Aurora MySQL LTS リリースの存続期間中、新しいパッチレベルは重要な問題の修正を導入します。パッチレベルには、新しい機能は含まれていません。LTS リリースを実行している DB クラスターに、このようなパッチを適用するかどうかを選択できます。特定の重要な修正については、Amazon は同じ LTS リリース内のパッチレベルへのマネージドアップグレードを実行する場合があります。そのようなマネージドアップグレードは、クラスターのメンテナンスウィンドウで自動的に実行されます。

ほとんど Aurora MySQL クラスターでは、LTS リリースを使用する代わりに、最新リリースへのアップグレードを推奨します。これにより、マネージドサービスとして Aurora を利用して最新の機

能とバグ修正にアクセスできます。LTS リリースは、次の特性を持つクラスターを対象としています。

- 重要なパッチの稀な問題以外のアップグレードのために、Aurora MySQL アプリケーションのダウンタイムに対応する余裕はありません。
- クラスターおよび関連アプリケーションのテストサイクルは、Aurora MySQL データベースエンジンの更新ごとに時間がかかります。
- Aurora MySQL クラスターのデータベースバージョンには、アプリケーションに必要なすべての DB エンジン機能とバグ修正が含まれています。

Aurora MySQL の現在の LTS リリースは以下のとおりです。

- Aurora MySQL バージョン 3.04.* LTS バージョンの詳細については、「Aurora MySQL のリリースノート」の「Amazon Aurora MySQL バージョン 3 のデータベースエンジンのアップデート」を参照してください。

> **Note**
>
> LTS バージョンの `AutoMinorVersionUpgrade` パラメータを `true` に設定しない (または AWS Management Console の [マイナーバージョン自動アップグレード] を有効にしない) ことをお勧めします。このような設定を行うと、DB クラスターが 3.05.2 などの非 LTS バージョンにアップグレードされる可能性があります。

Aurora MySQL ベータリリース

Aurora MySQL ベータリリースは、限定された数の AWS リージョン でのセキュリティ修正限定の早期リリースです。これらの修正は、次回のパッチリリースで対象範囲を広げ、すべてのリージョンにデプロイされる予定です。

ベータリリースの番号付けは、Aurora MySQL マイナーバージョンと似ていますが、2.12.0.1 や 3.05.0.1 など、4 桁目があります。

詳細については、Aurora MySQL のリリースノートの「Amazon Aurora MySQL バージョン 2 データベースエンジンの更新」と「Amazon Aurora MySQL バージョン 3 のデータベースエンジンの更新」を参照してください。

Amazon Aurora MySQL 互換エディションバージョン 2 の標準サポート終了に向けて準備する

Amazon Aurora MySQL 互換エディション バージョン 2 (MySQL 5.7 互換) は 2024 年 10 月 31 日に標準サポートを終了する予定です。Aurora MySQL バージョン 2 の標準サポート期間が終了する前に、Aurora MySQL バージョン 2 を実行しているすべてのクラスターをデフォルトの Aurora MySQL バージョン 3 (MySQL 8.0 互換) 以降にアップグレードすることをお勧めしています。2024 年 10 月 31 日、Amazon RDS はデータベースを Amazon RDS 延長サポートに自動的に登録します。これは、Aurora Serverless バージョン 1 クラスターで Amazon Aurora MySQL バージョン 2 (MySQL 5.7 互換) を実行している場合には適用されません。Aurora Serverless バージョン 1 のクラスターを Aurora MySQL バージョン 3 にアップグレードする場合は、「Aurora Serverless v1 DB クラスターのアップグレードパス」を参照してください。

Aurora メジャーバージョンの今後のサポート終了日については、「Amazon Aurora バージョン」を参照してください。

Aurora MySQL バージョン 2 を実行しているクラスターをお持ちの場合は、標準サポート期間の終了が近づくと、アップグレードの実施方法に関する最新情報を記載した通知が定期的に届きます。このページは定期的に最新情報で更新されます。

標準サポート終了のタイムライン

1. 現在から 2024 年 10 月 日まで – Aurora MySQL バージョン 1 (MySQL 5.6 互換) クラスターから Aurora MySQL バージョン 2 (MySQL 5.7 互換) または Aurora MySQL 3 (MySQL 8.0 互換) にアップグレードできます。
2. 2024 年 10 月 31 日 — この日をもって、Aurora MySQL バージョン 2 は標準サポートが終了します。Amazon RDS では、クラスターが自動的に Amazon RDS 延長サポートに登録されます。

RDS 延長サポートに自動的に登録されます。詳細については、「Amazon RDS 延長サポートの使用」を参照してください。

この製品終了プロセスの影響を受けるクラスターの確認

この製品終了プロセスの影響を受けるクラスターを確認するには、次の手順を使用します。

> **Important**
>
> これらの手順は、すべての AWS リージョン で、また、リソースが存在しているすべての AWS アカウント で必ず実行してください。

コンソール

Aurora MySQL バージョン 2 クラスターを見つけるには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、データベースを選択します。
3. [データベースによるフィルタ] ボックスで [5.7] と入力します。
4. エンジン列で Aurora MySQL を確認します。

AWS CLI

AWS CLI を使用してこのサポート終了プロセスの影響を受けるクラスターを見つけるには、describe-db-clusters コマンドを呼び出します。次のサンプルスクリプトを使用できます。

Example

```
aws rds describe-db-clusters --include-share --query 'DBClusters[?(Engine==`aurora-
mysql` && contains(EngineVersion,`5.7.mysql_aurora`))].{EngineVersion:EngineVersion,
 DBClusterIdentifier:DBClusterIdentifier, EngineMode:EngineMode}' --output table
 --region us-east-1
```

```
        +-----------------------------------------------------------------+
        |                       DescribeDBClusters                        |
        +---------------------+---------------+-------------------------+
        |         DBCI        |      EM       |          EV             |
        +---------------------+---------------+-------------------------+
        |    aurora-mysql2    |  provisioned  | 5.7.mysql_aurora.2.11.3 |
        | aurora-serverlessv1 |   serverless  | 5.7.mysql_aurora.2.11.3 |
        +---------------------+---------------+-------------------------+
```

RDS API

Aurora MySQL バージョン 2 を実行している Aurora MySQL DB クラスターを見つけるには、RDS DescribeDBClusters API オペレーションを次の必須パラメータとともに使用します。

- `DescribeDBClusters`
  - Filters.Filter.N
    - 名前
      - engine
    - Values.Value.N
      - ['aurora']

Amazon RDS 延長サポート

Amazon RDS 延長サポートは、2024 年 10 月 31 日のサポート終了日まで、コミュニティ MySQL 5.7 を介して無料でご利用いただけます。2024 年 10 月 31 日、Amazon RDS はデータベースを Aurora MySQL バージョン 2 の RDS 延長サポートに自動的に登録します。RDS Aurora 延長サポートは有料サービスで、2027 年 2 月の RDS 延長サポート終了まで、Aurora MySQL バージョン 2 のサポートが最大 28 か月間追加されます。RDS 延長サポートは Aurora MySQL マイナーバージョン 2.11 と 2.12 でのみ提供されます。標準サポート終了後に Amazon Aurora MySQL バージョン 2 を使用するには、2024 年 10 月 31 日までにこれらのマイナーバージョンのいずれかでデータベースを実行することを計画してください。

RDS 延長サポートに関する料金やその他の考慮事項などの詳細については、「Amazon RDS 延長サポートの使用」を参照してください。

アップグレードの実行

メジャーバージョン間のアップグレードでは、マイナーバージョンよりも広範な計画とテストが必要です。このプロセスにはかなりの時間がかかることがあります。アップグレードには、アップグレード前、アップグレード、アップグレード後のアクティビティの 3 段階のプロセスがあります。

アップグレード前:

アップグレード後のアプリケーションが予期した通りに機能するように、アップグレードを実行する前に、アップグレードしたクラスターのアプリケーションの互換性、パフォーマンス、メンテナンス手順、および同様の考慮事項をチェックすることをお勧めします。ここでは、アップグレードをより快適に行うための 5 つの推奨事項を紹介します。

- まず、メジャーバージョンの Aurora MySQL インプレースアップグレードの仕組み を理解することが重要です。
- 次に、Aurora MySQL バージョン 2 からバージョン 3 へのアップグレード で利用できるアップグレード方法を確認します。
- アップグレードの適切な時期とアプローチを決定するために、Aurora MySQL バージョン 3 と Aurora MySQL バージョン 2 と Aurora MySQL バージョン 3 の比較 での現在環境の違いについて学習することができます。
- 使いやすくて最適なオプションを決定したら、Aurora MySQL クラスターのメジャーバージョンアップグレードの計画 を使用して、クローンクラスターで模擬インプレースアップグレードを試してください。プレチェッカーを実行して、データベースを正常にアップグレードできるかどうか、アップグレード後にアプリケーションの非互換性の問題がないか、パフォーマンス、メンテナンス手順、および同様の考慮事項について判断できます。

  アップグレードチェックリストに関するブログのパート 1 とパート 2 を参照してください。
- すべての種類またはバージョンの Aurora MySQL クラスターでインプレースアップグレードメカニズムを使用できるわけではありません。詳細については、「Aurora MySQL メジャーバージョンのアップグレードプロセス」を参照してください。

ご質問やご不明点がございましたら、コミュニティフォーラムやプレミアムサポートから AWS サポートチームにお問い合わせください。

アップグレードの実行:

以下のいずれかのアップグレード手法を使用できます。システムで発生するダウンタイムの長さは、選択した手法によって異なります。

- ブルー/グリーンデプロイ - アプリケーションのダウンタイムの削減が最優先事項である場合は、Amazon RDS ブルー/グリーンデプロイを使用して、プロビジョニングされた Amazon Aurora DB クラスターのメジャーバージョンアップグレードを実行することもできます。ブルー/グリーンのデプロイは、本稼働環境をコピーするステージング環境を作成します。本稼働環境のワークロードに影響を与えずに、グリーン (ステージング) 環境の Aurora DB クラスターに特定の変更を加えることができます。スイッチオーバーには通常 1 分もかからず、データが失われることはありません。詳細については、「Aurora 用 Amazon RDS ブルー/グリーンデプロイの概要」を参照してください。これによりダウンタイムは最小限に抑えられますが、アップグレードの実行中に追加のリソースを実行する必要があります。
- インプレースアップグレード - Aurora が自動的に事前チェックプロセスを実行し、クラスターをオフラインにし、クラスターをバックアップし、アップグレードを実行して、クラスターをオンラ

インに戻すインプレースアップグレードを実行できます。メジャーバージョンのインプレースアップグレードは数回クリックするだけで実行でき、他のクラスターとの調整やフェイルオーバーは必要ありませんが、ダウンタイムは発生します。詳細については、「インプレースアップグレードの実行手順」を参照してください。

- スナップショットと復元 - Aurora MySQL バージョン 2 クラスターのアップグレードは、Aurora MySQL バージョン 2 スナップショットから、Aurora MySQL バージョン 3 クラスターに復元することで実行できます。そのためには、スナップショットを作成し、そこから復元するプロセスに従う必要があります。スナップショットから復元するため、このプロセスにはデータベースの中断が伴います。

アップグレード後:

アップグレード後は、システム (アプリケーションとデータベース) を注意深くモニタリングし、必要に応じて微調整する必要があります。アップグレード前の手順を綿密に実行することで、必要な変更を最小限に抑えることができます。詳細については、「Amazon Aurora MySQL データベースのパフォーマンスのトラブルシューティング」を参照してください。

Aurora MySQL メジャーバージョンのアップグレードの方法、計画、テスト、トラブルシューティングの詳細については、Aurora MySQL インプレースアップグレードのトラブルシューティング を含め「Amazon Aurora MySQL DB クラスターのメジャーバージョンのアップグレード」を精読してください。また、一部のインスタンスタイプは Aurora MySQL バージョン 3 ではサポートされていません。詳細については、「Aurora DB インスタンスクラス」を参照してください。

Aurora Serverless v1 DB クラスターのアップグレードパス

メジャーバージョン間のアップグレードでは、マイナーバージョンよりも広範な計画とテストが必要です。このプロセスにはかなりの時間がかかることがあります。アップグレードには、アップグレード前、アップグレード、アップグレード後のアクティビティの 3 段階のプロセスがあります。

Aurora MySQL バージョン 2 (MySQL 5.7 互換) は、引き続き Aurora Serverless v1 クラスターの標準サポートを受けることができます。

Amazon Aurora MySQL 3 (MySQL 8.0 互換) にアップグレードし、引き続き Aurora Serverless を実行する場合は、Amazon Aurora Serverless v2 を使用することができます。Aurora Serverless v1 と Aurora Serverless v2 の違いを理解するには、「Aurora Serverless v2 と Aurora Serverless v1 の比較」を参照してください。

Aurora Serverless v2 へのアップグレード： Aurora Serverless v1 クラスターを Aurora Serverless v2 にアップグレードできます。詳細については、「Aurora Serverless v1 クラスターから Aurora Serverless v2 クラスターへのアップグレード」を参照してください。

Amazon Aurora MySQL 互換エディションバージョン 1 のサポート終了に向けて準備する

Amazon Aurora MySQL 互換エディション バージョン 1 (MySQL 5.6 互換) は 2023 年 2 月 28 日にサポートを終了する予定です。Aurora MySQL バージョン 1 を実行しているすべてのクラスター (プロビジョニングおよび Aurora Serverless) を Aurora MySQL バージョン 2 (MySQL 5.7 互換) または Aurora MySQL バージョン 3 (MySQL 8.0 互換) にアップグレードすることをお勧めします。Aurora MySQL バージョン 1 のサポート期間が終了する前にこれを行ってください。

Aurora にプロビジョニングされた DB クラスターの場合、いくつかの方法で Aurora MySQL バージョン 1 から Aurora MySQL バージョン 2 へのアップグレードを完了できます。インプレースアップグレードメカニズムの手順については、「インプレースアップグレードの実行手順」を参照してください。アップグレードを完了するもう 1 つの方法は、Aurora MySQL バージョン 1 クラスターのスナップショットを取得し、Aurora MySQL バージョン 2 クラスターにスナップショットを復元することです。または、古いクラスターと新しいクラスターを並べて実行するマルチステッププロセスに従うこともできます。各メソッドの詳細については、「Amazon Aurora MySQL DB クラスターのメジャーバージョンのアップグレード」を参照してください。

Aurora Serverless v1DB クラスターの場合、Aurora MySQL バージョン 1 から Aurora MySQL バージョン 2 へのインプレースアップグレードを実行できます。各メソッドの詳細については、「Aurora Serverless v1 DB クラスターの変更」を参照してください。

Aurora にプロビジョニングされた DB クラスターの場合、2 段階のアップグレードプロセスを使用して Aurora MySQL バージョン 1 から Aurora MySQL バージョン 3 へのアップグレードを完了できます。

1. 前述の方法を使用して Aurora MySQL バージョン 1 から Aurora MySQL バージョン 2 にアップグレードします。
2. Aurora MySQL バージョン 2 から Aurora MySQL バージョン 3 にアップグレードするには、バージョン 1 からバージョン 2 へのアップグレードと同じ方法を使用します。詳細については、「Aurora MySQL バージョン 2 からバージョン 3 へのアップグレード」を参照してください。Aurora MySQL バージョン 2 と 3 の特徴の違い を書き留めます。

Aurora メジャーバージョンの今後の製品終了日については、「Amazon Aurora バージョン」を参照してください。Amazon は、お客様がサポート終了日より前にアップグレードしなかったクラスターを自動的にアップグレードします。製品終了日を過ぎると、後続のメジャーバージョンへのこれらの自動アップグレードは、クラスターのスケジュールされたメンテナンス期間中に行われます。

製品終了となる Aurora MySQL バージョン 1 クラスター (プロビジョニングおよび Aurora Serverless) のアップグレードに関するその他のマイルストーンを次に示します。各日付のスタート時刻は 00:00 協定世界時 (UTC) です。

1. 現在から 2023 年 2 月 28 日まで – Aurora MySQL バージョン 1 (MySQL 5.6 互換) クラスターから Aurora MySQL バージョン 2 (MySQL 5.7 互換) へのアップグレードをいつでも開始できます。Aurora MySQL バージョン 2 から、Aurora にプロビジョニングされた DB クラスターを Aurora MySQL バージョン 3 (MySQL 8.0 互換) にさらにアップグレードできます。
2. 2023 年 1 月 16 日 – これ以降、新しい Aurora MySQL バージョン 1 のクラスターまたはインスタンスを AWS Management Console からも AWS Command Line Interface (AWS CLI) からも作成できなくなります。Aurora Global Database に新しいセカンダリリージョンを追加することもできません。このために、「予期しない停止からの Amazon Aurora Global Database の復旧」で説明されているように、予期しない停止からの復旧が難しくなる可能性があります。この時点以降はステップ 5 と 6 を完了できないからです。また、Aurora MySQL バージョン 1 を実行する新しいクロスリージョンリードレプリカを作成することもできません。2023 年 2 月 28 日までは、既存の Aurora MySQL バージョン 1 クラスターに対して次の操作は実行できます。
   - Aurora MySQL バージョン 1 クラスターのスナップショットを元のスナップショットクラスターと同じバージョンに復元する。
   - リードレプリカの追加 (Aurora Serverless DB クラスターには適用されません)。
   - インスタンス設定を変更する。
   - ポイントインタイム復元を実行する。
   - 既存のバージョン 1 クラスターのクローンを作成する。
   - Aurora MySQL バージョン 2 以上を実行する新しいクロスリージョンリードレプリカを作成します。
3. 2023 年 2 月 28 日 – これ以降、次のスケジュールされたメンテナンス期間内に Aurora MySQL バージョン 1 のクラスターを Aurora MySQL バージョン 2 のデフォルトバージョンに自動的にアップグレードする予定です。Aurora MySQL バージョン 1 DB スナップショットを復元すると、復元されたクラスターが Aurora MySQL バージョン 2 のこの時点のデフォルトバージョンに自動的にアップグレードされます。

メジャーバージョン間のアップグレードでは、マイナーバージョンよりも広範な計画とテストが必要です。このプロセスにはかなりの時間がかかることがあります。

ダウンタイムの削減が最優先事項である場合は、ブルー/グリーンデプロイを使用して、プロビジョニングされた Amazon Aurora DB クラスターのメジャーバージョンアップグレードを実行することもできます。ブルー/グリーンのデプロイは、本稼働環境をコピーするステージング環境を作成します。本稼働環境のワークロードに影響を与えずに、グリーン (ステージング) 環境の Aurora DB クラスターに変更を加えることができます。切り替えには通常 1 分もかからず、データが失われることはなく、アプリケーションを変更する必要もありません。詳細については、「Aurora 用 Amazon RDS ブルー/グリーンデプロイの概要」を参照してください。

アップグレードの完了後に、フォローアップ作業が必要な場合もあります。例えば、SQL の互換性、特定の MySQL 関連機能の動作方法、またはパラメータ設定が古いバージョンと新しいバージョンで異なることが原因でフォローアップが必要になることがあります。

Aurora MySQL メジャーバージョンのアップグレードの方法、計画、テスト、トラブルシューティングの詳細については、「Amazon Aurora MySQL DB クラスターのメジャーバージョンのアップグレード」を精読してください。

この製品終了プロセスの影響を受けるクラスターの確認

この製品終了プロセスの影響を受けるクラスターを確認するには、次の手順を使用します。

> **Important**
>
> これらの手順は、すべての AWS リージョン で、また、リソースが存在しているすべての AWS アカウント で必ず実行してください。

コンソール

Aurora MySQL バージョン 1 クラスターを見つけるには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、データベース を選択します。
3. [Filter by databases] (データベースによるフィルタリング) ボックスで [5.6] と入力します。
4. エンジン列で Aurora MySQL を確認します。

AWS CLI

AWS CLI を使用してこのサポート終了プロセスの影響を受けるクラスターを見つけるには、describe-db-clusters コマンドを呼び出します。次のサンプルスクリプトを使用できます。

Example

```
aws rds describe-db-clusters --include-share --query 'DBClusters[?Engine==`aurora`].
{EV:EngineVersion, DBCI:DBClusterIdentifier, EM:EngineMode}' --output table --region
 us-east-1

        +-----------------------------------------+
        |            DescribeDBClusters           |
        +---------------+--------------+----------+
        |     DBCI      |     EM       |    EV    |
        +---------------+--------------+----------+
        |  my-database-1|  serverless  |  5.6.10a |
        +---------------+--------------+----------+
```

RDS API

Aurora MySQL バージョン 1 を実行している Aurora MySQL DB クラスターを見つけるには、RDS DescribeDBClusters API オペレーションを次の必須パラメータとともに使用します。

- `DescribeDBClusters`
  - Filters.Filter.N
    - 名前
      - engine
    - Values.Value.N
      - ['aurora']

Amazon Aurora MySQL DB クラスターのアップグレード

Aurora MySQL DB クラスターをアップグレードして、バグ修正や新しい Aurora MySQL 機能を取得したり、基盤となるデータベースエンジンを完全に新しいバージョンに変更したりできます。次のセクションで、その方法について説明します。

> **Note**
>
> 実行するアップグレードの種類は、許容できるクラスターのダウンタイム、実施する予定の検証テストの量、ユースケースにおける特定のバグ修正または新機能の重要度によって異なります。また、頻繁に実行予定の小規模なアップグレードなのか、複数の中間のバージョンをスキップする、ときどき実行するアップグレードなのかによっても異なります。アップグレードごとに、クラスターのメジャーバージョン、マイナーバージョン、およびパッチレベルを変更できます。Aurora MySQL メジャーバージョン、マイナーバージョン、パッチレベルの区別に慣れていない場合は、「Aurora MySQL のバージョン番号と特殊バージョン」で背景情報を読むことができます。

> **Tip**
>
> ブルー/グリーンデプロイを使用することで、DB クラスターのアップグレードに必要なダウンタイムを最小限に抑えることができます。詳細については、「データベース更新のために Amazon RDS ブルー/グリーンデプロイを使用する」を参照してください。

トピック

- Aurora MySQL DB クラスターのマイナーバージョンまたはパッチレベルのアップグレード
- Amazon Aurora MySQL DB クラスターのメジャーバージョンのアップグレード

Aurora MySQL DB クラスターのマイナーバージョンまたはパッチレベルのアップグレード

DB クラスターのマイナーバージョンをアップグレードしたり、DB クラスターにパッチを適用したりするには、次の方法を使用できます。

- エンジンのバージョンを変更して Aurora MySQL アップグレードする (Aurora MySQL バージョン 2 および 3)
- Aurora MySQL マイナーバージョン間の自動アップグレードの有効化

ダウンタイムなしのパッチ適用によってアップグレードプロセス中の中断を減らす方法については、「ダウンタイムのないパッチ適用の使用」を参照してください。

マイナーバージョンアップグレードを実行する前に

マイナーバージョンのアップグレード中のダウンタイムを低減するには、次のアクションを実行することをお勧めします。

- Aurora DB クラスターのメンテナンスは、トラフィックが少ない時間帯に実行する必要があります。メンテナンスウィンドウを適切に設定するには、Performance Insights を使用してこのような時間帯を特定します。Performance Insights については、「Amazon RDS での Performance Insights を使用した DB 負荷のモニタリング」を参照してください。DB クラスターのメンテナンスウィンドウの詳細については、「DB クラスターの適切なメンテナンスウィンドウの調整」を参照してください。
- エクスポネンシャルバックオフとジッターをサポートする AWS SDK を使用することが、ベストプラクティスです。詳細については、「エクスポネンシャルバックオフとジッター」を参照してください。

エンジンのバージョンを変更して Aurora MySQL アップグレードする

Aurora MySQL DB クラスターのマイナーバージョンをアップグレードすると、既存のクラスターに追加の修正と新しい機能が適用されます。

このようなアップグレードは、元のバージョンとアップグレード後のバージョンの両方が Aurora MySQL メジャーバージョン (バージョン 2 または バージョン 3) である Aurora MySQL クラスターに適用されます。このプロセスは、Aurora MySQL メタデータの変換やテーブルデータの再編成を必要としないため、迅速で単純明快です。

この種のアップグレードを実行するには、AWS Management Console、AWS CLI、RDS API のいずれかを使用して DB クラスターのエンジンバージョンを変更します。例えば、クラスターで Aurora MySQL 2.x が実行されている場合は、より高い 2.x バージョンを選択します。

Aurora Global Database でマイナーアップグレードを実行する場合は、プライマリクラスターをアップグレードする前に、すべてのセカンダリクラスターをアップグレードします。

> **Note**
>
> Aurora MySQL バージョン 3.03* 以上またはバージョン 2.12.* へのマイナーバージョンアップグレードを実行するには、次のプロセスを使用します。
>
> 1. グローバルクラスターからすべてのセカンダリリージョンを削除します。「Amazon Aurora Global Database からのクラスターの削除」の手順を実行します。

2. 必要に応じて、プライマリリージョンのエンジンバージョンをバージョン 3.03.* 以上、またはバージョン 2.12.* にアップグレードします。「To modify the engine version of a DB cluster」の手順を実行します。
3. グローバルクラスターにセカンダリリージョンを追加します。「AWS リージョン の Amazon Aurora Global Database への追加」の手順を実行します。

DB クラスターのエンジンバージョンを変更するには

- コンソールを使用する場合 -クラスターのプロパティを変更します。[DB クラスターの変更] ウィンドウの [DB エンジンバージョン] ボックスで、Aurora MySQL エンジンバージョンを変更します。クラスターを変更するための一般的な手順に慣れていない場合は、「コンソール、CLI、API を使用した DB クラスターの変更」の手順に従ってください。
- AWS CLI を使用する場合 - AWS CLI コマンドの modify-db-cluster を呼び出し、DB クラスターの名前を `--db-cluster-identifier` オプションで指定しながら、エンジンバージョンを `--engine-version` オプションで指定します。

  例えば、Aurora MySQL バージョン 2.12.1 にアップグレードするには、`--engine-version` オプションを `5.7.mysql_aurora.2.12.1` に設定します。DB クラスターのエンジンバージョンをすぐに更新するには、`--apply-immediately` オプションを指定します。
- RDS API を使用する場合 - ModifyDBCluster API オペレーションを呼び出し、`DBClusterIdentifier` で DB クラスターの名前を指定して `EngineVersion` パラメータでエンジンバージョンを指定します。DB クラスターのエンジンバージョンをすぐに更新するには、`ApplyImmediately` パラメータを `true` に設定します。

Aurora MySQL マイナーバージョン間の自動アップグレードの有効化

Amazon Aurora MySQL DB クラスターの場合、Aurora で DB クラスターを自動的に新しいマイナーバージョンにアップグレードするように指定できます。そのためには、DB クラスターの `AutoMinorVersionUpgrade` プロパティ (AWS Management Console の [マイナーバージョン自動アップグレード]) を設定します。

自動アップグレードはメンテナンスウィンドウ中に実行されます。DB クラスター内の個々の DB インスタンスのメンテナンスウィンドウがクラスターメンテナンスウィンドウと異なる場合は、クラスターメンテナンスウィンドウが優先されます。

マイナーバージョンの自動アップグレードは、次の種類の Aurora MySQL クラスターには適用されません。

- Aurora グローバルデータベースの一部であるクラスター
- クロスリージョンレプリカを持つクラスター

停止時間は、ワークロード、クラスターサイズ、バイナリログデータの量、および Aurora がゼロダウンタイムパッチ適用 (ZDP) 機能を使用できるかどうかによって異なります。Aurora はデータベースクラスターを再起動するため、クラスターの使用を再開する前に、可用性が短時間失われることがあります。特に、バイナリログデータの量が復旧時間に影響を与えます。DB インスタンスは、復旧中にバイナリログデータを処理します。したがって、バイナリログデータが大量である場合、復旧時間が長くなります。

> **Note**
>
> Aurora は、DB クラスター内のすべての DB インスタンスで `AutoMinorVersionUpgrade` 設定が有効になっている場合にのみ、自動アップグレードを実行します。設定方法、およびクラスターレベルとインスタンスレベルで適用した場合にどのように機能するかについては、「Aurora DB クラスターのマイナーバージョン自動アップグレード」を参照してください。
> その後、`AutoUpgrade` が設定されているマイナー DB エンジンへの DB クラスターインスタンスのアップグレードパスが存在する場合、アップグレードが実行されます。`AutoUpgrade` 設定は動的で、RDS によって設定されます。
> マイナーバージョンの自動アップグレードは、デフォルトのマイナーバージョンについて実行されます。

次のような CLI コマンドを使用すると、Aurora MySQL クラスター内のすべての DB インスタンスで `AutoMinorVersionUpgrade` 設定のステータスを確認できます。

```
aws rds describe-db-instances \
  --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVer
```

このコマンドでは、次のような出力が生成されます。

```
[
  {
```

```
        "DBInstanceIdentifier": "db-t2-medium-instance",
        "DBClusterIdentifier": "cluster-57-2020-06-03-6411",
        "AutoMinorVersionUpgrade": true
    },
    {
        "DBInstanceIdentifier": "db-t2-small-original-size",
        "DBClusterIdentifier": "cluster-57-2020-06-03-6411",
        "AutoMinorVersionUpgrade": false
    },
    {
        "DBInstanceIdentifier": "instance-2020-05-01-2332",
        "DBClusterIdentifier": "cluster-57-2020-05-01-4615",
        "AutoMinorVersionUpgrade": true
    },
... output omitted ...
```

この例では、DB クラスター cluster-57-2020-06-03-6411 の [マイナーバージョン自動アップグレードの有効化] がオフになっています。これは、クラスター内の DB インスタンスの 1 つでオフになっているためです。

ダウンタイムのないパッチ適用の使用

Aurora MySQL DB クラスターのアップグレードを実行する場合、データベースがシャットダウンされたときおよびアップグレード中に停止する可能性があります。デフォルトでは、データベースがビジー状態のときにアップグレードをスタートすると、DB クラスターが処理しているすべての接続とトランザクションが失われます。アップグレードを実行するためにデータベースがアイドル状態になるまで待機する場合は、長時間待機しなければならない場合があります。

ダウンタイムのないパッチ適用 (ZDP) 機能では、ベストエフォートに基づいて、Aurora MySQL アップグレード中のクライアント接続を維持するよう試みます。ZDP が正常に完了されると、アプリケーションのセッションが保持され、アップグレードの進行中にデータベースエンジンが再起動します。データベースエンジンの再起動により、数秒から約 1 分間スループットが低下する可能性があります。

ZDP は以下には適用されません。

- オペレーティングシステム (OS) のパッチとアップグレード
- メジャーバージョンのアップグレード

ZDP は、サポートされているすべての Aurora MySQL バージョンと DB インスタンスクラスで使用できます。

ZDP は Aurora Serverless v1 または Aurora グローバルデータベースではサポートされていません。

> **Note**
>
> T DB インスタンスクラスを、開発サーバーおよびテストサーバー、または他の本稼働以外のサーバーにのみ使用することをお勧めします。T インスタンスクラスの詳細については、「開発やテストのための T インスタンスクラスの使用」を参照してください。

MySQL エラーログで ZDP 中に重要な属性のメトリクスを確認できます。AWS Management Console の [イベント] ページでは、Aurora MySQL で ZDP を使用する場合や、ZDP の使用を選択しない場合に関する情報も確認できます。

Aurora MySQL 2.10 以降およびバージョン 3 では、バイナリログレプリケーションが有効かどうかにかかわらず、Aurora はダウンタイムゼロでパッチを実行できます。バイナリログレプリケーションが有効な場合、Aurora MySQL は、ZDP オペレーション中にバイナリログターゲットへの接続を自動的に切断します。Aurora MySQL は自動的にバイナリログターゲットに再接続し、再起動の完了後にレプリケーションを再開します。

また、ZDP は、Aurora MySQL 2.10 以降の再起動の機能拡張と組み合わせて機能させることもできます。ライター DB インスタンスにパッチを適用すると、同時にリーダーにパッチが自動適用されます。パッチの実行後、Aurora は、ライター DB インスタンスとリーダー DB インスタンスの両方で接続を復元します。Aurora MySQL 2.10 より前のバージョンでは、ZDP はクラスターのライター DB インスタンスにのみ適用されます。

ZDP は、以下の状態では正常に完了されない場合があります。

- 長期実行クエリまたはトランザクションが進行中である。この場合、Aurora が ZDP を実行すると、未処理のトランザクションはすべてキャンセルされます。
- データ定義言語 (DDL) ステートメントの実行中などは、一時テーブルまたはテーブルロックが使用中です。この場合、Aurora が ZDP を実行すると、未処理のトランザクションはすべてキャンセルされます。
- パラメータの変更が保留中である。

上記の状態のいずれかにより、ZDP を実行するための適切な時間枠が確保されない場合、パッチ適用はスタンダードの動作に戻ります。

> **Note**
>
> Aurora MySQL バージョン 2 が 2.11.0 より前で、バージョン 3 が 3.04.0 より前の場合、Secure Socket Layer (SSL) 接続または Transport Layer Security (TLS) 接続が開いていると ZDP が正常に完了しない場合があります。

ZDP オペレーションが成功した後も接続はそのまま残りますが、一部の可変と機能は再初期化されます。次の種類の情報は、ダウンタイムのないパッチ適用によって生じる再起動を通じては保持されません。

- グローバル可変 Aurora はセッション可変を復元しますが、再起動後のグローバル可変の復元は行いません。
- ステータス可変。特に、エンジンステータスによって報告される稼働時間の値は、ZDR または ZDP メカニズムを使用する再起動後にリセットされます。
- `LAST_INSERT_ID`.
- テーブルのメモリ内 `auto_increment` 状態。メモリ内自動インクリメント状態が再初期化されます。自動インクリメント値の詳細については、MySQL リファレンスマニュアルを参照してください。
- `INFORMATION_SCHEMA` および `PERFORMANCE_SCHEMA` テーブルからの診断情報。この診断情報は、`SHOW PROFILE` や `SHOW PROFILES` などのコマンドの出力にも表示されます。

ダウンタイムのない再起動に関連する次のアクティビティが [Events] (イベント) ページで報告されます。

- ダウンタイムなしでのデータベースのアップグレードを試みています。
- ダウンタイムなしでのデータベースのアップグレードの試行が終了しました。イベントは、プロセスにかかった時間を報告します。このイベントでは、再起動中に保持された接続の数とドロップされた接続の数も報告されます。データベースエラーログを参照して、再起動中に発生した処理の詳細を確認できます。

代替のブルー/グリーンのアップグレードテクニック

状況によっては、古いクラスターからアップグレードされたクラスターへの即時の切り替えが最優先事項です。このような場合、古いクラスターと新しいクラスターを並べて実行するマルチステッププロセスを使用できます。ここでは、新しいクラスターが引き継ぐ準備ができるまで、古いクラスター

から新しいクラスターにデータをレプリケートします。詳細については、「データベース更新のために Amazon RDS ブルー/グリーンデプロイを使用する」を参照してください。

Amazon Aurora MySQL DB クラスターのメジャーバージョンのアップグレード

2.12.1 などの Aurora MySQL バージョン番号では、2 がメジャーバージョンを表します。Aurora MySQL バージョン 2 は MySQL 5.7 との互換性があります。Aurora MySQL バージョン 3 は MySQL 8.0 との互換性があります。

メジャーバージョン間のアップグレードでは、マイナーバージョンよりも広範な計画とテストが必要です。このプロセスにはかなりの時間がかかることがあります。アップグレードの完了後に、フォローアップ作業が必要な場合もあります。例えば、これは SQL 互換性の違いや、特定の MySQL 関連機能の動作方法の違いが原因で発生する可能性があります。または、古いバージョンと新しいバージョンでパラメータ設定が異なることが原因である可能性があります。

目次

- Aurora MySQL バージョン 3 のアップグレード後のクリーンアップ
  - 空間インデックス

Aurora MySQL バージョン 2 からバージョン 3 へのアップグレード

MySQL 5.7 互換のクラスターがあり、それを MySQL-8.0 互換クラスターにアップグレードする場合は、クラスター自体でアップグレードプロセスを実行します。この種のアップグレードは、新しいクラスターを作成して行うアップグレードとは対照的なインプレースアップグレードです。この手法では、クラスターのエンドポイントやその他の特性を保持します。アップグレードは、すべてのデータを新しいクラスターボリュームにコピーする必要がないため、比較的高速で実行できます。この安定性により、アプリケーションの構成の変更を最小限に抑えることができます。また、アップグレードしたクラスターのテスト量を減らすことができます。これは、DB インスタンスとそのインスタンスクラス数はすべて同じままになるためです。

インプレースアップグレードのメカニズムでは、オペレーションの実行中に DB クラスターをシャットダウンする必要があります。Aurora はクリーンシャットダウンを実行し、トランザクションのロールバックやパージの取り消しなどの未処理のオペレーションを完了します。詳細については、「メジャーバージョンの Aurora MySQL インプレースアップグレードの仕組み」を参照してください。

インプレースアップグレード手順は簡単に実行でき、関連するアプリケーションでの構成の変更を最小限に抑えることができるため有用です。例えば、インプレースアップグレードでは、クラスターのエンドポイントと一連の DB インスタンスが保持されます。ただし、インプレースアップグレードの所要時間は、スキーマのプロパティとクラスターのビジー状態によって異なります。したがって、クラスターのニーズに応じて、複数のアップグレード方法から選択できます。

- インプレースアップグレード
- ブルー/グリーンデプロイ
- スナップショット復元

> **Note**
>
> スナップショット復元のアップグレード手順に AWS CLI または RDS API を使用する場合、後続のオペレーションを実行して、復元された DB クラスターにライター DB インスタンスを作成する必要があります。

Aurora MySQL バージョン 3 および新機能に関する一般的な情報については、「Aurora MySQL バージョン 3 は MYSQL 8.0 との互換性があります。」を参照してください。

アップグレードの計画の詳細については、「Aurora MySQL クラスターのメジャーバージョンアップグレードの計画」と「インプレースアップグレードの実行手順」を参照してください。

Aurora MySQL クラスターのメジャーバージョンアップグレードの計画

アップグレードの適切な時期とアプローチを決定するには、Aurora MySQL バージョン 3 と現在の環境の違いを確認できます。

- RDS for MySQL 8.0 または MySQL 8.0 Community Edition から変換する場合は、「Aurora MySQL バージョン 3 と MySQL 8.0 コミュニティエディションの比較」を参照してください。
- Aurora MySQL バージョン 2、RDS for MySQL 5.7、またはコミュニティ MySQL 5.7 からアップグレードする場合は、「Aurora MySQL バージョン 2 と Aurora MySQL バージョン 3 の比較」を参照してください。
- カスタムパラメータグループの新しい MySQL 8.0 互換バージョンを作成します。必要なカスタムパラメータ値を新しいパラメータグループに適用します。Aurora MySQL バージョン 3 のパラメータ変更 に相談して、パラメータの変更について学びます。
- MySQL 8.0 Community Edition で導入された新しい予約キーワードの使用方法について、Aurora MySQL バージョン 2 のデータベーススキーマとオブジェクト定義を確認してください。アップグレードの前に行ってください。詳細については、MySQL ドキュメントの「MySQL 8.0 の新しいキーワードと予約語」を参照してください。

MySQL 固有のアップグレードに関する考慮事項とヒントについては、MySQL リファレンスマニュアルの MySQL 8.0 での変更を参照してください。例えば、コマンド `mysqlcheck --check-upgrade` を使用して、既存の Aurora MySQL データベースを分析し、潜在的なアップグレードの問題を特定します。

> **Note**
>
> インプレースアップグレードまたはスナップショット復元技術を使用して Aurora MySQL バージョン 3 にアップグレードする場合は、大きな DB インスタンスクラスを使用することをお勧めします。例として、db.r5.24xlarge、db.r6g.16xlarge があります。これにより、DB インスタンスで使用可能な CPU 容量の大部分を使用して、アップグレードプロセスをより早く完了できます。メジャーバージョンのアップグレード完了後、必要な DB インスタンスクラスに変更できます。

アップグレード自体を完了したら、「Aurora MySQL バージョン 3 のアップグレード後のクリーンアップ」のアップグレード後の手順に従います。最後に、アプリケーションの機能とパフォーマンスをテストします。

RDS from MySQL またはコミュニティ MySQL から変換する場合は、「Amazon Aurora MySQL DB クラスターへのデータの移行」で説明している移行手順に従います。場合によっては、バイナリログレプリケーションを使用して、移行の一環として Aurora MySQL バージョン 3 クラスターとデータを同期することがあります。その場合、ソースシステムはターゲット DB クラスターとの互換性があるバージョンを実行する必要があります。

クラスターをメジャーバージョン間でアップグレードした後、アプリケーションと管理手順をスムーズに動作させるには、事前のプランと準備が必要です。AWS CLI スクリプトまたは RDS API ベースのアプリケーションの更新に使用する管理コードの種類については、「インプレースアップグレードはクラスターのパラメータグループにどのような影響を与えるか」を参照してください。また、「Aurora MySQL バージョン間のクラスターのプロパティの変更」も参照してください。

アップグレード中に発生する可能性がある問題については、「Aurora MySQL インプレースアップグレードのトラブルシューティング」を参照してください。アップグレードに長時間を要する可能性がある問題については、これらの条件を事前にテストして修正することができます。

> **Note**
>
> インプレースアップグレードでは、オペレーションの実行中に DB クラスターをシャットダウンする必要があります。Aurora MySQL はクリーンシャットダウンを実行し、UNDO パージなどの未処理のオペレーションを完了します。パージする UNDO レコードが多いと、アップグレードに時間がかかることがあります。履歴リストの長さ (HLL) が短くなった後にのみ、アップグレードを実行することをお勧めします。HLL の一般的な許容値は 100,000 以下です。詳細については、このブログ記事を参照してください。

DB クラスターのクローン作成によるアップグレードのシミュレーション

アップグレードしたクラスターのアプリケーションの互換性、パフォーマンス、メンテナンス手順、および同様の考慮事項を確認できます。そのためには、実際のアップグレードの実行前に、アップグレードのシミュレーションを実行できます。この手法は、本番稼働用クラスターで特に役に立ちます。ここでは、ダウンタイムを最小限に抑え、アップグレードが完了したらすぐにアップグレードされたクラスターを使用できるようにすることが重要です。

以下のステップを使用します。

1. 元のクラスターのクローンを作成します。「Amazon Aurora DB クラスターのボリュームのクローン作成」 の手順に従います。
2. 元のクラスターと同じようなライターおよびリーダー DB インスタンスのセットを設定します。
3. クローンが作成されたクラスターのインプレースアップグレードを実行します。「インプレースアップグレードの実行手順」 の手順に従います。

   クローンを作成したら、すぐにアップグレードをスタートします。これにより、クラスターボリュームは元のクラスターと同じ状態になります。アップグレードの実行前にクローンがアイドル状態になっている場合、Aurora がバックグラウンドでデータベースのクリーンアップ処理を実行します。その場合、クローンのアップグレードは、元のクラスターのアップグレードを正確にシミュレーションしません。
4. クローンが作成されたクラスターを使用して、アプリケーションの互換性、パフォーマンス、管理手順などをテストします。
5. 問題が発生した場合は、それらを考慮してアップグレードの計画を調整してください。例えば、上のバージョンの特徴セットと互換性を持たせるように、任意のアプリケーションコードを適用させます。クラスター内のデータ量を基に、アップグレードにかかる時間を推定します。クラスターがビジーではない時間にアップグレードをスケジュールすることもできます。
6. テストクラスターでアプリケーションとワークロードが適切に動作することが確認できたら、運用クラスターのインプレースアップグレードを実行します。
7. メジャーバージョンのアップグレード中のクラスターの合計ダウンタイムを最小限に抑えます。そのためには、アップグレード時にクラスターのワークロードが低いか、0 であることを確認します。特に、アップグレードのスタート時は、進行中の長時間実行トランザクションがないことを確認してください。

ブルー/グリーンアップグレード手法の使用

古いクラスターと新しいクラスターを並べて実行するブルー/グリーンデプロイを作成することもできます。ここでは、新しいクラスターが引き継ぐ準備ができるまで、古いクラスターから新しいクラスターにデータをレプリケートします。詳細については、「データベース更新のために Amazon RDS ブルー/グリーンデプロイを使用する」を参照してください。

Aurora MySQL のメジャーバージョンアップグレードの事前チェック

MySQL 8.0 には MySQL 5.7 との非互換性がいくつかあります。このような非互換性が原因で MySQL バージョン 2 からバージョン 3 へのアップグレード中に問題が生じる可能性があります。アップグレードを成功させるには、データベースで何らかの準備が必要になる場合があります。

Aurora MySQL バージョン 2 から 3 へのアップグレードをスタートすると、Amazon Aurora では、これらの非互換性を検出するために自動的に事前チェックが実行されます。

これらの事前確認は必須です。スキップすることはできません。事前チェックには次の利点があります。

- アップグレード中の計画外のダウンタイムを回避することができます。
- 非互換性がある場合、Amazon Aurora でアップグレードすることはできません。詳細を示すログが出力されます。ログを使用して、非互換性を排除することにより、データベースをバージョン 3 にアップグレードする準備を行うことができます。非互換性の排除の詳細については、MySQL ドキュメントの「アップグレードのためのインストールの準備」と「MySQL 8.0? へのアップグレード」を参照してください。MySQL Server ブログの「知っておくべきこと」を参照してください。

  MySQL 8.0 へのアップグレードの詳細については、MySQL ドキュメントの「MySQL のアップグレード」を参照してください。

事前チェックには、MySQL に含まれている証明書と Aurora チーム用によって特別に作成された証明書が含まれます。MySQL が提供する事前確認については、Upgrade Checker Utility」を参照してください。

DB インスタンスがアップグレードで停止される前に事前チェックが実行されます。つまり、実行時にダウンタイムが発生することはありません。事前チェックで非互換性が見つかった場合、DB インスタンスが停止する前に、Aurora により自動的にアップグレードがキャンセルされます。Aurora では、非互換性のためのイベントも生成されます。Amazon Aurora イベントの詳細については、「Amazon RDS イベント通知の操作」を参照してください。

Aurora は、ログファイル `PrePatchCompatibility.log` に各非互換性に関する詳細情報を記録します。ほとんどの場合、ログエントリには非互換性を修正するための MySQL のドキュメントへのリンクが含まれています。ログの表示の詳細については、「データベースログファイルの表示とリスト化」を参照してください。

事前チェックの性質上、データベース内のオブジェクトが分析されます。この分析によりリソースが消費され、アップグレードが完了するまでの時間が長くなります。

コミュニティ MySQL アップグレードの事前チェック

以下は、MySQL 5.7 から 8.0 までの非互換性の一般的なリストです。

- MySQL 5.7 互換 DB クラスターでは、MySQL 8.0 でサポートされていない機能を使用しないでください。

詳細については、MySQL ドキュメントの「MySQL 8.0 で削除された機能」を参照してください。

- キーワードや予約語に違反してはいけません。MySQL 8.0 では、以前に予約されていなかったキーワードもあります。

  詳細については、MySQL ドキュメントの「キーワードと予約語」を参照してください。

- Unicode サポートが向上するように、utf8mb3 文字セットを使用するように、utf8mb4 文字セットを使用するオブジェクトを変換することを検討してください。utf8mb3 文字セットは廃止されました。また、utf8mb4 の代わりに utf8 を文字セット参照に使用することを検討してください。現在 utf8 は utf8mb3 文字セットの別名であるためです。

  詳細については、MySQL ドキュメントの「utf8mb3 文字セット (3 バイトの UTF-8 Unicode エンコード)」を参照してください。

- デフォルト以外の行形式の InnoDB テーブルは使用できません。
- ZEROFILL または display の長さ型属性は使用できません。
- ネイティブのパーティショニングをサポートしていないストレージエンジンを使用するパーティショニングされたテーブルがあってはいけません。
- MySQL 5.7 の mysql システムデータベースに、MySQL 8.0 データディクショナリで使用されるテーブルと同じ名前のテーブルがあってはいけません。
- テーブルで古いデータ型や関数を使用してはいけません。
- 64 文字を超える外部キーの制約名があってはいけません。
- sql_mode システム可変設定で、古い SQL モードを定義してはいけません。
- 長さが 255 文字を超える ENUM または SET 列要素をそれぞれ持つテーブルまたはストアドプロシージャが存在してはいけません。
- 共有 InnoDB テーブルスペースに存在するテーブルパーティションが存在してはいけません。
- 表領域データファイルパスに循環参照が存在してはいけません。
- ASC 句に DESC または GROUP BY 修飾子を使用するクエリおよびストアドプログラム定義が存在してはいけません。
- 削除されたシステム変数が存在してはならず、システム変数は MySQL 8.0 の新しいデフォルト値を使用する必要があります。
- ゼロ (0) の日付、日時、タイムスタンプ値は使用できません。
- ファイルの削除または破損によるスキーマの不一致が存在してはいけません。
- FTS 文字列を含むテーブル名が存在してはいけません。

- 別のエンジンに属する InnoDB テーブルが存在してはいけません。
- MySQL 5.7 に無効なテーブル名またはスキーマ名が存在してはいけません。

MySQL 8.0 へのアップグレードの詳細については、MySQL ドキュメントの「MySQL のアップグレード」を参照してください。

Aurora MySQL のアップグレードの事前チェック

バージョン 2 からバージョン 3 にアップグレードする場合、Aurora MySQL には独自の固有要件があります。

- ビュー、ルーチン、トリガー、イベントに、SQL_CACHE、SQL_NO_CACHE、QUERY_CACHE などの非推奨の SQL 構文があってはいけません。
- FTS インデックスのないテーブルには FTS_DOC_ID 列が存在しない必要があります。
- InnoDB データディクショナリと実際のテーブル定義の間に列定義の不一致が存在してはいけません。
- lower_case_table_names パラメータが 1 に設定されている場合は、すべてのデータベース名とテーブル名を小文字にする必要があります。
- イベントとトリガーの definer は欠落していても、空にすることもできず、トリガーに無効な作成コンテキストでもいけません。
- データベース内のすべてのトリガー名は一意である必要があります。
- DDL リカバリと高速 DDL は、Aurora MySQL バージョン 3 ではサポートされていません。これらの機能に関連するデータベースにアーティファクトが存在してはいけません。
- REDUNDANT または COMPACT行形式のテーブルには、767 バイトを超えるインデックスを含めることはできません。
- tiny テキスト列に定義されているインデックスのプレフィックスの長さは 255 バイトを超えることはできません。utf8mb4 文字セットでは、サポートされるプレフィックスの長さは 63 文字に制限されます。

  MySQL 5.7 では、innodb_large_prefix パラメータを使用して、より大きなプレフィックスの長さが許可されました。このパラメータは MySQL 8.0 では廃止されました。
- mysql.host テーブルに InnoDB メタデータの不整合が存在してはいけません。
- システムテーブルに列のデータ型の不一致が存在してはいけません。
- prepared 状態の XA トランザクションが存在してはいけません。
- ビューの列名は 64 文字以下にする必要があります。

- ストアドプロシージャの特殊文字に一貫性を持たせることはできません。
- テーブルにデータファイルパスの不整合が存在してはいけません。

Aurora MySQL メジャーバージョンのアップグレードプロセス

すべての種類またはバージョンの Aurora MySQL クラスターでインプレースアップグレードメカニズムを使用できるわけではありません。次の表を参照して、各 Aurora MySQL クラスターの適切なアップグレードパスを確認してください。

Aurora MySQL DB クラスターのタイプ	インプレースアップグレードを使用できますか?	アクション
Aurora MySQL プロビジョニングされたクラスター、2.0 以降	はい	インプレースアップグレードは、5.7 互換 Aurora MySQL クラスターでサポートされます。 Aurora MySQL バージョン 3 へのアップグレードの詳細については、Aurora MySQL クラスターのメジャーバージョンアップグレードの計画 と インプレースアップグレードの実行手順 を参照してください。
Aurora MySQL プロビジョニングされたクラスター、3.01.0 以降	該当なし	Aurora MySQL バージョン 3 バージョン間でアップグレードするには、マイナーバージョンアップグレード手順を使用してください。
Aurora Serverless v1 クラスター	該当なし	現時点では、Aurora Serverless v1 はバージョン 2 でのみ Aurora MySQL でサポートされています。
Aurora Serverless v2 クラスター	該当なし	現時点では、Aurora Serverless v2 はバージョン 3 でのみ Aurora MySQL でサポートされています。
Aurora Global Database のクラスター	はい	Aurora MySQL をバージョン 2 からバージョン 3 にアップグレードするには、Aurora グローバルデータベースでクラスターのインプレースアップグレードを実行する

Aurora MySQL DB クラスターのタイプ	インプレースアップグレードを使用できますか?	アクション
		手順に従います。グローバルクラスターでアップグレードを実行します。Aurora は、グローバルデータベースのプライマリクラスター、ならびにすべてのセカンダリクラスターに対し、同時に自動アップグレードを実行します。 AWS CLI または RDS API を使用する場合は、`modify-global-cluster` または `ModifyGlobalCluster` の代わりに `modify-db-cluster` コマンドまたは `ModifyDBCluster` オペレーションを呼び出します。 `lower_case_table_names` パラメータがオンの場合、Aurora MySQL バージョン 2 からバージョン 3 へのインプレースアップグレードを実行できません。詳細については、「メジャーバージョンのアップグレード」を参照してください。
パラレルクエリクラスター	はい	インプレースアップグレードを実行できます。この場合、Aurora MySQL バージョンに 2.09.1 以降を選択します。
バイナリログレプリケーションのターゲットのクラスター	おそらく	バイナリログのレプリケーションが Aurora MySQL クラスターからのものであれば、インプレースアップグレードを実行できます。バイナリログのレプリケーションが RDS for MySQL またはオンプレミス MySQL DB インスタンスからのものである場合は、アップグレードを実行できません。その場合は、スナップショットの復元メカニズムを使用してアップグレードします。

Aurora MySQL DB クラスターのタイプ	インプレースアップグレードを使用できますか?	アクション
DB インスタンスがゼロのクラスター	いいえ	AWS CLI または RDS API を使用すると、DB インスタンスをアタッチせずに、 Aurora MySQL クラスターを作成できます。同様に、クラスターボリューム内のデータをそのまま残しつつ、Aurora MySQL クラスターからすべての DB インスタンスを削除することもできます。クラスターに DB インスタンスがない場合、インプレースアップグレードを実行することはできません。 アップグレードメカニズムでは、システムテーブルやデータファイルなどに対して変換を実行するため、クラスターのライターインスタンスが必要です。この場合、AWS CLI または RDS API を使用して、クラスターのライターインスタンスを作成します。その後、インプレースアップグレードを実行します。
バックトラックが有効なクラスター	はい	バックトラック機能を使用する Aurora MySQL クラスターのインプレースアップグレードを実行できます。ただし、アップグレード後は、クラスターをアップグレード前の時間までバックトラックすることはできません。

メジャーバージョンの Aurora MySQL インプレースアップグレードの仕組み

Aurora MySQL は、メジャーバージョンのアップグレードを多段階プロセスとして実行します。アップグレードの現在のステータスを確認できます。アップグレードの一部のステップでは、進捗情報も確認できます。各ステージのスタート時に、Aurora MySQL によってイベントが記録されます。RDS コンソールの [イベント] ページで、発生したイベントを確認できます。イベントの使用の詳細については、「Amazon RDS イベント通知の操作」を参照してください。

> **Important**
>
> プロセスがスタートされると、アップグレードが成功または失敗するまで実行されます。進行中は、アップグレードをキャンセルできません。アップグレードが失敗した場合、Aurora はすべての変更をロールバックし、クラスターのエンジンバージョン、メタデータなどが前と同じ状態になります。

アップグレードプロセスには、3 つのステージがあります。

1. Aurora は、アップグレードプロセスをスタートする前に一連の事前チェックを実行します。Aurora がこれらのチェックを実行している間、クラスターは実行を続けます。例えば、クラスターは、準備済みステートメントの XA トランザクションを使用したり、データ定義言語 (DDL) ステートメントを処理したりすることはできません。例えば、特定の種類の SQL ステートメントを送信中のアプリケーションをシャットダウンする必要がある場合があります。または、長時間実行される特定のステートメントが終了するまで待たなければならない可能性もあります。その場合は、アップグレードを再試行してください。一部のチェックでは、アップグレードの妨げにはならないものの、アップグレードに長時間かかる可能性がある条件をテストします。

   Aurora で必要な条件が満たされていないことが検出された場合は、イベントの詳細に示されている条件を変更します。Aurora MySQL インプレースアップグレードのトラブルシューティング のガイダンスに従います。Aurora で、アップグレードを遅らせる原因となる条件が検出された場合は、長期間にわたりアップグレードをモニタリングすることを計画します。
2. Aurora はクラスターをオフラインにします。次に、Aurora が前のステージと同様の一連のテストを実行し、シャットダウンプロセス中に新たな問題が発生していないことを確認します。この時点で、Aurora がアップグレードの妨げになる条件を検出した場合、Aurora はアップグレードをキャンセルし、クラスターをオンラインに戻します。この場合、条件がいつから適用されていないかを確認し、アップグレードを再度スタートします。
3. Aurora は、クラスターボリュームのスナップショットを作成します。アップグレードの完了後に、互換性やその他の種類の問題を発見したとします。または、元のクラスターとアップグレードされたクラスターの両方を使用してテストを実行するとします。このような場合、このスナップショットから復元し、元のエンジンバージョンと元のデータを使用して新しいクラスターを作成することができます。

> **Tip**
>
> このスナップショットは手動スナップショットです。ただし、Aurora は、手動スナップショットのクォータに達した場合でも、それを作成してアップグレードプロセスを続行することができます。このスナップショットは、削除するまで永続的に残ります (必要な場合)。アップグレード後のテストをすべて完了したら、このスナップショットを削除してストレージ料金を最小限に抑えることができます。

4. クラスターボリュームの Aurora クローンを作成します。クローン作成は、実際のテーブルデータのコピーを伴わない高速オペレーションです。アップグレード中、Aurora に問題が発生すると、クローンが作成されたクラスターボリュームから元のデータに戻り、クラスターをオンラインに戻します。アップグレード中のクローン作成の一時ボリュームは、単一のクラスターボリュームでの通常のクローン数の制限を受けません。
5. Aurora は、ライター DB インスタンスのクリーンシャットダウンを実行します。クリーンシャットダウン中は、以下のオペレーションの進行状況イベントが 15 分ごとに記録されます。RDS コンソールの [イベント] ページで、発生したイベントを確認できます。
   - Aurora は、古いバージョンの行の UNDO レコードを消去します。
   - Aurora は、コミットされていないトランザクションをロールバックします。
6. Aurora は、ライター DB インスタンスのエンジンバージョンをアップグレードします。
   - Aurora は、新しいエンジンバージョンのバイナリをライター DB インスタンスにインストールします。
   - Aurora は、ライター DB インスタンスを使用して、データを MySQL 5.7 互換のフォーマットにアップグレードします。このステージでは、Aurora によってシステムテーブルが変更され、クラスターボリュームのデータに影響を与えるその他の変換が実行されます。特に、Aurora はシステムテーブル内のパーティションのメタデータをアップグレードして、MySQL 5.7 のパーティションのフォーマットとの互換性を持たせます。クラスター内のテーブルに多数のパーティションがある場合、このステージには長時間かかります。

     このステージでエラーが発生した場合は、MySQL エラーログで詳細を確認できます。このステージのスタート後に何らかの理由でアップグレードプロセスが失敗した場合、Aurora はクローンが作成されたクラスターボリュームから元のデータを復元します。
7. Aurora は、リーダー DB インスタンスのエンジンバージョンをアップグレードします。
8. アップグレードプロセスは完了しました。アップグレードプロセスが正常に完了したことを示す最終イベントが、Aurora により記録されます。DB クラスターが新しいメジャーバージョンで実行されています。

代替の Blue/Green のアップグレードテクニック

状況によっては、古いクラスターからアップグレードされたクラスターへの即時の切り替えが最優先事項です。このような場合、古いクラスターと新しいクラスターを並べて実行するマルチステッププロセスを使用できます。ここでは、新しいクラスターが引き継ぐ準備ができるまで、古いクラスターから新しいクラスターにデータをレプリケートします。詳細については、「データベース更新のために Amazon RDS ブルー/グリーンデプロイを使用する」を参照してください。

インプレースアップグレードの実行手順

メジャーバージョンの Aurora MySQL インプレースアップグレードの仕組み の背景のマテリアルを確認することをお勧めします。

「Aurora MySQL クラスターのメジャーバージョンアップグレードの計画」の説明に従い、アップグレード前の計画とテストを行います。

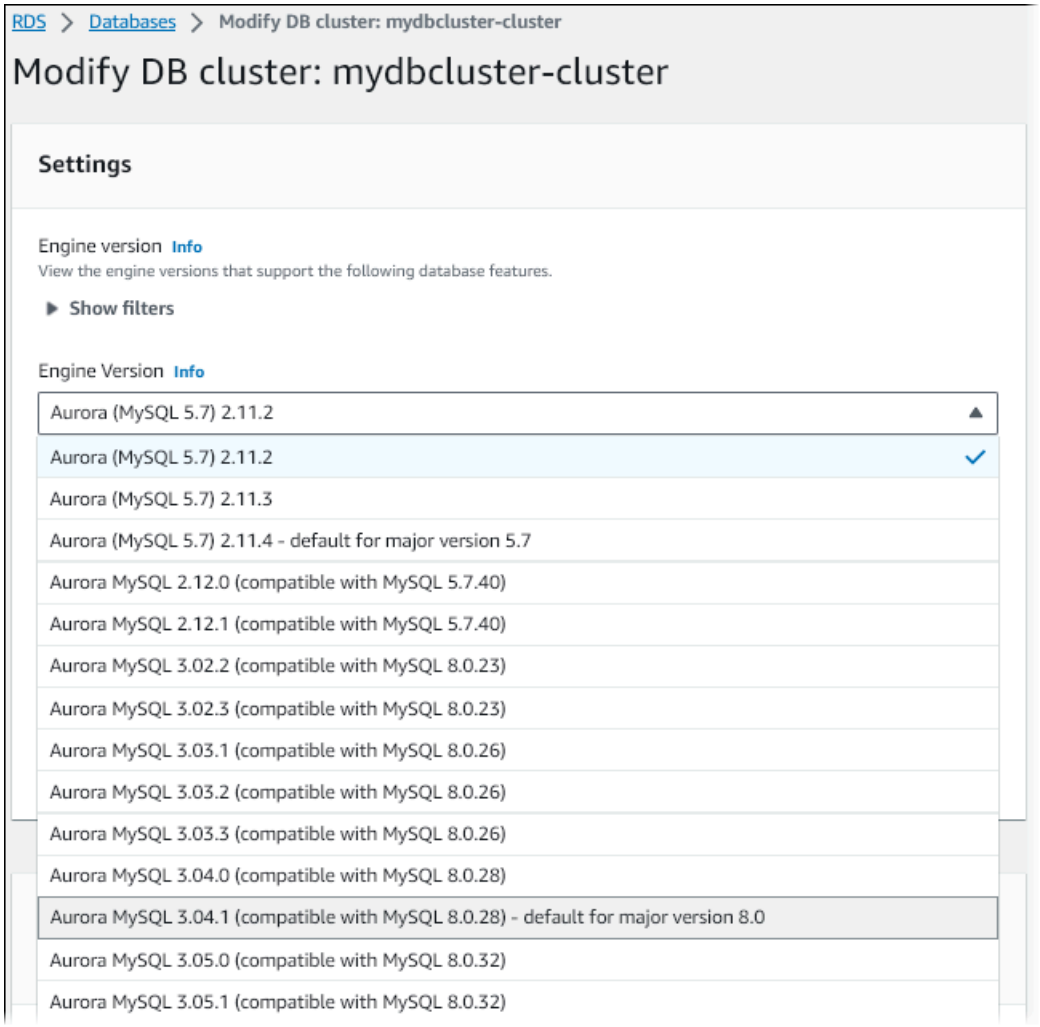
コンソール

次の例では、`mydbcluster-cluster` DB クラスターを Aurora MySQL バージョン 3.04.1 にアップグレードします。

Aurora MySQL DB クラスターのメジャーバージョンをアップグレードするには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. 元の DB クラスターでカスタムパラメータグループを使用した場合は、新しいメジャーバージョン互換のパラメータグループを作成します。新しいパラメータグループの設定パラメータに必要な調整を行います。詳細については、「インプレースアップグレードはクラスターのパラメータグループにどのような影響を与えるか」を参照してください。
3. ナビゲーションペインで、[データベース] を選択します。
4. リストから、変更する DB クラスターを選択します。
5. Modify を選択します。
6. [Version] (バージョン) で、新しい Aurora MySQL のメジャーバージョンを選択します。

   通常、メジャーバージョンの最新のマイナーバージョンを使用することをお勧めします。ここでは、現在のデフォルトバージョンを選択します。



7. Continue (続行) をクリックします。
8. 次のページで、アップグレードを実行するタイミングを指定します。[次の定期メンテナンス期間中] または [今すぐ] を選択します。
9. (オプション) アップグレード中、RDS コンソールの [イベント] ページを定期的に確認します。これにより、アップグレードの進行状況をモニタリングし、問題を特定することができます。アップグレードで問題が発生した場合は、Aurora MySQL インプレースアップグレードのトラブルシューティング を参照してステップを実行してください。
10. この手順のスタート時に、新しいパラメータグループを作成した場合は、アップグレードしたクラスターにカスタムパラメータグループを関連付けます。詳細については、「インプレースアップグレードはクラスターのパラメータグループにどのような影響を与えるか」を参照してください。

> **Note**
>
> このステップを実行するには、クラスターを再起動して新しいパラメータグループを適用する必要があります。

11. (オプション) アップグレード後のテストが完了したら、アップグレードのスタート時に Aurora によって作成された手動スナップショットを削除します。

AWS CLI

Aurora MySQL DB クラスターのメジャーバージョンをアップグレードするには、次の必須パラメータを指定しながら、AWS CLI の modify-db-cluster コマンドを実行します。

- `--db-cluster-identifier`
- `--engine-version`
- `--allow-major-version-upgrade`
- `--apply-immediately` または `--no-apply-immediately`

クラスターでカスタムパラメータグループを使用する場合は、次のオプションの 1 つ、または両方を含めます。

- `--db-cluster-parameter-group-name`、クラスターがカスタムクラスターのパラメータグループを使用している場合
- `--db-instance-parameter-group-name`、クラスター内のインスタンスがカスタム DB のパラメータグループを使用している場合

次の例では、`sample-cluster` DB クラスターを Aurora MySQL バージョン 3.04.1 にアップグレードします。アップグレードは、次のメンテナンスウィンドウを待つことなく、すぐに実行されます。

Example

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster \
          --db-cluster-identifier sample-cluster \
          --engine-version 8.0.mysql_aurora.3.04.1 \
          --allow-major-version-upgrade \
```

```
          --apply-immediately
```

Windows の場合:

```
aws rds modify-db-cluster ^
          --db-cluster-identifier sample-cluster ^
          --engine-version 8.0.mysql_aurora.3.04.1 ^
          --allow-major-version-upgrade ^
          --apply-immediately
```

他の CLI `modify-db-cluster` コマンドをと組み合わせて、アップグレードを実行および検証するための自動化されたエンドツーエンドのプロセスを作成できます。詳細な説明と例については、「Aurora MySQL インプレースアップグレードのチュートリアル」を参照してください。

> **Note**
>
> クラスターが Aurora Global Database の一部である場合、インプレースアップグレードの手順は若干異なります。`modify-db-cluster` の代わりに、modify-global-cluster コマンドオペレーションを呼び出します。詳細については、「グローバルデータベースのインプレースメジャーアップグレード」を参照してください。

RDS API

Aurora MySQL DB クラスターのメジャーバージョンをアップグレードするには、次の必須パラメータを指定して RDS API の ModifyDBCluster オペレーションを使用します。

- `DBClusterIdentifier`
- `Engine`
- `EngineVersion`
- `AllowMajorVersionUpgrade`
- `ApplyImmediately` (`true`、または `false` に設定)

> **Note**
>
> クラスターが Aurora Global Database の一部である場合、インプレースアップグレードの手順は若干異なります。 `ModifyDBCluster` の代わりに、 modifyGlobalCluster オペレーショ

ンを呼び出します。詳細については、「グローバルデータベースのインプレースメジャーアップグレード」を参照してください。

インプレースアップグレードはクラスターのパラメータグループにどのような影響を与えるか

Aurora パラメータグループには、MySQL 5.7 または 8.0 と互換性のあるクラスターとは異なる構成設定のセットがあります。インプレースアップグレードを実行する際、アップグレードされたクラスターとそのすべてのインスタンスで、対応するクラスターおよびインスタンスのパラメータグループを使用する必要があります。

クラスターとインスタンスは、デフォルトの 5.7 互換パラメータグループを使用する場合があります。その場合、アップグレードされたクラスターとインスタンスは、デフォルトの 8.0 互換パラメータグループで開始されます。クラスターとインスタンスでカスタムパラメータグループを使用している場合は、対応する、または 8.0 互換のパラメータグループを作成してください。また、アップグレード処理中に必ずそれらを指定してください。

Note

ほとんどのパラメータ設定では、2 つのポイントでカスタムパラメータグループを選択できます。これらは、クラスターの作成時や、パラメータグループをクラスターに関連付ける場合です。
ただし、デフォルト以外の設定を `lower_case_table_names` パラメータに使用する場合は、事前にこの設定を使用してカスタム パラメータグループを設定する必要があります。その後、スナップショット復元を実行してクラスターを作成する際、パラメータグループを指定します。クラスター作成後の`lower_case_table_names` パラメータへの変更は、影響がありません。
Aurora MySQL バージョン 2 からバージョン 3 にアップグレードする場合にも、同じ `lower_case_table_names` の設定を使用することをお勧めします。
Aurora MySQL に基づく Aurora グローバルデータベースでは、`lower_case_table_names` パラメータがオンの場合、Aurora MySQL バージョン 2 からバージョン 3 へのインプレースアップグレードを実行できません。使用できる方法の詳細については、「メジャーバージョンのアップグレード」を参照してください。

> **Important**
>
> アップグレードプロセス中にカスタムパラメータグループを指定した場合は、アップグレード終了後にクラスターを手動で再起動してください。再起動すると、クラスターがカスタムパラメータ設定の使用をスタートできます。

Aurora MySQL バージョン間のクラスターのプロパティの変更

Aurora MySQL バージョン 2 からバージョン 3 にアップグレードする場合は、Aurora MySQL クラスターと DB インスタンスのセットアップと管理に使用するアプリケーションやスクリプトを確認してください。

また、5.7 および 8.0 互換クラスターではデフォルトのパラメータグループ名が異なることを考慮して、パラメータグループを操作するコードを変更します。Aurora MySQL バージョン 2 と 3 のクラスターのデフォルトのパラメータグループ名は、それぞれ `default.aurora-mysql5.7` と `default.aurora-mysql8.0` です。

例えば、アップグレード前にクラスターに適用される次のようなコードがあるとします。

```
# Check the default parameter values for MySQL 5.7-compatible clusters.
aws rds describe-db-parameters --db-parameter-group-name default.aurora-mysql5.7 --
region us-east-1
```

クラスターのメジャーバージョンをアップグレードした後、そのコードを次のように変更します。

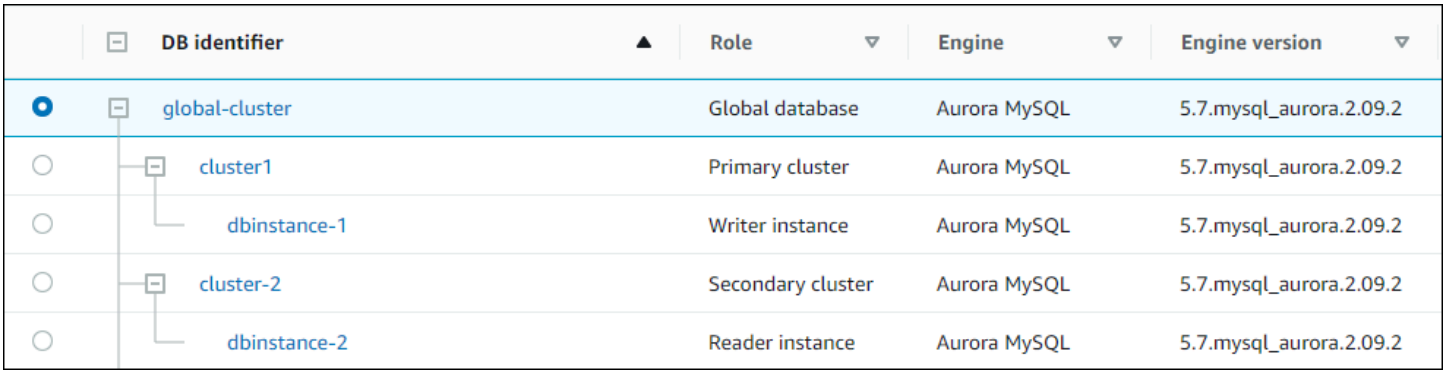
```
# Check the default parameter values for MySQL 8.0-compatible clusters.
aws rds describe-db-parameters --db-parameter-group-name default.aurora-mysql8.0 --
region us-east-1
```

グローバルデータベースのインプレースメジャーアップグレード

Aurora Global Database の場合は、グローバルデータベースクラスターをアップグレードします。Aurora は、すべてのクラスターを同時かつ自動的にアップグレードし、すべてのクラスターで、同じエンジンバージョンが実行されることを保証します。これは、システムテーブルやデータファイル形式などの変更が、すべてのセカンダリクラスターに自動的にレプリケートされるためです。

「メジャーバージョンの Aurora MySQL インプレースアップグレードの仕組み」の手順に従います アップグレードする対象を指定するときは、そのデータベースに含まれるクラスターの 1 つではなく、グローバルデータベースクラスターを選択してください。

AWS Management Console を使用する場合、[Global database] (グローバルデータベース) のロールを持つアイテムを選択します。



DB identifier	Role	Engine	Engine version
global-cluster	Global database	Aurora MySQL	5.7.mysql_aurora.2.09.2
cluster1	Primary cluster	Aurora MySQL	5.7.mysql_aurora.2.09.2
dbinstance-1	Writer instance	Aurora MySQL	5.7.mysql_aurora.2.09.2
cluster-2	Secondary cluster	Aurora MySQL	5.7.mysql_aurora.2.09.2
dbinstance-2	Reader instance	Aurora MySQL	5.7.mysql_aurora.2.09.2

AWS CLI または RDS API を使用する場合は、modify-global-cluster コマンドまたは ModifyGlobalCluster オペレーションを呼び出して、アップグレードプロセスをスタートします。`modify-db-cluster` または `ModifyDBCluster` の代わりにこれらのいずれかを使用します。

> **Note**
>
> Aurora グローバルデータベースのメジャーバージョンアップグレードを実行している間、グローバルデータベースクラスターのカスタムパラメータグループを指定することはできません。グローバルクラスターの各リージョンにカスタムパラメータグループを作成します。次に、アップグレード後に手動でリージョンクラスターに適用します。

AWS CLI を使用して Aurora MySQL グローバルデータベースクラスターのメジャーバージョンをアップグレードするには、以下の必須パラメータを指定して、modify-global-cluster コマンドを使用します。

- `--global-cluster-identifier`
- `--engine aurora-mysql`
- `--engine-version`
- `--allow-major-version-upgrade`

次の例では、グローバルデータベースクラスターを Aurora MySQL バージョン 2.10.2 にアップグレードします。

Example

Linux、macOS、Unix の場合:

```
aws rds modify-global-cluster \
          --global-cluster-identifier global_cluster_identifier \
          --engine aurora-mysql \
          --engine-version 5.7.mysql_aurora.2.10.2 \
          --allow-major-version-upgrade
```

Windows の場合:

```
aws rds modify-global-cluster ^
          --global-cluster-identifier global_cluster_identifier ^
          --engine aurora-mysql ^
          --engine-version 5.7.mysql_aurora.2.10.2 ^
          --allow-major-version-upgrade
```

バックトラックに関する考慮事項

アップグレードしたクラスターでバックトラック機能が有効になっている場合、アップグレードしたクラスターをアップグレード前の時間にバックトラックすることはできません。

Aurora MySQL インプレースアップグレードのチュートリアル

次の Linux の例では、AWS CLI を使用してインプレースアップグレードの一般的なステップを実行する方法を示します。

この最初の例では、2.x バージョンの Aurora MySQL を実行する Aurora DB クラスターを作成します。クラスターには、ライター DB インスタンスとリーダー DB インスタンスが含まれます。`wait db-instance-available` コマンドは、ライター DB インスタンスが利用可能になるまで一時停止します。クラスターをアップグレードする準備ができた時点です。

```
aws rds create-db-cluster --db-cluster-identifier mynewdbcluster --engine aurora-mysql
 \
  --db-cluster-version 5.7.mysql_aurora.2.10.2
...
```

```
aws rds create-db-instance --db-instance-identifier mynewdbcluster-instance1 \
  --db-cluster-identifier mynewdbcluster --db-instance-class db.t4g.medium --engine
 aurora-mysql
...
aws rds wait db-instance-available --db-instance-identifier mynewdbcluster-instance1
```

クラスターをアップグレードできる Aurora MySQL 3.x バージョンは、クラスターが現在実行している 2.x バージョンと、クラスターがある AWS リージョン によって異なります。`--output text` による初期のコマンドは、使用可能なターゲットバージョンだけを表示させます。2 番目のコマンドは、レスポンスの完全な JSON 出力を表示します。その応答では、`engine` パラメータに使用した `aurora-mysql` 値などの詳細を確認できます。また、3.02.0 へのアップグレードがメジャーバージョンアップグレードであることがわかります。

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].{EngineVersion:EngineVersion}' --output text
5.7.mysql_aurora.2.10.2

aws rds describe-db-engine-versions --engine aurora-mysql --engine-version
 5.7.mysql_aurora.2.10.2 \
  --query '*[].[ValidUpgradeTarget]'
...
{
    "Engine": "aurora-mysql",
    "EngineVersion": "8.0.mysql_aurora.3.02.0",
    "Description": "Aurora MySQL 3.02.0 (compatible with MySQL 8.0.23)",
    "AutoUpgrade": false,
    "IsMajorVersionUpgrade": true,
    "SupportedEngineModes": [
        "provisioned"
    ],
    "SupportsParallelQuery": true,
    "SupportsGlobalDatabases": true,
    "SupportsBabelfish": false
},
...
```

この例では、クラスターの有効なアップグレードターゲットではないターゲットバージョン番号が入力された場合に、Aurora がアップグレードを実行しないようにする方法を示しています。Aurora は、`--allow-major-version-upgrade` パラメータを指定しない限り、メジャーバージョンのアップグレードを実行しません。これにより、アプリケーションコードの大規模なテストや変更を必要とする可能性のあるアップグレードを誤って実行することはありません。

```
aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 5.7.mysql_aurora.2.09.2 --apply-immediately
An error occurred (InvalidParameterCombination) when calling the ModifyDBCluster
 operation: Cannot find upgrade target from 5.7.mysql_aurora.2.10.2 with requested
 version 5.7.mysql_aurora.2.09.2.

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 8.0.mysql_aurora.3.02.0 --region us-east-1 --apply-immediately
An error occurred (InvalidParameterCombination) when calling the ModifyDBCluster
 operation: The AllowMajorVersionUpgrade flag must be present when upgrading to a new
 major version.

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 8.0.mysql_aurora.3.02.0 --apply-immediately --allow-major-version-
upgrade
{
  "DBClusterIdentifier": "mynewdbcluster",
  "Status": "available",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.10.2"
}
```

クラスターおよび関連付けられた DB インスタンスのステータスが upgrading に変わるまで、しばらく時間がかかります。クラスターおよび DB インスタンスのバージョン番号は、アップグレードが完了したときにのみ変更されます。この場合も、ライター DB インスタンスの `wait db-instance-available` コマンドを使用して、アップグレードが完了するまで待機してから続行します。

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].[Status,EngineVersion]' --output text
upgrading 5.7.mysql_aurora.2.10.2

aws rds describe-db-instances --db-instance-identifier mynewdbcluster-instance1 \
  --query '*[].
{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceStatus:DBInstanceStatus} | [0]'
{
    "DBInstanceIdentifier": "mynewdbcluster-instance1",
    "DBInstanceStatus": "upgrading"
}

aws rds wait db-instance-available --db-instance-identifier mynewdbcluster-instance1
```

この時点で、クラスターのバージョン番号が、アップグレード用に指定されたバージョン番号と一致します。

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].[EngineVersion]' --output text

8.0.mysql_aurora.3.02.0
```

前の例では、--apply-immediately パラメータを指定して即時のアップグレードを実行しました。クラスターがビジー状態ではないと予想される都合の良いときにアップグレードを実行するために、--no-apply-immediately パラメータを指定できます。これにより、クラスターの次のメンテナンスウィンドウの表示中にアップグレードがスタートされます。メンテナンスウィンドウでは、メンテナンスオペレーションをスタートできる期間を定義します。長時間実行されるオペレーションは、メンテナンスウィンドウの中で終了しない場合があります。したがって、アップグレードに長時間かかることが予想される場合でも、より大きなメンテナンスウィンドウを定義する必要はありません。

次の例では、初期に Aurora MySQL バージョン 2.10.2 を実行しているクラスターをアップグレードします。describe-db-engine-versions 出力では、False および True 値は IsMajorVersionUpgrade プロパティを表します。バージョン 2.10.2 から、他の 2.* バージョンにアップグレードできます。これらのアップグレードはメジャーバージョンアップグレードとはみなされないため、一括アップグレードは必要ありません。インプレースアップグレードは、リストに示されている 3.* バージョンへのアップグレードでのみ利用できます。

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].{EngineVersion:EngineVersion}' --output text
5.7.mysql_aurora.2.10.2

aws rds describe-db-engine-versions --engine aurora-mysql --engine-version
 5.7.mysql_aurora.2.10.2 \
  --query '*[].[ValidUpgradeTarget]|[0][0]|[*].[EngineVersion,IsMajorVersionUpgrade]'
 --output text

5.7.mysql_aurora.2.10.3 False
5.7.mysql_aurora.2.11.0 False
5.7.mysql_aurora.2.11.1 False
8.0.mysql_aurora.3.01.1 True
8.0.mysql_aurora.3.02.0 True
8.0.mysql_aurora.3.02.2 True
```

```
aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 8.0.mysql_aurora.3.02.0 --no-apply-immediately --allow-major-
version-upgrade
...
```

指定されたメンテナンスウィンドウを使用せずにクラスターを作成する場合、Aurora はランダムな曜日を選択します。この場合、modify-db-cluster コマンドは月曜日に送信されます。したがって、メンテナンスウィンドウを火曜日の朝に変更します。すべての時刻は UTC タイムゾーンで表されます。tue:10:00-tue:10:30 期間は、太平洋標準時の午前 2 時から 2 時 30 分に相当します。メンテナンスウィンドウの変更はすぐに有効になります。

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster --query '*[].
[PreferredMaintenanceWindow]'
[
    [
        "sat:08:20-sat:08:50"
    ]
]

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster --preferred-
maintenance-window tue:10:00-tue:10:30"
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster --query '*[].
[PreferredMaintenanceWindow]'
[
    [
        "tue:10:00-tue:10:30"
    ]
]
```

次に、アップグレードによって生成されたイベントのレポートを取得する例を示します。--duration 引数は、イベント情報を取得する時間 (分) を表します。デフォルトでは、describe-events は過去 1 時間のイベントのみを返すため、この引数が必要です。

```
aws rds describe-events --source-type db-cluster --source-identifier mynewdbcluster --
duration 20160
{
    "Events": [
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "DB cluster created",
            "EventCategories": [
```

```
                "creation"
            ],
            "Date": "2022-11-17T01:24:11.093000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Performing online pre-upgrade checks.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T22:57:08.450000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Performing offline pre-upgrade checks.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T22:57:59.519000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-
mynewdbcluster-5-7-mysql-aurora-2-10-2-to-8-0-mysql-aurora-3-02-0-2022-11-18-22-55].",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:00:22.318000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Cloning volume.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:01:45.428000+00:00",
```

```
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Purging undo records for old row versions.
Records remaining: 164",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:02:25.141000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Purging undo records for old row versions.
Records remaining: 164",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:06:23.036000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Upgrading database objects.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:06:48.208000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster major version has been upgraded",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:10:28.999000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        }
```

```
    ]
}
```

アップグレードが失敗した理由の確認

前のチュートリアルでは、Aurora MySQL バージョン 2 からバージョン 3 へのアップグレードに成功しました。ただし、アップグレードが失敗した場合は、必要に応じて、その理由を確認できます。

まず、`describe-events` AWS CLI コマンドを使用して DB クラスターのイベントを表示できます。この例は、過去 10 時間の `mydbcluster` のイベントを示しています。

```
aws rds describe-events \
    --source-type db-cluster \
    --source-identifier mydbcluster \
    --duration 600
```

この例では、アップグレードの事前チェックが失敗しています。

```
{
    "Events": [
        {
            "SourceIdentifier": "mydbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster engine version upgrade started.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2024-04-11T13:23:22.846000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster"
        },
        {
            "SourceIdentifier": "mydbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster is in a state that cannot be upgraded: Upgrade
 prechecks failed. For more details, see the
             upgrade-prechecks.log file. For more information on troubleshooting the
 cause of the upgrade failure, see
             https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
AuroraMySQL.Updates.MajorVersionUpgrade.html#AuroraMySQL.Upgrading.Troubleshooting.",
            "EventCategories": [
                "maintenance"
            ],
```

```
            "Date": "2024-04-11T13:23:24.373000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster"
        }
    ]
}
```

問題の正確な原因を診断するには、ライター DB インスタンスのデータベースログを調べます。Aurora MySQL バージョン 3 へのアップグレードが失敗すると、ライターインスタンスに upgrade-prechecks.log という名前のログファイルが追加されます。この例では、そのログの存在を検出し、それを調査のためにローカルファイルにダウンロードする方法を示します。

```
aws rds describe-db-log-files --db-instance-identifier mydbcluster-instance \
    --query '*[].[LogFileName]' --output text

error/mysql-error-running.log
error/mysql-error-running.log.2024-04-11.20
error/mysql-error-running.log.2024-04-11.21
error/mysql-error.log
external/mysql-external.log
upgrade-prechecks.log

aws rds download-db-log-file-portion --db-instance-identifier mydbcluster-instance \
    --log-file-name upgrade-prechecks.log \
    --starting-token 0 \
    --output text >upgrade_prechecks.log
```

この upgrade-prechecks.log ファイルは JSON 形式です。別の JSON ラッパー内で JSON 出力がエンコードされないように、--output text オプションを使用してこのファイルをダウンロードします Aurora MySQL バージョン 3 のアップグレードでは、このログには常に、特定の情報と警告メッセージが含まれます。エラーメッセージは、アップグレードが失敗した場合にのみ含まれます。アップグレードが成功すると、ログファイルはまったく生成されません。

すべてのエラーを集約して関連するオブジェクトや説明のフィールドを表示するには、upgrade-prechecks.log ファイルの内容に対して grep -A 2 '"level": "Error"' コマンドを実行できます。これにより、各エラー行とその後の 2 行が表示されます。これらの行には、対応するデータベースオブジェクトの名前と、問題の修正方法に関するガイダンスが含まれています。

```
$ cat upgrade-prechecks.log | grep -A 2 '"level": "Error"'

"level": "Error",
"dbObject": "problematic_upgrade.dangling_fulltext_index",
```

```
"description": "Table `problematic_upgrade.dangling_fulltext_index` contains dangling
 FULLTEXT index. Kindly recreate the table before upgrade."
```

この例では、問題のあるテーブルに対して次の SQL コマンドを実行して問題を解決するか、ダングリングインデックスなしでテーブルを再作成することができます。

```
OPTIMIZE TABLE problematic_upgrade.dangling_fulltext_index;
```

次に、アップグレードを再試行します。

Aurora MySQL インプレースアップグレードのトラブルシューティング

以下のヒントを使用して、Aurora MySQL インプレースアップグレードに関する問題のトラブルシューティングを行います。これらのヒントは、Aurora Serverless DB クラスターには適用されません。

インプレースアップグレードがキャンセルされた、または遅い理由	効果	メンテナンス期間内にインプレースアップグレードを完了するためのソリューション
関連する Aurora クロスリージョンレプリカに対するパッチが未適用	Aurora はアップグレードをキャンセルします。	Aurora クロスリージョンレプリカをアップグレードして、もう一度試します。
クラスターに準備済み状態の XA トランザクションがある	Aurora はアップグレードをキャンセルします。	準備済みのすべての XA トランザクションをコミットまたはロールバックします。
クラスターはデータ定義言語 (DDL) ステートメントを処理しています	Aurora はアップグレードをキャンセルします。	すべての DDL ステートメントが終了したら、アップグレードを待って実行することをお勧めします。
クラスターの多くの行で、コミットされていない変更があります	アップグレードには長時間かかる場合があります。	アップグレードプロセスは、コミットされていない変更をロールバックします。この条件のインジケータは、`TRX_ROWS_MODIFIED` テーブ

インプレースアップグレードがキャンセルされた、または遅い理由	効果	メンテナンス期間内にインプレースアップグレードを完了するためのソリューション
		ル内の INFORMATION_SCHEMA.INNODB_TRX の値です。 すべての大きなトランザクションがコミットまたはロールバックされた後にのみ、アップグレードを実行することをお勧めします。
クラスターには多数の UNDO レコードがあります	アップグレードには長時間かかる場合があります。	コミットされていないトランザクションが多数の行に影響を与えない場合でも、大量のデータが含まれる可能性があります。例えば、大きな BLOB を挿入する場合などです。Aurora は、この種のトランザクションアクティビティのイベントを自動的に検出または生成しません。この条件のインジケータは、履歴リスト (HLL) の長さです。アップグレードプロセスは、コミットされていない変更をロールバックします。 HLL は、SHOW ENGINE INNODB STATUS SQL コマンドからの出力で確認することも、次の SQL クエリを使用して直接確認することもできます。 `SELECT count FROM information_schema .innodb_metrics WHERE name = 'trx_rseg_history_len';` また、Amazon CloudWatch で RollbackSegmentHistoryListLength メトリクスをモニタリングすることもできます。 HLL の長さが短くなった後にのみ、アップグレードを実行することをお勧めします。

インプレースアップグレードがキャンセルされた、または遅い理由	効果	メンテナンス期間内にインプレースアップグレードを完了するためのソリューション
クラスターは、大きなバイナリログトランザクションをコミット中です	アップグレードには長時間かかる場合があります。	アップグレードプロセスは、バイナリログの変更が適用されるまで待ちます。この期間中にさらに多くのトランザクションまたは DDL ステートメントがスタートされる可能性があり、アップグレードプロセスの速度はさらに低下します。 クラスターがバイナリログレプリケーションの変更を生成するためのビジー状態でない場合に、アップグレードプロセスがスケジュールされます。Aurora は、この条件のイベントを自動的に検出または生成しません。

インプレースアップグレードがキャンセルされた、または遅い理由	効果	メンテナンス期間内にインプレースアップグレードを完了するためのソリューション
ファイルの削除または破損によるスキーマの不一致	Aurora はアップグレードをキャンセルします。	一時テーブルのデフォルトのストレージエンジンを MyISAM から InnoDB に変更します。以下のステップを実行します。 1. default_tmp_storage_engine DB パラメータを InnoDB に変更します。 2. DB クラスターを再起動します。 3. 再起動後、default_tmp_storage_engine DB パラメータが InnoDB に設定されていることを確認します。以下のコマンドを使用します。 `show global variables like 'default_tmp_storage_engine';` 4. MyISAM ストレージエンジンを使用する一時テーブルは作成しないでください。間もなくアップグレードを行う予定なので、データベースワークロードを一時停止し、新しいデータベース接続を作成しないことをお勧めします。 5. インプレースアップグレードを再試行してください。

インプレースアップグレードがキャンセルされた、または遅い理由	効果	メンテナンス期間内にインプレースアップグレードを完了するためのソリューション

マスターユーザーが削除されました	Aurora はアップグレードをキャンセルします。	**Important**
		マスターユーザーを削除しないでください。
		
		ただし、何らかの理由でマスターユーザーを削除する必要がある場合は、次の SQL コマンドを使用して復元します。
		
		`CREATE USER 'master_username '@'%' IDENTIFIED BY ' master_user_password ' REQUIRE NONE PASSWORD EXPIRE DEFAULT ACCOUNT UNLOCK;`
		
		`GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, PROCESS, REFERENCES, INDEX, ALTER, SHOW DATABASES, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, LOAD FROM S3, SELECT INTO S3, INVOKE LAMBDA, INVOKE SAGEMAKER, INVOKE COMPREHEND ON *.* TO 'master_username '@'%' WITH GRANT OPTION;`

アップグレードの事前チェックの失敗を起こす問題のトラブルシューティングの詳細については、以下のブログを参照してください。

- Amazon Aurora MySQL バージョン 2 (MySQL 5.7 互換) からバージョン 3 (MySQL 8.0 互換) へのアップグレードチェックリスト、パート 1

- Amazon Aurora MySQL バージョン 2 (MySQL 5.7 互換) からバージョン 3 (MySQL 8.0 互換) へのアップグレードチェックリスト、パート 2

以下のステップを使用して、上記の表の一部の条件に対して独自のチェックを実行できます。これにより、データベースがアップグレードを正常かつ迅速に完了できる状態にあることが分かっているときに、アップグレードをスケジュールすることができます。

- XA RECOVER ステートメントを実行することで、開いている XA トランザクションを確認できます。アップグレードのスタート前に、XA トランザクションをコミットまたはロールバックできます。
- SHOW PROCESSLIST ステートメントを実行し、出力でCREATE、DROP、ALTER、RENAME、および TRUNCATE ステートメントを探して、DDL ステートメントを確認できます。アップグレードのスタート前に、すべての DDL ステートメントを完了させます。
- INFORMATION_SCHEMA.INNODB_TRX テーブルをクエリすることで、コミットされていない行の総数を確認できます。テーブルには、トランザクションごとに 1 つの行が含まれます。TRX_ROWS_MODIFIED 列には、トランザクションによって変更または挿入された行の数が含まれます。
- InnoDB 履歴リストの長さを確認するには、SHOW ENGINE INNODB STATUS SQL ステートメントを実行し、出力で History list length を探します。次のクエリを実行して、値を直接確認することもできます。

```
SELECT count FROM information_schema.innodb_metrics WHERE name =
 'trx_rseg_history_len';
```

  履歴リストの長さは、マルチバージョン同時実行制御 (MVCC) の実装のためにデータベースに保存される UNDO 情報の量に対応します。

Aurora MySQL バージョン 3 のアップグレード後のクリーンアップ

Aurora MySQL バージョン 2 クラスターを Aurora MySQL バージョン 3 にアップグレードした後、次のようなその他のクリーンアップアクションを実行できます。

- カスタム パラメータグループの新しい MySQL 8.0 互換バージョンを作成します。必要なカスタムパラメータ値を新しいパラメータグループに適用します。

- CloudWatch アラーム、セットアップスクリプトなどを更新して、インクルーシブな言語変更の影響を受けた名前のメトリクスに、新しい名前を使用します。このようなメトリクスの一覧は、Aurora MySQL バージョン 3 に対する包括的な言語変更 をご覧ください。
- AWS CloudFormation テンプレートをすべて更新して、包括的な言語の変更によって影響を受けるすべてのコンフィギュレーション パラメータの名前を新しくします。そのようなパラメータのリストについては、Aurora MySQL バージョン 3 に対する包括的な言語変更 を参照してください。

空間インデックス

Aurora MySQL バージョン 3 にアップグレードした後、空間インデックスに関連するオブジェクトおよびインデックスを削除または再作成する必要があるかどうかをチェックします。MySQL 8.0 より前では、Aurora は空間リソース識別子 (SRID) を含まないインデックスを使用して空間クエリを最適化できました。Aurora MySQL バージョン 3 では、SRID を含む空間インデックスのみを使用します。アップグレード中、Aurora は SRID のない空間インデックスを自動的にドロップし、警告メッセージをデータベースログに出力します。このような警告メッセージが表示された場合は、アップグレード後に SRID を使用して新しい空間インデックスを作成します。MySQL 8.0 での空間関数とデータ型の変更の詳細については、MySQL リファレンスマニュアルの「MySQL 8.0 での変更」を参照してください。

Amazon Aurora MySQL に関するデータベースエンジンの更新と修正

「Amazon Aurora MySQL 互換エディションのリリースノート」では、以下の情報を参照できます。

- Amazon Aurora MySQL バージョン 3 のデータベースエンジンの更新
- Amazon Aurora MySQL バージョン 2 のデータベースエンジンの更新
- Amazon Aurora MySQL バージョン 1 のデータベースエンジンの更新 (非推奨)
- Aurora MySQL データベースエンジンの更新で修正された MySQL のバグ
- Amazon Aurora MySQL で修正されたセキュリティの脆弱性

Amazon Aurora PostgreSQL の操作

Amazon Aurora PostgreSQL は、フルマネージド型で PostgreSQL 互換の、ACID 準拠のリレーショナルデータベースエンジンです。Amazon Aurora のスピード、信頼性、管理性を、オープンソースデータベースのシンプルさとコスト効率でご利用いただけます。PostgreSQL を Aurora PostgreSQL に差し替えることで、新規および既存の PostgreSQL のデプロイを簡単に、コスト効率よく設定、操作、スケーリングできるようになり、ユーザーは本来のビジネスやアプリケーションに専念できます。Aurora 全般についての詳細は、「Amazon Aurora とは」を参照してください。

Aurora PostgreSQL は、Aurora の利点に加え、Amazon RDS から Aurora に移行する便利な方法を提供します。プッシュボタン式の移行ツールを使用することで、既存の RDS for PostgreSQL アプリケーションを Aurora PostgreSQL に変換できます。プロビジョニング、パッチ適用、バックアップ、リカバリ、障害検出、修復などの日常的なデータベースタスクも Aurora PostgreSQL で簡単に管理できます。

Aurora PostgreSQL は多くの業界スタンダードに準拠しています。例えば、Aurora PostgreSQL データベースを使用して、HIPAA 準拠のアプリケーションを構築し、AWS との間で締結した事業提携契約 (BAA) に基づき、保護された医療情報 (PHI) などの医療関連情報を保存できます。

Aurora PostgreSQL は FedRAMP HIGH に対応しています。AWS およびコンプライアンスの取り組みの詳細については、コンプライアンスプログラムによる AWS 対象範囲内のサービスを参照してください。

トピック

- データベースプレビュー環境の使用
- Amazon Aurora PostgreSQL でのセキュリティ
- 新しい SSL/TLS 証明書を使用して Aurora PostgreSQL DB クラスターに接続するようにアプリケーションを更新する
- Aurora PostgreSQL で Kerberos 認証を使用する
- PostgreSQL と互換性がある Amazon Aurora にデータを移行する
- Amazon Optimized Reads による Aurora PostgreSQL のクエリパフォーマンスの向上
- Babelfish for Aurora PostgreSQL の使用
- Amazon Aurora PostgreSQL の管理

- Aurora PostgreSQL の待機イベントでのチューニング
- Amazon DevOps Guru のプロアクティブインサイトによる Aurora PostgreSQL のチューニング
- Amazon Aurora PostgreSQL を使用する際のベストプラクティス
- Amazon Aurora PostgreSQL でのレプリケーション
- Amazon Bedrock のナレッジベースとしての Aurora PostgreSQL の使用
- Amazon Aurora PostgreSQL を他の AWS のサービスと統合する
- Aurora PostgreSQL のクエリ実行計画のモニタリング
- Aurora PostgreSQL のクエリ実行計画の管理
- エクステンションと外部データラッパーの使用
- Trusted Language Extensions for PostgreSQL を使用した操作
- Amazon Aurora PostgreSQL のリファレンス
- Amazon Aurora PostgreSQL の更新

データベースプレビュー環境の使用

PostgreSQL コミュニティは、PostgreSQL の新しいメジャーバージョンを毎年リリースしています。同様に、Amazon Aurora では特定の PostgreSQL メジャーバージョンをプレビューリリースとして提供するようにしています。これにより、プレビューバージョンを使用して DB クラスターを作成し、その機能をデータベースプレビュー環境でテストできます。

データベースプレビュー環境の Aurora PostgreSQL DB クラスターは、機能的には他の Aurora PostgreSQL DB クラスターと似ています。ただし、プレビューバージョンは本稼働に使用できません。

次の重要な制約事項に留意してください。

- すべての DB インスタンスおよび DB クラスターは、作成から 60 日後にバックアップおよびスナップショットとともに削除されます。
- DB インスタンスは、Amazon VPC サービスに基づく仮想プライベートクラウド (VPC) でのみ作成できます。
- DB インスタンスのスナップショットを本稼働環境にコピーすることはできません。

プレビューでは、以下のオプションがサポートされています。

- DB インスタンスは、r5、r6g、r6i、r7g、x2g、t3 および t4g インスタンスタイプのみで作成できます。インスタンスクラスの詳細については、「Aurora DB インスタンスクラス」を参照してください。
- シングル AZ 配置とマルチ AZ 配置の両方を使用できます。
- スタンダードの PostgreSQL ダンプおよびロード機能を使用して、データベースをデータベースプレビュー環境にエクスポートしたり、データベースプレビュー環境にインポートしたりできます。

サポートされている DB インスタンスクラスタイプ

Amazon Aurora PostgreSQL は、プレビューリージョンで次の DB インスタンスクラスをサポートしています。

メモリ最適化クラス

- db.r5 – メモリ最適化インスタンスクラス
- db.r6g – AWS Graviton2 プロセッサを搭載したメモリ最適化インスタンスクラス
- db.r6i – メモリ最適化インスタンスクラス
- db.x2g – AWS Graviton2 プロセッサを搭載したメモリ最適化インスタンスクラス

> **Note**
>
> インスタンスクラスのリストの詳細については、「DB インスタンスクラスタイプ」を参照してください。

バーストクラス

- db.t3.medium
- db.t3.large
- db.t4g.medium
- db.t4g.large

プレビュー環境でサポートされない機能

以下の機能は、プレビュー環境で使用できません。

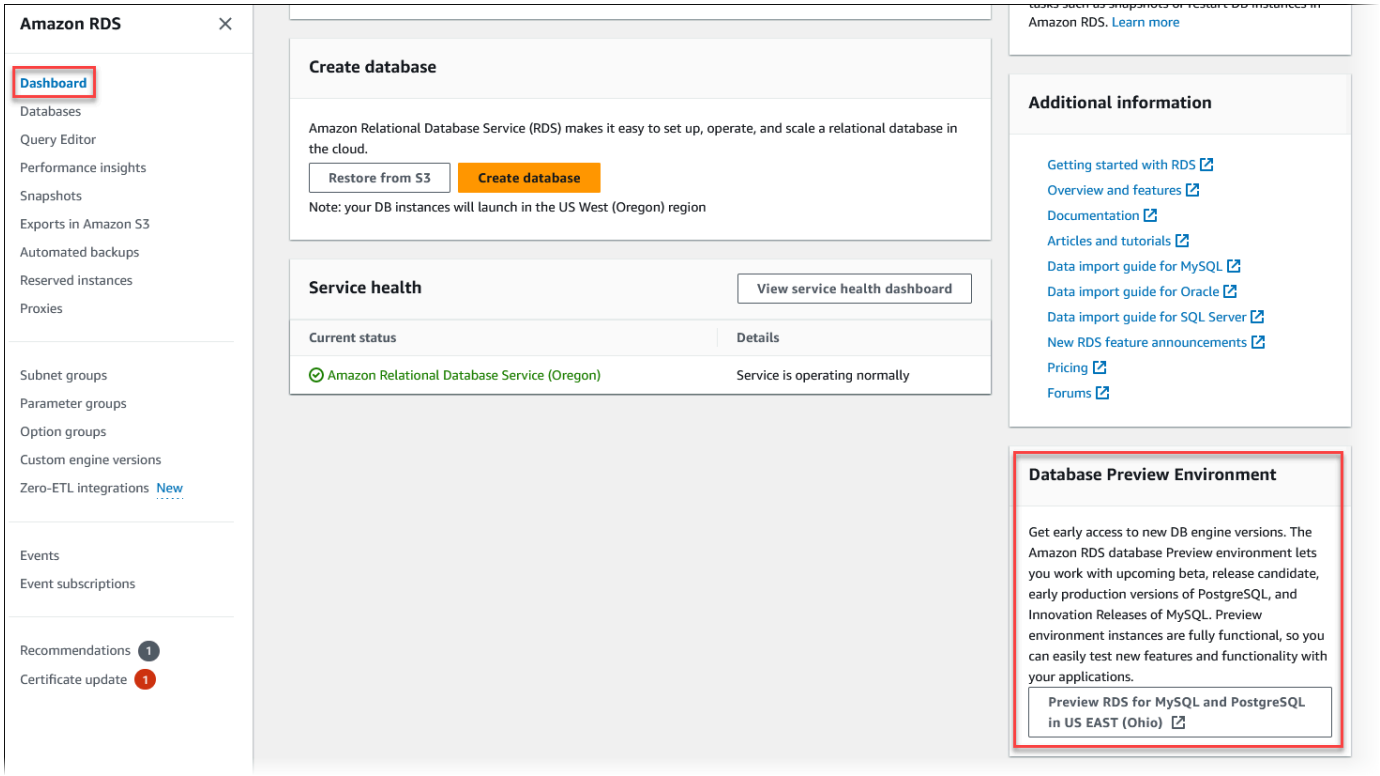
- Aurora Serverless v1 および v2
- メジャーバージョンのアップグレード (MVU)
- プレビューリージョンでは新しいマイナーバージョンはリリースされません
- RDS for PostgreSQL から Aurora PostgreSQL へのインバウンドレプリケーション
- Amazon RDS ブルー/グリーンデプロイ
- クロスリージョンスナップショットのコピー
- Aurora Global Database
- データベースアクティビティストリーム (DAS)、RDS Proxy、および AWS DMS
- リードレプリカの自動スケーリング
- AWS Bedrock
- RDS エクスポート
- Performance Insights
- グローバル書き込み転送
- Optimized Reads
- Babelfish
- カスタムエンドポイント
- スナップショットコピー

プレビュー環境での新しい DB クラスターの作成

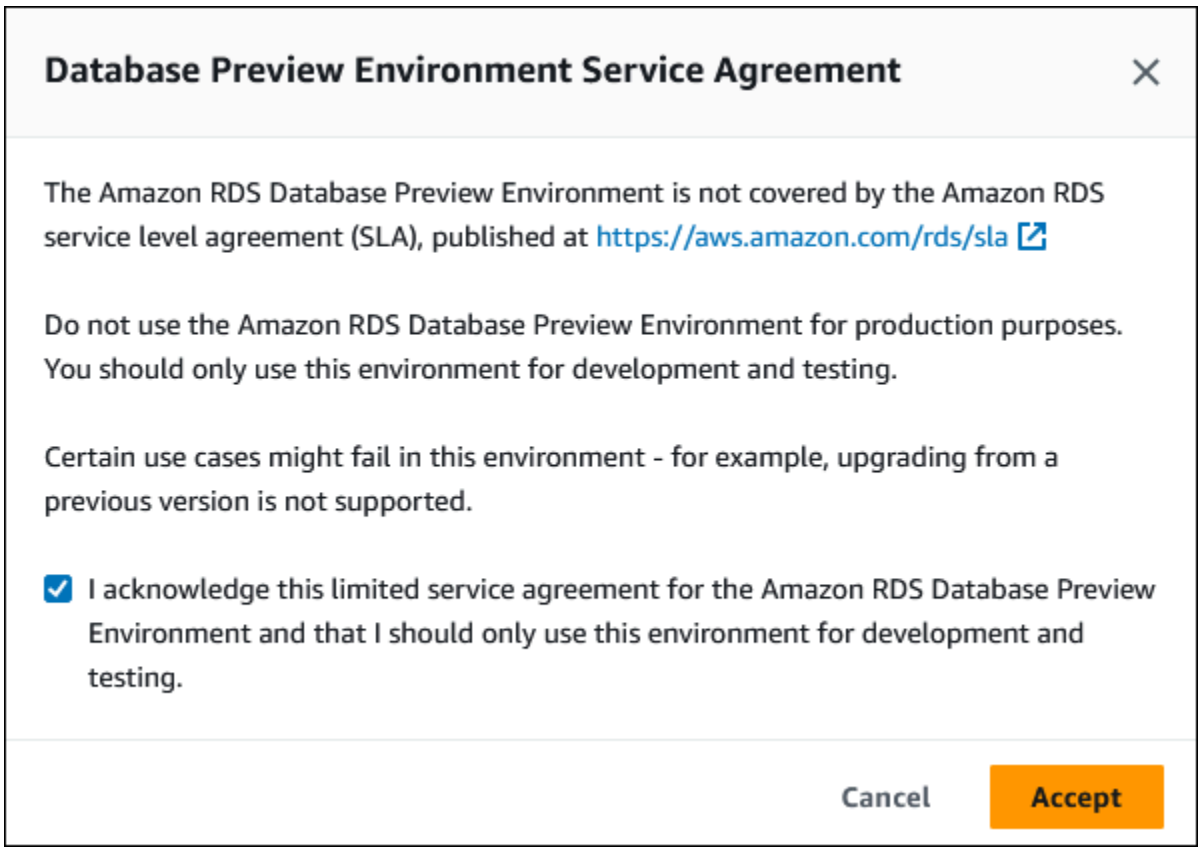
プレビュー環境で DB クラスターを作成するには、次の手順を使用します。

プレビュー環境で新しい DB クラスターを作成するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[ダッシュボード] を選択します。
3. [Dashboard] (ダッシュボード) ページで、次の図に示すように、[Dashboard] (ダッシュボード) ページの [Database Preview Environment] (データベースプレビュー環境) セクションを見つけます。



また、[Database Preview Environment] (データベースプレビュー環境) に直接移動することもできます。先に進む前に、制限事項を確認して同意する必要があります。



4. Aurora PostgreSQL DB クラスターを作成するには、Aurora DB クラスターを作成する場合と同じプロセスに従います。詳細については、「Amazon Aurora DB クラスターの作成」を参照してください。

Aurora API または AWS CLI を使用してデータベースプレビュー環境でインスタンスを作成するには、次のエンドポイントを使用します。

```
rds-preview.us-east-2.amazonaws.com
```

データベースプレビュー環境の PostgreSQL バージョン 16

⚠ これは Aurora PostgreSQL バージョン 16 のプレビュードキュメントです。このドキュメントは変更される可能性があります。

PostgreSQL バージョン 16.0 が Amazon RDS データベースプレビュー環境で利用可能になりました。PostgreSQL バージョン 16 には、次の PostgreSQL ドキュメントに記載されているいくつかの改善点が含まれています。

- PostgreSQL 16 がリリースされました

データベースプレビュー環境の詳細については、「データベースプレビュー環境の使用」を参照してください。コンソールからプレビュー環境にアクセスするには、https://console.aws.amazon.com/rds-preview/ を選択します。さい。

> **Note**
>
> Aurora PostgreSQL バージョン 16.1 が一般利用可能になったため、データベースプレビュー環境で PostgreSQL バージョン 16.0 を使用することはお勧めしません。詳細については、「Amazon Aurora PostgreSQL の更新」を参照してください。

Amazon Aurora PostgreSQL でのセキュリティ

Aurora のセキュリティの概要については、「Amazon Aurora でのセキュリティ」を参照してください。Amazon Aurora PostgreSQL のセキュリティは、いくつかの異なるレベルで管理できます。

- Aurora PostgreSQL DB クラスターと DB インスタンスに対し Amazon RDS 管理アクションを実行できるユーザーを管理するには、AWS Identity and Access Management (IAM) を使用します。IAM は、ユーザーがサービスにアクセスする前に、ユーザー ID の認証を処理します。また、承認、つまりユーザーが行おうとしていることが許可されているかどうかについても処理します。IAM データベース認証は、Aurora PostgreSQL DB クラスターを作成するときに選択できる追加の認証方法です。詳細については、「Amazon Aurora での Identity and Access Management」を参照してください。

  IAM を Aurora PostgreSQL DB クラスターで使用する場合は、Amazon RDS コンソールを https://console.aws.amazon.com/rds/ で開く前に、まず、IAM 認証情報を使用して AWS Management Console にサインインしてください。
- Aurora DB クラスターを Amazon VPC サービスに基づいて仮想プライベートクラウド (VPC) で作成してください。VPC 内の Aurora DB クラスター用の DB インスタンスのエンドポイントとポートに対して接続を開くことができるデバイスと Amazon EC2 インスタンスを制御するには、VPC セキュリティグループを使用します。これらのエンドポイントとポート接続は、Secure Sockets Layer (SSL) を使用して作成できます。さらに、会社のファイアウォールルールでも、社内のいず

れのデバイスが DB インスタンスへの接続を開くことができるかを制御できます。VPC の詳細については、「Amazon VPC VPC とAmazon Aurora」を参照してください。

サポートされている VPC テナンシーは、Aurora PostgreSQL DB クラスターで使用しているインスタンスクラスによって異なります。`default` VPC テナンシーでは、DB クラスターは共有ハードウェアで実行されます。`dedicated` VPC テナンシーでは、DB クラスターは専用ハードウェアインスタンスで実行されます。バーストパフォーマンス DB インスタンスクラスでは、デフォルト VPC テナンシーのみがサポートされています。バーストパフォーマンス DB インスタンスクラスには、db.t3 および db.t4g DB インスタンスクラスが含まれます。その他すべての Aurora PostgreSQL DB インスタンスクラスでは、デフォルトと専用 VPC テナンシーの両方がサポートされています。

インスタンスクラスの詳細については、「Aurora DB インスタンスクラス」を参照してください。`default` および `dedicated` VPC テナントの詳細については、Amazon Elastic Compute Cloud ユーザーガイドの「ハードウェア専有インスタンス」を参照してください。

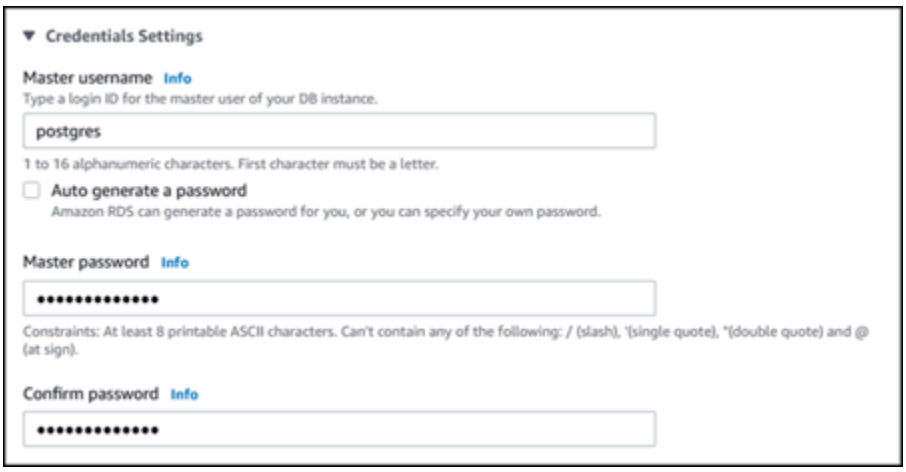
- Amazon Aurora DB クラスターで実行されている PostgreSQL データベースにアクセス権限を付与するには、PostgreSQL のスタンドアロンインスタンスと同じ一般的なアプローチを使用します。`CREATE ROLE`、`ALTER ROLE`、`GRANT`、`REVOKE` などのコマンドは、オンプレミスデータベースでの方法と同様に、データベース、スキーマ、およびテーブルを直接変更します。

  PostgreSQL は、ロールを使用して権限を管理します。`rds_superuser` ロールは、Aurora PostgreSQL DB クラスターで最も権限があるロールです。このロールは自動的に作成され、DB クラスターを作成するユーザー (マスターユーザーアカウント、デフォルトでは `postgres`) に付与されます。詳細については、「PostgreSQL のロールとアクセス権限について」を参照してください。

バージョン 10、11、12、13、14 以降のリリースを含む Aurora PostgreSQL バージョンでは、メッセージダイジェスト (MD5) の代替として、Salted Challenge Response Authentication Mechanism (SCRAM) がサポートされています。SCRAM は MD5 よりも安全であるため、使用が推奨されています。データベースユーザーパスワードを MD5 から SCRAM に移行する方法など、詳細については、「PostgreSQL のパスワード暗号化に SCRAM を使用する」を参照してください。

PostgreSQL のロールとアクセス権限について

AWS Management Console を使用してAurora PostgreSQL DB クラスター を作成すると、管理者アカウントが同時に作成されます。次のスクリーンショットに示すように、デフォルトでは `postgres` という名前になります。



デフォルト設定 (postgres) を受け入れるのではなく、別の名前を選択することもできます。この場合、選択する名前はアルファベットで始まり、1 文字以上 16 文字以下の英数字である必要があります。このガイドでは、わかりやすくするために、このメインユーザーアカウントをデフォルトの値 (postgres) で表記しています。

AWS Management Console ではなくcreate-db-cluster AWS CLI を使用する場合は、master-username パラメータと一緒に渡すことでユーザー名を作成します。詳細については、ステップ 2: Aurora PostgreSQL DB クラスターを作成する をご参照ください。

AWS Management Console、AWS CLI、または Amazon RDS API のいずれを使用する場合でも、またデフォルトの postgres 名を使用するか、別の名前を選択するかにかかわらず、この最初のデータベースユーザーアカウントは rds_superuser グループのメンバーであり、rds_superuser 権限を持つことになります。

トピック

- rds_superuser ロールを理解する
- PostgreSQL データベースへのユーザーアクセスのコントロール
- ユーザーパスワード管理の委任と制御
- PostgreSQL のパスワード暗号化に SCRAM を使用する

rds_superuser ロールを理解する

PostgreSQL では、ロールは、データベース内のさまざまなオブジェクトに対して、ユーザー、グループ、またはグループやユーザーに与えられた特定のアクセス権限を定義することができます。CREATE USER と CREATE GROUP に対する PostgreSQL コマンドは、データベースユーザーを

区別するために、より一般的な、特定のプロパティを持つ CREATE ROLE に置き換えられました。データベースユーザーは、LOGIN 権限を持つロールと考えることができます。

> **Note**
>
> CREATE USER および CREATE GROUP コマンドは引き続き使用できます。詳細については、PostgreSQL のドキュメントの「データベースロール」セクションを参照してください。

postgres ユーザーは、Aurora PostgreSQL DB クラスター で最も権限があるデータベースユーザーです。ユーザーには、次の CREATE ROLE ステートメントで定義される特性があります。

```
CREATE ROLE postgres WITH LOGIN NOSUPERUSER INHERIT CREATEDB CREATEROLE NOREPLICATION
 VALID UNTIL 'infinity'
```

特に指定がない限り、プロパティ NOSUPERUSER、NOREPLICATION、INHERIT、および VALID UNTIL 'infinity' が CREATE ROLE のデフォルトオプションです。

デフォルトでは、postgres には rds_superuser ロールに付与された権限と、ロールとデータベースを作成するアクセス許可があります。rds_superuser ロールでは、postgres ユーザーによる次の操作を許可します。

- Aurora PostgreSQL」を参照してください。詳細については、エクステンションと外部データラッパーの使用 をご参照ください。
- ユーザーのロールを作成し、ユーザーに権限を付与します。詳細については、PostgreSQL のドキュメントの「CREATE ROLE」および「GRANT」セクションを参照してください。
- データベースの作成 詳細については、PostgreSQL のドキュメントの「CREATE DATABASE」を参照してください。
- これらの権限を持たないユーザーロールに対する rds_superuser 権限を付与し、必要に応じてそれらの権限を取り消します。このロールは、スーパーユーザータスクを実行するユーザーにのみ付与することをお勧めします。つまり、データベース管理者 (DBA) またはシステム管理者にこのロールを付与できます。
- rds_superuser ロールを持たないデータベースユーザーに rds_replication ロールを付与 (または取り消し) します。
- rds_superuser ロールを持たないデータベースユーザーに rds_password ロールを付与 (または取り消し) します。

- pg_stat_activity ビューを使用して、すべてのデータベース接続に関するステータス情報を取得します。必要に応じて、rds_superuser で pg_terminate_backend または pg_cancel_backend を使用して接続を停止できます。

CREATE ROLE postgres... ステートメントで、postgres ユーザーロールは PostgreSQL の superuser アクセス許可を特に禁止することがわかります。Aurora PostgreSQL はマネージドサービスのため、ホスト OS へのアクセスや、PostgreSQL superuser アカウントを使用した接続はできません。スタンドアロンの PostgreSQL で superuser のアクセスが必要な作業の多くは、Aurora で自動的に管理されます。

権限の付与に関する詳細については、PostgreSQL のドキュメントの「GRANT」を参照してください。

rds_superuser ロールは、Aurora PostgreSQL DB クラスターにおけるいくつかの事前定義済みロールの 1 つです。

> **Note**
>
> PostgreSQL 13 以前のリリースでは、定義済みロールはデフォルトロールと呼ばれていました。

次のリストに、新しい Aurora PostgreSQL DB クラスターのために自動的に作成される他の定義済みロールの一部を示します。定義済みロールとその権限は変更できません。これらの定義済みロールに対して削除、名前の変更、変更を行うことはできません。それらの操作を試みると、エラーが発生します。

- rds_password - データベースユーザーのパスワードを変更し、パスワード制約を設定できるロールです。rds_superuser ロールにはデフォルトでこのロールが付与され、データベースユーザーにロールを付与できます。詳細については、「PostgreSQL データベースへのユーザーアクセスのコントロール」を参照してください。
  - 14 より前のバージョンの RDS for PostgreSQL の場合、rds_password ロールはパスワードを変更し、データベースユーザーと rds_superuser ロールを持つユーザーのパスワード制約を設定できます。RDS for PostgreSQL 14 以降のバージョンでは、rds_password ロールがユーザーのパスワードを変更し、パスワード制約を設定できるのは、データベースユーザーに対してのみです。rds_superuser ロールを持つユーザーのみが、rds_superuser ロールを持つ他のユーザーに対して上記のアクションを実行できます。

- rdsadmin - superuser 権限を持つ管理者がスタンドアロンの PostgreSQL データベースで行う管理タスクの多くを処理するために作成されるロールです。このロールは、Aurora PostgreSQL によって多くの管理タスクのために内部的に使用されます。

定義済みのロールをすべて表示するには、Aurora PostgreSQL DB クラスターのプライマリインスタンス に接続し、psql \du メタコマンドを使用します。出力は次のとおりです。

```
List of roles
  Role name   |      Attributes                    |      Member of
--------------+------------------------------------+-----------------------------------
postgres      | Create role, Create DB            +| {rds_superuser}
              | Password valid until infinity      |
rds_superuser | Cannot login                       | {pg_monitor,pg_signal_backend,
              |                                   +|   rds_replication,rds_password}
...
```

出力では、rds_superuser がデータベースユーザーロールではない (ログインできない) が、他の多くのロールの特権を持っていることがわかります。また、そのデータベースユーザー postgres は rds_superuser ロールのメンバーであることも確認できます。前述のように、postgres が Amazon RDS コンソールの [Create database] (データベースを作成) ページのデフォルト値です。別の名前を選択した場合、代わりにその名前がロールのリストに表示されます。

> **Note**
>
> Aurora PostgreSQL バージョン 15.2 と 14.7 では、rds_superuser ロールの限定的動作が導入されました。Aurora PostgreSQL ユーザーには、ユーザーに rds_superuser ロールが付与されている場合でも、対応する接続対象のデータベースで CONNECT 権限を付与する必要があります。Aurora PostgreSQL バージョン 14.7 と 15.2 より前のバージョンでは、ユーザーにrds_superuser 権限が付与されていれば、どのデータベースとシステムテーブルにも接続できました。この制限的動作は、AWS そして、セキュリティの継続的な改善に取り組むという Amazon Aurora コミットメントに整合しています。
> 上記の機能強化が影響する場合は、アプリケーションのそれぞれのロジックを更新してください。

PostgreSQL データベースへのユーザーアクセスのコントロール

PostgreSQL の新しいデータベースは、常にデータベースの public スキーマに、すべてのデータベースユーザーとロールがオブジェクトを作成できるようなデフォルトの権限セットで作成されます。これらの権限により、例えば、データベースユーザーがデータベースに接続し、接続しながら一時テーブルを作成することができます。

Aurora PostgreSQL DB クラスターのプライマリノード に作成するデータベースインスタンスへのユーザーアクセスをよりよく制御するために、これらのデフォルトの public 権限を取り消すことを推奨します。その後、次の手順で示すように、データベースのユーザーに特定の権限をより詳細に付与します。

新しいデータベースインスタンスのロールと権限を設定するには

全員がデータベースへの読み取り/書き込みアクセスを必要とする複数の研究者が使用するために、新しく作成された Aurora PostgreSQL DB クラスター 上にデータベースをセットアップしているとします。

1. psql (または pgAdmin) を使用して、Aurora PostgreSQL DB クラスター上のプライマリ DB インスタンス に接続します。

   ```
   psql --host=your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password
   ```

   プロンプトが表示されたら、パスワードを入力します。psql クライアントが接続し、プロンプトとしてデフォルトの管理用接続データベースである postgres=> を表示します。
2. データベースユーザーが public スキーマでオブジェクトを作成できないようにするには、次の操作を行います。

   ```
   postgres=> REVOKE CREATE ON SCHEMA public FROM PUBLIC;
   REVOKE
   ```
3. 次に、新しいデータベースインスタンスを作成します。

   ```
   postgres=> CREATE DATABASE lab_db;
   CREATE DATABASE
   ```
4. この新しいデータベースの PUBLIC スキーマからすべての権限を取り消します。

   ```
   postgres=> REVOKE ALL ON DATABASE lab_db FROM public;
   ```

```
REVOKE
```

5. データベースユーザーのロールを作成します。

```
postgres=> CREATE ROLE lab_tech;
CREATE ROLE
```

6. このロールを持つデータベースユーザーに、データベースに接続する機能を付与します。

```
postgres=> GRANT CONNECT ON DATABASE lab_db TO lab_tech;
GRANT
```

7. lab_tech ロールを持つすべてのユーザーに、このデータベースのすべての権限を付与します。

```
postgres=> GRANT ALL PRIVILEGES ON DATABASE lab_db TO lab_tech;
GRANT
```

8. 次のように、データベースユーザーを作成します。

```
postgres=> CREATE ROLE lab_user1 LOGIN PASSWORD 'change_me';
CREATE ROLE
postgres=> CREATE ROLE lab_user2 LOGIN PASSWORD 'change_me';
CREATE ROLE
```

9. これら 2 人のユーザーに lab_tech ロールに関連付けられた権限を付与します。

```
postgres=> GRANT lab_tech TO lab_user1;
GRANT ROLE
postgres=> GRANT lab_tech TO lab_user2;
GRANT ROLE
```

この時点で、lab_user1 と lab_user2 は lab_db データベースに接続できます。この例は、複数のデータベースインスタンスの作成、異なるスキーマの作成、制限されたアクセス許可の付与などを含む、エンタープライズで使用するためのベストプラクティスに従ったものではありません。詳細な情報と追加のシナリオについては、「PostgreSQL ユーザーとロールの管理」を参照してください。

PostgreSQL データベースでの権限の詳細については、PostgreSQL のドキュメントの GRANT コマンドを参照してください。

ユーザーパスワード管理の委任と制御

DBA は、ユーザーパスワードの管理を委任する場合があります。または、データベースユーザーがパスワードを変更したり、パスワードの有効期間などのパスワード制約を再設定したりしないようにする場合もあります。選択したデータベースユーザーのみがパスワード設定を変更できるようにするには、制限されたパスワード管理の機能をオンにします。この機能をアクティブにすると、rds_password ロールを付与されたデータベースユーザーのみがパスワードを管理できます。

> **Note**
>
> 制限されたパスワード管理を使用するには、 Aurora PostgreSQL DB クラスターで Amazon Aurora PostgreSQL 10.6 以上を実行している必要があります。

次に示すように、デフォルトではこの機能は off になっています。

```
postgres=> SHOW rds.restrict_password_commands;
  rds.restrict_password_commands
--------------------------------
 off
(1 row)
```

この機能をオンにするには、カスタムパラメータグループを使用して、rds.restrict_password_commands の設定を 1 に変更します。設定を有効にするには、Aurora PostgreSQL のプライマリ DB インスタンス を必ず再起動してください。

この機能をアクティブにすると、次の SQL コマンドには rds_password 権限が必要になります。

```
CREATE ROLE myrole WITH PASSWORD 'mypassword';
CREATE ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2023-01-01';
ALTER ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2023-01-01';
ALTER ROLE myrole WITH PASSWORD 'mypassword';
ALTER ROLE myrole VALID UNTIL '2023-01-01';
ALTER ROLE myrole RENAME TO myrole2;
```

ロールの名前の変更 (ALTER ROLE myrole RENAME TO newname) は、パスワードが MD5 ハッシュアルゴリズムを使用する場合にも制限されます。

この機能が有効な場合、rds_password ロールのアクセス許可なしでこれらの SQL コマンドの実行を試みると、次のエラーが発生します。

```
ERROR: must be a member of rds_password to alter passwords
```

rds_password は、パスワード管理専用の少数のロールにのみ付与することをお勧めします。rds_superuser 権限を持たないデータベースユーザーに rds_password 権限を付与する場合は、CREATEROLE 属性も付与する必要があります。

パスワード要件 (クライアント側の有効期限や必要な複雑さなど) を確認してください。パスワード関連の変更に独自のクライアント側ユーティリティを使用する場合、そのユーティリティは rds_password のメンバーであり、CREATE ROLE 権限を持つ必要があります。

PostgreSQL のパスワード暗号化に SCRAM を使用する

SCRAM (Salted Challenge Response Authentication Mechanism) は、パスワードを暗号化するための PostgreSQL のデフォルトのメッセージダイジェスト (MD5) アルゴリズムの代替手段です。SCRAM 認証メカニズムは MD5 よりも安全であると見なされます。これら 2 つの異なるパスワードを保護する方法の詳細については、PostgreSQL のドキュメントの「パスワード認証」を参照してください。

Aurora PostgreSQL DB クラスターに対しては、パスワード暗号化方式として MD5 ではなく SCRAM を使用することをお勧めします。Aurora PostgreSQL 14 リリース現在、SCRAM はバージョン 10、11、12、13、14 を含む利用可能なすべての Aurora PostgreSQL バージョンでサポートされています。これは、パスワード認証と暗号化のために scram-sha-256 アルゴリズムを使用する暗号化チャレンジレスポンスのメカニズムです。

SCRAM をサポートするために、クライアントアプリケーションのライブラリを更新する必要があります。例えば、42.2.0 より前の JDBC バージョンで SCRAM はサポートされていません。詳細については、PostgreSQL JDBC ドライバーのドキュメントの「PostgreSQL JDBC ドライバー」を参照してください。その他の PostgreSQL ドライバーおよび SCRAM サポートの一覧については、PostgreSQL のドキュメントの「ドライバーの一覧」を参照してください。

> **Note**
>
> Aurora PostgreSQL バージョン 14 以上では、新しい DB クラスターに対してデフォルトでパスワード暗号化に scram-sha-256 をサポートしています。つまり、デフォルトの DB クラスターパラメータグループ (default.aurora-postgresql14) が持っている password_encryption 値を scram-sha-256 に設定します。

SCRAM を要求するために Aurora PostgreSQL DB クラスターを設定する

Aurora PostgreSQL 14.3 以降のバージョンでは、scram-sha-256 アルゴリズムを使用するパスワードのみを受け入れるために、Aurora PostgreSQL DB クラスターに を要求できます。

> **Important**
>
> PostgreSQL データベースを使用する既存の RDS プロキシでは、SCRAM のみを使用するようにデータベース認証を変更すると、プロキシは最大 60 秒間使用できなくなります。この問題を回避するには、以下のいずれかの方法で対応します。
>
> - データベースが SCRAM と MD5 認証の両方を許可していることを確認します。
> - SCRAM 認証のみを使用するには、新しいプロキシを作成し、アプリケーショントラフィックを新しいプロキシに移行してから、以前にデータベースに関連付けられていたプロキシを削除します。

システムに変更を加える前に、次の完全なプロセスを理解していることを確認してください。

- すべてのデータベースユーザーのすべてのロールとパスワードの暗号化に関する情報を取得します。
- パスワードの暗号化を制御するパラメータを指定するために、Aurora PostgreSQL DB クラスターのパラメータ設定を再確認してください。
- Aurora PostgreSQL DB クラスター でデフォルトのパラメータグループを使用する場合は、カスタムの DB クラスターのパラメータグループ を作成して、それを Aurora PostgreSQL DB クラスター に適用し、必要なときにパラメータを変更できるようにする必要があります。Aurora PostgreSQL DB クラスター がカスタムパラメータグループを使用している場合、必要に応じて、プロセスの後で必要なパラメータを変更できます。
- `password_encryption` パラメータを `scram-sha-256` に変更します。
- パスワードを更新する必要があることをすべてのデータベースユーザーに通知します。`postgres` アカウントに同じ操作を行います。新しいパスワードは暗号化され、scram-sha-256 アルゴリズムを使用して保存されます。
- 暗号化の種類を使用して、すべてのパスワードが暗号化されていることを確認します。
- すべてのパスワードで scram-sha-256 が使用されている場合、`rds.accepted_password_auth_method` パラメータを `md5+scram` から `scram-sha-256` に変更できます。

> **Warning**
>
> rds.accepted_password_auth_method を scram-sha-256 のみに変更した後、md5 で暗号化されたパスワードを持つすべてのユーザー (ロール) は接続できなくなります。

Aurora PostgreSQL DB クラスターに SCRAM を要求する準備

お使いの Aurora PostgreSQL DB クラスター、に変更を加える前に、既存のデータベースユーザーアカウントをすべて確認します。また、パスワードに使用されている暗号化の種類を確認してください。確認するためには、rds_tools 拡張機能を使用します。この拡張機能は、Aurora PostgreSQL 13.1 以上のリリースでサポートされています。

データベースユーザー (ロール) とパスワードの暗号化方法のリストを取得するには

1. 次のように、psql を使用して Aurora PostgreSQL DB クラスターのプライマリ インスタンスをに接続します。

   ```
   psql --host=cluster-name-instance-1.111122223333.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password
   ```

2. rds_tools 拡張機能をインストールします。

   ```
   postgres=> CREATE EXTENSION rds_tools;
   CREATE EXTENSION
   ```

3. ロールと暗号化のリストを取得します。

   ```
   postgres=> SELECT * FROM
         rds_tools.role_password_encryption_type();
   ```

   以下のような出力結果が表示されます。

   ```
          rolname        | encryption_type
   ----------------------+-----------------
    pg_monitor            |
    pg_read_all_settings  |
    pg_read_all_stats     |
    pg_stat_scan_tables   |
    pg_signal_backend     |
    lab_tester            | md5
   ```

```
 user_465             | md5
 postgres             | md5
(8 rows)
```

カスタム DB クラスターのパラメータグループの作成

> **Note**
>
> Aurora PostgreSQL DB クラスター で既にカスタムパラメータグループを使用している場合、新しいパラメータグループを作成する必要はありません。

Aurora のパラメータグループの概要については、「DB クラスターのパラメータグループの作成」を参照してください。

パスワードに使用されるパスワード暗号化タイプは、1 つのパラメータ password_encryption で設定します。Aurora PostgreSQL DB クラスター で許可される暗号化は、別のパラメータ rds.accepted_password_auth_method で設定されます。これらのいずれかをデフォルト値から変更するには、カスタム DB クラスターのパラメータグループ を作成して、クラスター に適用する必要があります。

また、AWS Management Console または RDS API を使用して、カスタムの DB クラスターのパラメータグループ を作成することもできます。詳細については、「DB クラスターのパラメータグループの作成」を参照してください。

これで、カスタムパラメータグループを DB インスタンスに関連付けることができます。

カスタム DB クラスターのパラメータグループを作成するには

1. create-db-cluster-parameter-group CLI コマンドを使用して、クラスターのカスタムパラメータグループを作成します。次の例では aurora-postgresql13 をこのカスタムパラメータグループのソースとして使用します。

   Linux、macOS、Unix の場合:

   ```
   aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-name 'docs-lab-scram-passwords' \
     --db-parameter-group-family aurora-postgresql13  --description 'Custom DB cluster parameter group for SCRAM'
   ```

Windows の場合:

```
aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-name "docs-lab-scram-passwords" ^
  --db-parameter-group-family aurora-postgresql13  --description "Custom DB cluster parameter group for SCRAM"
```

これで、カスタムパラメータグループをクラスターに関連付けることができます。

2. modify-db-cluster CLI コマンドを使用して、このカスタムパラメータグループを Aurora PostgreSQL DB クラスターに適用します。

   Linux、macOS、Unix の場合:

   ```
   aws rds modify-db-cluster --db-cluster-identifier 'your-instance-name' \
           --db-cluster-parameter-group-name "docs-lab-scram-passwords
   ```

   Windows の場合:

   ```
   aws rds modify-db-cluster --db-cluster-identifier "your-instance-name" ^
           --db-cluster-parameter-group-name "docs-lab-scram-passwords
   ```

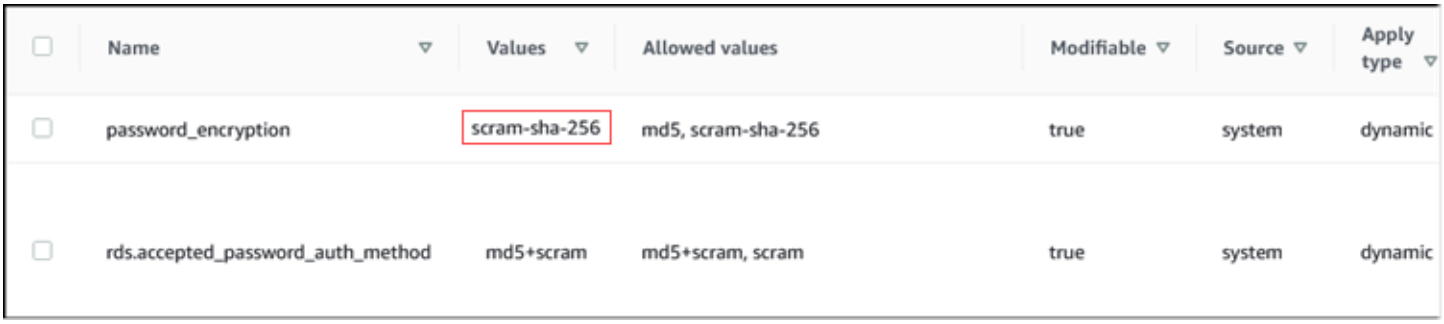
   Aurora PostgreSQL DB クラスターとカスタム DB クラスターパラメータグループ、を再同期するには、プライマリインスタンスおよびクラスターの他のすべてのインスタンスを再起動します。

SCRAM を使用するためのパスワード暗号化の設定

Aurora PostgreSQL DB クラスター で使用されるパスワード暗号化メカニズムは、password_encryption パラメータの DB クラスターのパラメータグループ に設定されています。指定できる値は、未設定、md5 または scram-sha-256 です。デフォルト値は、次のように Aurora PostgreSQL のバージョンによって異なります。

- Aurora PostgreSQL 14 — デフォルトは scram-sha-256
- Aurora PostgreSQL 13 — デフォルトは md5

Aurora PostgreSQL DB クラスター にアタッチされているカスタム DB クラスターパラメータグループ では、パスワード暗号化パラメータの値を変更できます。



Name	Values	Allowed values	Modifiable	Source	Apply type
password_encryption	scram-sha-256	md5, scram-sha-256	true	system	dynamic
rds.accepted_password_auth_method	md5+scram	md5+scram, scram	true	system	dynamic

パスワード暗号化の設定を scram-sha-256 に変更するには

- 次のように、パスワード暗号化の値を scram-sha-256 に設定します。パラメータが動的であるため、変更をすぐに適用できます。そのため、変更を有効にするために再起動は不要です。

  Linux、macOS、Unix の場合:

  ```
  aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name \
    'docs-lab-scram-passwords' --parameters
   'ParameterName=password_encryption,ParameterValue=scram-
  sha-256,ApplyMethod=immediate'
  ```

  Windows の場合:

  ```
  aws rds modify-db-parameter-group --db-parameter-group-name ^
    "docs-lab-scram-passwords" --parameters
   "ParameterName=password_encryption,ParameterValue=scram-
  sha-256,ApplyMethod=immediate"
  ```

ユーザーロールのパスワードを SCRAM に移行する

以下に説明するように、ユーザーロールのパスワードを SCRAM に移行できます。

データベースユーザー (ロール) のパスワードを MD5 から SCRAM に移行するには

1. 次のように、管理者ユーザーとしてログインします (デフォルトのユーザー名、`postgres`)。

   ```
   psql --host=cluster-name-instance-1.111122223333.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password
   ```

2. 次のコマンドを使って、RDS for PostgreSQL DB インスタンスの `password_encryption` パラメータの設定を確認します。

```
postgres=> SHOW password_encryption;
 password_encryption
---------------------
 md5
 (1 row)
```

3. このパラメータの値を scram-sha-256 に変更します。これは動的パラメータであるため、この変更を行った後でインスタンスを再起動する必要はありません。値をもう一度チェックして、次のように scram-sha-256 に設定されていることを確認します。

```
postgres=> SHOW password_encryption;
 password_encryption
---------------------
 scram-sha-256
 (1 row)
```

4. パスワードの変更をすべてのデータベースユーザーに通知します。アカウント postgres (rds_superuser 権限を持つデータベースユーザー) のパスワードも必ず変更してください。

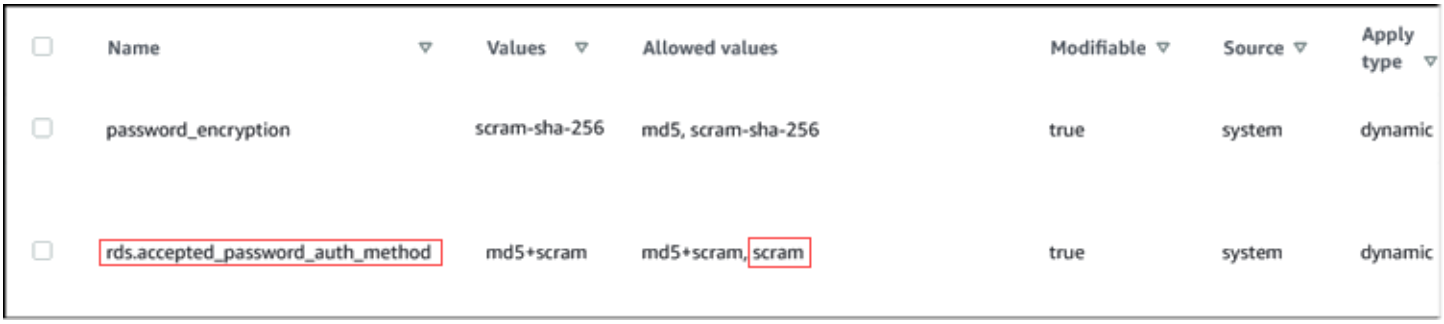
```
labdb=> ALTER ROLE postgres WITH LOGIN PASSWORD 'change_me';
ALTER ROLE
```

5. Aurora PostgreSQL DB クラスターのすべてのデータベースに対してこの処理を繰り返します。

SCRAM を要求するようにパラメータを変更する

これがプロセスの最後のステップです。次の手順で変更した後、パスワードに引き続き md5 暗号化を使用するユーザーアカウント (ロール) は Aurora PostgreSQL DB クラスターにログインできません。

rds.accepted_password_auth_method は、ログインプロセス中に Aurora PostgreSQL DB クラスターがユーザーパスワードに対して受け入れる暗号化方式を指定します。デフォルト値は md5+scram です。つまり、どちらの方法も受け入れられます。次の画像では、このパラメータのデフォルト設定が表示されています。



Name	Values	Allowed values	Modifiable	Source	Apply type
password_encryption	scram-sha-256	md5, scram-sha-256	true	system	dynamic
rds.accepted_password_auth_method	md5+scram	md5+scram, scram	true	system	dynamic

このパラメータに指定できる値は、md5+scram または scram のみです。このパラメータの値を scram に変更すると、これが要件となります。

パスワードの SCRAM 認証を要求するようにパラメータ値を変更するには

1. Aurora PostgreSQL DB クラスター の上のすべてのデータベースに対するすべてのデータベースユーザーパスワードが、パスワード暗号化に scram-sha-256 を使用していることを確認します。そのためには、rds_tools にロール (ユーザー) と暗号化タイプについて、次のようにクエリします。

```
postgres=> SELECT * FROM rds_tools.role_password_encryption_type();
  rolname        | encryption_type
  ---------------------+-----------------
  pg_monitor           |
  pg_read_all_settings |
  pg_read_all_stats    |
  pg_stat_scan_tables  |
  pg_signal_backend    |
  lab_tester           | scram-sha-256
  user_465             | scram-sha-256
  postgres             | scram-sha-256
  ( rows)
```

2. Aurora PostgreSQL DB クラスターのすべての DB インスタンスでクエリを繰り返します。

   すべてのパスワードで scram-sha-256 が使用されている場合は続行できます。

3. 次のように、受け入れたパスワード認証の値を scram-sha-256 に設定します。

   Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name 'docs-
lab-scram-passwords' \
```

```
  --parameters
 'ParameterName=rds.accepted_password_auth_method,ParameterValue=scram,ApplyMethod=immediat
```

Windows の場合:

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name "docs-
lab-scram-passwords" ^
  --parameters
 "ParameterName=rds.accepted_password_auth_method,ParameterValue=scram,ApplyMethod=immediat
```

SSL/TLS での Aurora PostgreSQL データの保護

Amazon RDS では、Aurora PostgreSQL DB クラスターの Secure Socket Layer (SSL) 暗号化と Transport Layer Security (TLS) 暗号化をサポートしています。SSL/TLS を使用して、アプリケーションと Aurora PostgreSQL DB クラスターとの接続を暗号化できます。また、Aurora PostgreSQL DB クラスターへのすべての接続に SSL/TLS の使用を強制することができます。Amazon Aurora PostgreSQL は、Transport Layer Security (TLS) バージョン 1.1 および 1.2 をサポートしています。暗号化された接続には TLS 1.2 を使用することをお勧めします。次の Aurora PostgreSQL のバージョンから TLSv1.3 のサポートが追加されました。

- 15.3 以降のすべてのバージョン
- 14.8 以降の 14 バージョン
- 13.11 以降の 13 バージョン
- 12.15 以降の 12 バージョン
- 11.20 以降の 11 バージョン

SSL/TLS サポートおよび PostgreSQL データベースの一般情報については、PostgreSQL ドキュメントの「SSL Support」を参照してください。JDBC を介した SSL/TLS 接続の使用については、PostgreSQL ドキュメントの「Configuring the Client」を参照してください。

トピック

- Aurora PostgreSQL DB クラスターへの SSL/TLS 接続を必須にする
- SSL/TLS 接続ステータスを確認する
- Aurora PostgreSQL DB クラスターへの接続用暗号スイートを設定する

Aurora PostgreSQL の SSL/TLS サポートは、すべての AWS リージョンで利用可能です。DB クラスターが作成される際、Amazon RDS により、Aurora PostgreSQL DB クラスター用の SSL/TLS 証明書が作成されます。SSL/TLS 証明書認証を有効にした場合、SSL/TLS 証明書には、なりすまし攻撃から保護するために、SSL/TLS 証明書の共通名 (CN) として DB クラスターのエンドポイントが含まれます。

SSL/TLS を使用して Aurora PostgreSQL DB クラスターに接続する方法

1. 証明書をダウンロードします。

   証明書のダウンロードについては、SSL/TLS を使用した DB クラスターへの接続の暗号化 を参照してください。
2. オペレーティングシステムに証明書をインポートします。
3. SSL/TLS を使用して Aurora PostgreSQL DB クラスターに接続します。

   SSL/TLS を使用して接続するとき、クライアントでは証明書チェーンを検証するかどうかを選択できます。接続パラメータで `sslmode=verify-ca` または `sslmode=verify-full` を指定すると、RDS CA 証明書が信頼ストアに存在するか、または接続 URL で参照されることをクライアントは要求します。この要求は、データベース証明書に署名する証明書チェーンを検証するためのものです。

   psql や JDBC など、クライアントに SSL/TLS サポートが設定されている場合、クライアントは初期にデフォルトで SSL/TLS を使用してデータベースへの接続を試みます。クライアントが SSL/TLS を使用して接続できない場合、SSL/TLS を使用しない接続に戻ります。デフォルトでは、JDBC および libpq ベースのクライアントの `sslmode` オプションは `prefer` に設定されています。

   `sslrootcert` パラメータを使用して証明書を参照します (`sslrootcert=rds-ssl-ca-cert.pem` など)。

psql を使用して Aurora PostgreSQL DB クラスターに接続する例は次のとおりです。

```
$ psql -h testpg.cdhmuqifdpib.us-east-1.rds.amazonaws.com -p 5432 \
    "dbname=testpg user=testuser sslrootcert=rds-ca-2015-root.pem sslmode=verify-full"
```

Aurora PostgreSQL DB クラスターへの SSL/TLS 接続を必須にする

rds.force_ssl パラメータを使用することで、Aurora PostgreSQL DB クラスターへの接続で SSL/TLS の使用を必須にすることができます。デフォルトでは、rds.force_ssl パラメータが 0 (オフ) に設定されています。rds.force_ssl パラメータを 1 (オン) に設定すれば、DB クラスターへの接続で SSL/TLS を必須にすることができます。rds.force_ssl パラメータを更新することでも PostgreSQL ssl パラメータは 1 (オン) に設定され、新しい SSL/TLS 設定をサポートするように DB クラスターの pg_hba.conf ファイルが変更されます。

rds.force_ssl パラメータの値は、DB クラスターの DB クラスターパラメータグループを更新することで設定できます。DB クラスターパラメータグループがデフォルトのものではなく、ssl パラメータを 1 に設定するときに rds.force_ssl パラメータが 1 に設定済みである場合は、DB クラスターを再起動する必要はありません。それ以外の場合は、変更を反映するために DB クラスターを再起動する必要があります。パラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。

DB クラスターで rds.force_ssl パラメータが 1 に設定されている場合、次のような出力が接続時に表示され、SSL/TLS が必須であることが示されます。

```
$ psql postgres -h SOMEHOST.amazonaws.com -p 8192 -U someuser
psql (9.3.12, server 9.4.4)
WARNING: psql major version 9.3, server major version 9.4.
Some psql features might not work.
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.

postgres=>
```

SSL/TLS 接続ステータスを確認する

接続の暗号化ステータスは、DB クラスターに接続するときにログオンバナーに表示されます。

```
Password for user master:
psql (9.3.12)
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.

postgres=>
```

また、sslinfo 拡張機能をロードしてから、ssl_is_used() 関数を呼び出して、SSL/TLS が使用されているかどうかを確認することもできます。この関数は、この接続が SSL/TLS を使用している場合に t を返し、それ以外の場合に f を返します。

```
postgres=> create extension sslinfo;
CREATE EXTENSION

postgres=> select ssl_is_used();
 ssl_is_used
---------
t
(1 row)
```

select ssl_cipher() コマンドを使用して、SSL/TLS 暗号を確認することができます。

```
postgres=> select ssl_cipher();
ssl_cipher
--------------------
DHE-RSA-AES256-SHA
(1 row)
```

set rds.force_ssl を有効にして DB クラスターを再起動すると、次のメッセージが表示され SSL 以外の接続は拒否されます。

```
$ export PGSSLMODE=disable
$ psql postgres -h SOMEHOST.amazonaws.com -p 8192 -U someuser
psql: FATAL: no pg_hba.conf entry for host "host.ip", user "someuser", database
 "postgres", SSL off
$
```

sslmode オプションの詳細については、PostgreSQL ドキュメントの「データベース接続制御関数」を参照してください。

Aurora PostgreSQL DB クラスターへの接続用暗号スイートを設定する

設定可能な暗号スイートを使用すると、データベース接続のセキュリティをより詳細に制御できます。データベースへのクライアント SSL/TLS 接続を保護するために許可する暗号スイートのリスト

を指定できます。設定可能な暗号スイートを使用すると、データベースサーバーが受け入れる接続暗号化を制御できます。これにより、安全でない暗号や非推奨の暗号の使用を防ぐことができます。

設定可能な暗号スイートは、Aurora PostgreSQL バージョン 11.8 以降でサポートされています。

接続を暗号化するために許容できる暗号のリストを指定するには、`ssl_ciphers` クラスターパラメータを変更します。AWS Management Console、AWS CLI、または RDS API を使用して、`ssl_ciphers` パラメータをクラスターパラメータグループのカンマ区切りの暗号文字列に設定します。クラスターパラメータを設定するには、「DB クラスターパラメータグループのパラメータの変更」を参照してください。

次の表は、有効な Aurora PostgreSQL エンジンバージョンでサポートされる暗号を示しています。

Aurora PostgreSQL のエンジンバージョン	サポートされる暗号
9.6、10.20 以前、11.15 以前、12.10 以前、13.6 以前	• DHE-RSA-AES128-SHA • DHE-RSA-AES128-SHA256 • DHE-RSA-AES128-GCM-SHA256 • DHE-RSA-AES256-SHA • DHE-RSA-AES256-SHA256 • DHE-RSA-AES256-GCM-SHA384 • ECDHE-ECDSA-AES256-SHA • ECDHE-ECDSA-AES256-GCM-SHA384 • ECDHE-RSA-AES256-SHA384 • ECDHE-RSA-AES128-SHA • ECDHE-RSA-AES128-SHA256 • ECDHE-RSA-AES128-GCM-SHA256 • ECDHE-RSA-AES256-SHA • ECDHE-RSA-AES256-GCM-SHA384
10.21、11.16、12.11、13.7、14.3、14.4	• DHE-RSA-AES128-SHA • DHE-RSA-AES128-SHA256

Aurora PostgreSQL のエンジンバージョン	サポートされる暗号
	• DHE-RSA-AES128-GCM-SHA256 • DHE-RSA-AES256-SHA • DHE-RSA-AES256-SHA256 • DHE-RSA-AES256-GCM-SHA384 • ECDHE-ECDSA-AES256-SHA • ECDHE-ECDSA-AES256-GCM-SHA384 • ECDHE-RSA-AES256-SHA384 • ECDHE-RSA-AES128-SHA • ECDHE-RSA-AES128-GCM-SHA256 • ECDHE-RSA-AES256-SHA • ECDHE-RSA-AES256-GCM-SHA384 • TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA • TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 • TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA • TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 • TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA • TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 • TLS_RSA_WITH_AES_256_GCM_SHA384 • TLS_RSA_WITH_AES_256_CBC_SHA

Aurora PostgreSQL のエンジンバージョン	サポートされる暗号
	• TLS_RSA_WITH_AES_128_GCM_SHA256 • TLS_RSA_WITH_AES_128_CBC_SHA • TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256

Aurora PostgreSQL のエンジンバージョン	サポートされる暗号
10.22 以降、11.17 以降、12.12 以降、13.8 以降、14.5 以降、15.2 以降	• DHE-RSA-AES128-SHA • DHE-RSA-AES128-SHA256 • DHE-RSA-AES128-GCM-SHA256 • DHE-RSA-AES256-SHA • DHE-RSA-AES256-SHA256 • DHE-RSA-AES256-GCM-SHA384 • ECDHE-ECDSA-AES256-SHA • ECDHE-ECDSA-AES256-GCM-SHA384 • ECDHE-RSA-AES256-SHA384 • ECDHE-RSA-AES128-SHA • ECDHE-RSA-AES128-SHA256 • ECDHE-RSA-AES128-GCM-SHA256 • ECDHE-RSA-AES256-SHA • ECDHE-RSA-AES256-GCM-SHA384 • TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA • TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 • TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA • TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 • TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 • TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA

Aurora PostgreSQL のエンジンバージョン	サポートされる暗号
	• TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 • TLS_RSA_WITH_AES_256_GCM_SHA384 • TLS_RSA_WITH_AES_256_CBC_SHA • TLS_RSA_WITH_AES_128_GCM_SHA256 • TLS_RSA_WITH_AES_128_CBC_SHA256 • TLS_RSA_WITH_AES_128_CBC_SHA • TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256

Aurora PostgreSQL のエンジンバージョン	サポートされる暗号
15.3、14.8、13.11、12.15、11.20	• DHE-RSA-AES128-SHA • DHE-RSA-AES128-SHA256 • DHE-RSA-AES128-GCM-SHA256 • DHE-RSA-AES256-SHA • DHE-RSA-AES256-SHA256 • DHE-RSA-AES256-GCM-SHA384 • ECDHE-ECDSA-AES256-SHA • ECDHE-ECDSA-AES256-GCM-SHA384 • ECDHE-RSA-AES256-SHA384 • ECDHE-RSA-AES128-SHA • ECDHE-RSA-AES128-SHA256 • ECDHE-RSA-AES128-GCM-SHA256 • ECDHE-RSA-AES256-SHA • ECDHE-RSA-AES256-GCM-SHA384 • TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA • TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 • TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA • TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 • TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 • TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA

Aurora PostgreSQL のエンジンバージョン	サポートされる暗号
	• TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 • TLS_RSA_WITH_AES_256_GCM_SHA384 • TLS_RSA_WITH_AES_256_CBC_SHA • TLS_RSA_WITH_AES_128_GCM_SHA256 • TLS_RSA_WITH_AES_128_CBC_SHA256 • TLS_RSA_WITH_AES_128_CBC_SHA • TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 • TLS_AES_128_GCM_SHA256 • TLS_AES_256_GCM_SHA384

また、CLI コマンドの describe-engine-default-cluster-parameters を使用して、特定のパラメータグループファミリーで現在サポートされている暗号スイートを特定することもできます。次の例では、Aurora PostgreSQL 11 の `ssl_cipher` クラスターパラメータで許可される値を取得する方法を示しています。

```
aws rds describe-engine-default-cluster-parameters --db-parameter-group-family aurora-
postgresql11

    ...some output truncated...
 {
  "ParameterName": "ssl_ciphers",
  "Description": "Sets the list of allowed TLS ciphers to be used on secure
 connections.",
  "Source": "engine-default",
  "ApplyType": "dynamic",
  "DataType": "list",
  "AllowedValues": "DHE-RSA-AES128-SHA,DHE-RSA-AES128-SHA256,DHE-RSA-AES128-GCM-
SHA256,DHE-RSA-AES256-SHA,DHE-RSA-AES256-SHA256,DHE-RSA-AES256-GCM-SHA384,
```

```
  ECDHE-RSA-AES128-SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-AES128-GCM-
SHA256,ECDHE-RSA-AES256-SHA,ECDHE-RSA-AES256-SHA384,ECDHE-RSA-AES256-GCM-
SHA384,TLS_RSA_WITH_AES_256_GCM_SHA384,

 TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_128_CBC_SHA256,T

 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AE
  "IsModifiable": true,
  "MinimumEngineVersion": "11.8",
  "SupportedEngineModes": [
   "provisioned"
  ]
 },
    ...some output truncated...
```

ssl_ciphers パラメータのデフォルトは、許可されているすべての暗号スイートです。暗号の詳細については、PostgreSQL ドキュメントに記載されている変数 ssl_ciphers を参照してください。

新しい SSL/TLS 証明書を使用して Aurora PostgreSQL DB クラスターに接続するようにアプリケーションを更新する

2023 年 1 月 13 日に Amazon RDS は、Secure Socket Layer または Transport Layer Security（SSL/TLS）を使用して Aurora DB クラスターに接続するための新しい認証局（CA）証明書を公開しました。ここでは、新しい証明書を使用するためのアプリケーションの更新について説明します。

このトピックでは、クライアントアプリケーションが SSL/TLS を使用して DB クラスターに接続されているかどうかを判断できます。該当する場合はさらに、それらのアプリケーションの接続に証明書の検証が必要かどうかを確認できます。

> **Note**
>
> 一部のアプリケーションは、サーバー上の証明書を正常に検証できる場合にのみ、Aurora PostgreSQL DB クラスターに接続されるように設定されています。
> そのようなアプリケーションの場合は、新しい CA 証明書を含むように、クライアントアプリケーションの信頼ストアを更新する必要があります。

クライアントアプリケーションの信頼ストアで CA 証明書を更新した後、DB クラスターで証明書をローテーションできます。これらの手順を開発環境またはステージング環境でテストしてから、本番環境で実装することを強くお勧めします。

証明書のローテーションの詳細については、「SSL/TLS 証明書のローテーション」を参照してください。証明書のダウンロードの詳細については、SSL/TLS を使用した DB クラスターへの接続の暗号化 を参照してください。PostgreSQL DB クラスターで SSL/TLS を使用する方法については、「SSL/TLS での Aurora PostgreSQL データの保護」を参照してください。

トピック

- アプリケーションが SSL を使用して Aurora PostgreSQL DB クラスターに接続しているかどうかの確認
- クライアントが接続するために証明書の検証を必要とするかどうかの確認
- アプリケーション信頼ストアの更新
- 各種アプリケーションでの SSL/TLS 接続の使用

アプリケーションが SSL を使用して Aurora PostgreSQL DB クラスターに接続しているかどうかの確認

DB クラスターの設定で `rds.force_ssl` パラメータの値を確認します。デフォルトでは、`rds.force_ssl` パラメータは `0` (オフ) に設定されます。`rds.force_ssl` パラメータが `1` (オン) に設定されている場合、クライアントは接続に SSL/TLS を使用する必要があります。パラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。

`rds.force_ssl` が `1` (オン) に設定されていない場合は、`pg_stat_ssl` ビューにクエリを発行して、SSL を使用する接続を確認します。例えば次のクエリは、SSL 接続、および SSL を使用するクライアントに関する情報のみを返します。

```
select datname, usename, ssl, client_addr from pg_stat_ssl inner join pg_stat_activity
 on pg_stat_ssl.pid = pg_stat_activity.pid where ssl is true and usename<>'rdsadmin';
```

SSL/TLS 接続を使用する行のみが、接続に関する情報とともに表示されます。以下は出力例です。

```
 datname  | usename | ssl | client_addr
----------+---------+-----+-------------
 benchdb  | pgadmin | t   | 53.95.6.13
 postgres | pgadmin | t   | 53.95.6.13
(2 rows)
```

上記のクエリでは、クエリ実行時点の接続のみが表示されます。結果が表示されなくても、SSL 接続を使用しているアプリケーションが存在しないわけではありません。それ以降に別の SSL 接続が確立される場合もあります。

クライアントが接続するために証明書の検証を必要とするかどうかの確認

psql や JDBC など、クライアントに SSL サポートが設定されている場合、クライアントは初期にデフォルトで SSL を使用してデータベースへの接続を試みます。クライアントが SSL を使用して接続できない場合、SSL を使用しない接続に戻ります。使用されるデフォルトの `sslmode` モードは、libpq ベースのクライアント (psql など) と JDBC では異なります。libpq ベースのクライアントはデフォルトで `prefer` に設定されますが、JDBC クライアントは `verify-full` に設定されます。サーバー上の証明書が検証されるのは、`sslmode` が `verify-ca` または `verify-full` に設定されて、`sslrootcert` が指定されている場合のみです。証明書が無効な場合は、エラーがスローされます。

PGSSLROOTCERT を使用して、PGSSLMODE 環境変数で証明書を検証します (PGSSLMODE は `verify-ca` または `verify-full` に設定)。

```
PGSSLMODE=verify-full PGSSLROOTCERT=/fullpath/ssl-cert.pem psql -h
 pgdbidentifier.cxxxxxxxx.us-east-2.rds.amazonaws.com -U primaryuser -d postgres
```

`sslrootcert` 引数を使用して、接続文字列形式で `sslmode` を使用して証明書を検証します (`sslmode` は `verify-ca` または `verify-full` に設定) 。

```
psql "host=pgdbidentifier.cxxxxxxxx.us-east-2.rds.amazonaws.com sslmode=verify-full
 sslrootcert=/full/path/ssl-cert.pem user=primaryuser dbname=postgres"
```

たとえば前述の例の場合は、無効なルート証明書を使用すると、クライアントで次のようなエラーが表示されます。

```
psql: SSL error: certificate verify failed
```

アプリケーション信頼ストアの更新

PostgreSQL アプリケーション信頼ストアの更新については、PostgreSQL ドキュメントの「Secure TCP/IP Connections with SSL」を参照してください。

> **Note**
>
> 信頼ストアを更新するとき、新しい証明書を追加できるだけでなく、古い証明書を保持できます。

JDBC のアプリケーション信頼ストアの更新

SSL/TLS 接続に JDBC を使用するアプリケーションの信頼ストアを更新できます。

ルート証明書のダウンロードについては、SSL/TLS を使用した DB クラスターへの接続の暗号化 を参照してください。

証明書をインポートするサンプルスクリプトについては、証明書を信頼ストアにインポートするためのサンプルスクリプト を参照してください。

各種アプリケーションでの SSL/TLS 接続の使用

以下に、各種アプリケーションでの SSL/TLS 接続の使用に関する情報を示します。

- psql

  接続文字列または環境可変としてオプションを指定することで、クライアントがコマンドラインから呼び出されます。SSL/TLS 接続の場合、関連するオプションは `sslmode` (環境可変 `PGSSLMODE`)、`sslrootcert` (環境可変 `PGSSLROOTCERT`) です。

  オプションの詳細なリストについては、PostgreSQL ドキュメントの「Parameter Key Words」を参照してください。環境可変の詳細なリストについては、PostgreSQL ドキュメントの「Environment Variables」を参照してください。

- pgAdmin

  このブラウザベースのクライアントは、PostgreSQL データベースに接続するために使用できる使いやすいインターフェイスです。

  接続の設定については、pgAdmin のドキュメントを参照してください。

- JDBC

  JDBC は、Java アプリケーションとのデータベース接続を可能にします。

JDBC を使用した PostgreSQL データベースへの接続に関する一般情報については、PostgreSQL ドキュメントの「Connecting to the Database」を参照してください。SSL/TLS を使用した接続については、PostgreSQL ドキュメントの「Configuring the client」を参照してください。

- Python

  PostgreSQL データベースに接続するために一般的に使用される Python ライブラリは、`psycopg2` です。

  `psycopg2` の使用については、psycopg2 のドキュメントを参照してください。PostgreSQL データベースへの接続方法に関する簡単なチュートリアルについては、「Psycopg2 Tutorial」を参照してください。connect コマンドで受け入れられるオプションについては、「The psycopg2 module content」を参照してください。

> **Important**
>
> データベース接続で SSL/TLS を使用することを決定し、アプリケーションの信頼ストアを更新したら、rds-ca-rsa2048-g1 証明書を使用するようにデータベースを更新できます。ステップについては、「DB インスタンスの変更による CA 証明書の更新」のステップ 3 を参照してください。

Aurora PostgreSQL で Kerberos 認証を使用する

ユーザーが PostgreSQL が実行されている DB クラスターに接続する場合、Kerberos を使用してそのユーザーを認証できます。そのためには、Kerberos 認証に AWS Directory Service for Microsoft Active Directory を使用するように DB クラスターを設定します。AWS Directory Service for Microsoft Active Directory は AWS Managed Microsoft AD とも呼ばれます。これは、AWS Directory Service で利用できる機能です。詳細については、「AWS Directory Service 管理ガイド」の「AWS Directory Service とは」を参照してください。

まず、ユーザー認証情報を格納する AWS Managed Microsoft AD ディレクトリを作成します。次に、Active Directory のドメインおよびその他の情報を PostgreSQL DB クラスターに提供します。ユーザーが PostgreSQL DB クラスターを使用して認証を実行すると、認証要求は AWS Managed Microsoft AD ディレクトリに転送されます。

同じディレクトリにすべての認証情報を保持することで時間と労力を節約できます。複数の DB クラスターの認証情報を一元的に保存および管理できます。また、ディレクトリを使用することで、セキュリティプロファイル全体を向上できます。

また、独自のオンプレミスの Microsoft Active Directory から認証情報にアクセスできます。そのためには、信頼するドメイン関係を作成して、AWS Managed Microsoft AD ディレクトリがオンプレミスの Microsoft Active Directory を信頼するようにします。これにより、ユーザーは、オンプレミスネットワークのワークロードにアクセスするときと同じ Windows シングルサインオン (SSO) の使い方で、PostgreSQL クラスターにアクセスできます。

データベースは Kerberos、AWS Identity and Access Management (IAM)、または Kerberos 認証と IAM 認証の両方。ただし、Kerberos 認証と IAM 認証では異なる認証方法が提供されるため、特定のユーザーは、一方の認証方法のみを使用してデータベースにログインできますが、両方を使用することはできません。IAM 認証の詳細については、「 の IAM データベース認証」を参照してください。

トピック

- リージョンとバージョンの可用性
- PostgreSQL DB クラスターの Kerberos 認証の概要
- PostgreSQL DB クラスターの Kerberos 認証のセットアップ
- ドメイン内の DB クラスターの管理
- PostgreSQL を Kerberos 認証と接続する
- Aurora PostgreSQL アクセスコントロールへの AD セキュリティグループの使用

リージョンとバージョンの可用性

機能の可用性とサポートは、各データベースエンジンの特定のバージョン、および AWS リージョンによって異なります。Kerberos 認証で Aurora PostgreSQL を使用する場合、利用できるバージョンとリージョンの詳細については、「Aurora PostgreSQL で Kerberos 認証を使用する」を参照してください。

PostgreSQL DB クラスターの Kerberos 認証の概要

PostgreSQL DB クラスターに Kerberos 認証を設定するには、以下で説明するステップを実行します。

1. AWS Managed Microsoft AD を使用して AWS Managed Microsoft AD ディレクトリを作成します。AWS Management Console、AWS CLI、AWS Directory Service API を使用して、ディレクト

リを作成できます。ディレクトリがクラスターと通信できるように、ディレクトリセキュリティグループで関連するアウトバウンドポートを必ず開いてください。

2. AWS Managed Microsoft AD ディレクトリを呼び出すためのアクセスを Amazon Aurora に許可するロールを作成します。これにより、マネージド IAM ポリシー `AmazonRDSDirectoryServiceAccess` を使用する AWS Identity and Access Management (IAM) ロールが作成されます。

   IAM ロールによるアクセスを許可するには、AWS Security Token Service (AWS STS) エンドポイントを AWS アカウントの AWS リージョンでアクティベートする必要があります。AWS STS エンドポイントはすべての AWS リージョン でデフォルトでアクティブになっているため、他のアクションを実行せずに、エンドポイントを使用することができます。詳細については、IAM ユーザーガイドの「AWS STS リージョンでの AWS のアクティブ化と非アクティブ化」を参照してください。

3. Microsoft Active Directory のツールを使用して、AWS Managed Microsoft AD ディレクトリでユーザーとグループを作成し、設定します。Active Directory にユーザーを作成する方法の詳細については、AWS 管理ガイドの「AWS Directory Service マネージド Microsoft AD でユーザーとグループを管理する」を参照してください。

4. 異なる AWS アカウントまたは Virtual Private Cloud (VPC) 内にディレクトリおよび DB インスタンスを配置する場合は、VPC ピア接続を設定します。詳細については、Amazon VPC Peering Guideの「VPC ピア機能とは」を参照してください。

5. 以下のいずれかの方法を使用して、コンソール、CLI、RDS API から PostgreSQL DB クラスターを作成または変更します。

   - Aurora PostgreSQL DB クラスターの作成と接続
   - Amazon Aurora DB クラスターの変更
   - DB クラスターのスナップショットからの復元
   - DB クラスターを指定の時点の状態に復元する

   クラスターは、ディレクトリと同じ Amazon Virtual Private Cloud (VPC)、または別の AWS アカウントまたは VPC にあります。PostgreSQL DB クラスターの作成または変更時に、次のステップを行います。

   - ディレクトリの作成時に、生成されたドメイン識別子 (d-* 識別子) を指定します。
   - 作成した IAM ロール名を指定します。
   - DB インスタンスのセキュリティグループが、ディレクトリのセキュリティグループからインバウンドトラフィックを受信できることを確認します。

6. RDS マスターユーザー認証情報を使用して、PostgreSQL DB クラスターインスタンスに接続します。外部で識別されるように PostgreSQL でユーザーを作成します。外部で識別されたユーザーは、Kerberos 認証を使用して PostgreSQL DB クラスターにログインできます。

PostgreSQL DB クラスターの Kerberos 認証のセットアップ

AWS Directory Service for Microsoft Active Directory (AWS Managed Microsoft AD) を使用して、PostgreSQL DB クラスターに Kerberos 認証をセットアップします。Kerberos 認証をセットアップするには、次のステップに従います。

トピック

ステップ 1: AWS Managed Microsoft AD を使用してディレクトリを作成する

AWS Directory Service はフルマネージド型の Active Directory を AWS クラウド内に作成します。AWS Managed Microsoft AD ディレクトリを作成すると、AWS Directory Service が 2 つのドメインコントローラーと DNS サーバーを作成します。ディレクトリサーバーは、VPC 内の異なるサブネットで作成されます。この冗長性によって、障害が発生してもディレクトリにアクセス可能な状態を維持できます。

AWS Managed Microsoft AD ディレクトリを作成すると、AWS Directory Service がユーザーに代わって次のタスクを実行します。

- VPC 内に Active Directory を設定します。
- ユーザー名 `Admin` と指定されたパスワードを使用してディレクトリ管理者アカウントを作成します。このアカウントを使用してディレクトリを管理します。

> **Important**
>
> このパスワードは必ず保管してください。AWS Directory Service にはこのパスワードは保存されず、復元やリセットもできません。

- ディレクトリコントローラー用セキュリティグループを作成します。セキュリティグループは、PostgreSQL DB クラスターとの通信を許可する必要があります。

AWS Directory Service for Microsoft Active Directory を起動すると、AWS は組織単位 (OU) を作成します。OU にはディレクトリのオブジェクトがすべて含まれています。この OU はドメインルートにあります。OU にはディレクトリを作成する際に入力した NetBIOS 名があります。ドメインルートは AWS が所有し、管理します。

Admin ディレクトリに作成された AWS Managed Microsoft AD アカウントには、OU に対して頻繁に実行される管理行為の権限が含まれています。

- ユーザーを作成、更新、削除する
- ファイルやプリントサーバーなどのドメインにリソースを追加して、追加したリソースへのアクセス許可を OU のユーザーとグループに割り当てる
- 追加の OU やコンテナを作成する
- 権限を委譲する
- 削除されたオブジェクトを Active Directory のごみ箱から元に戻す
- Active Directory Web Service で Windows PowerShell 用の Active Directory と Domain Name Service (DNS) モジュールを実行する。

Admin アカウントには、ドメイン全体に関係するアクティビティを実行する権限もあります。

- DNS 設定 (レコード、ゾーン、フォワーダーの追加、削除、更新) を管理する
- DNS イベントログを参照する
- セキュリティイベントログを参照する

AWS Managed Microsoft AD でディレクトリを作成するには

1. AWS Directory Serviceコンソールのナビゲーションペインで、[ディレクトリ]、[ディレクトリのセットアップ] の順に選択します。

2. AWS Managed Microsoft AD を選択します。現在、 Amazon Aurora での使用では AWS Managed Microsoft AD のオプションのみがサポートされています。
3. [Next] を選択します。
4. [ディレクトリ情報の入力] ページに、以下の情報を指定します。

   エディション

   目的の要件を満たすエディションを選択します。

   ディレクトリの DNS 名

   ディレクトリの完全修飾名 (例: **corp.example.com**)。

   ディレクトリの NetBIOS 名

   ディレクトリの短縮名 (例: CORP)。

   ディレクトリの説明

   必要に応じて、ディレクトリの説明。

   管理者パスワード

   ディレクトリ管理者のパスワードです。ディレクトリの作成プロセスでは、ユーザー名 Admin とこのパスワードを使用して管理者アカウントが作成されます。

   ディレクトリ管理者のパスワードには、「admin」の単語を含めることはできません。パスワードは大文字と小文字を区別し、8-64 文字にします。また、以下の 4 つのカテゴリうち 3 つから少なくとも 1 文字を含める必要があります。

   - 小文字 (a ~ z)
   - 大文字 (A ~ Z)
   - 数字 (0 ~ 9)
   - 英数字以外の文字 (~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/)

   パスワードを確認

   管理者のパスワードをもう一度入力します。

   > **Important**
   >
   > このパスワードは必ず保管してください。AWS Directory Service にはこのパスワードは保存されず、復元やリセットもできません。

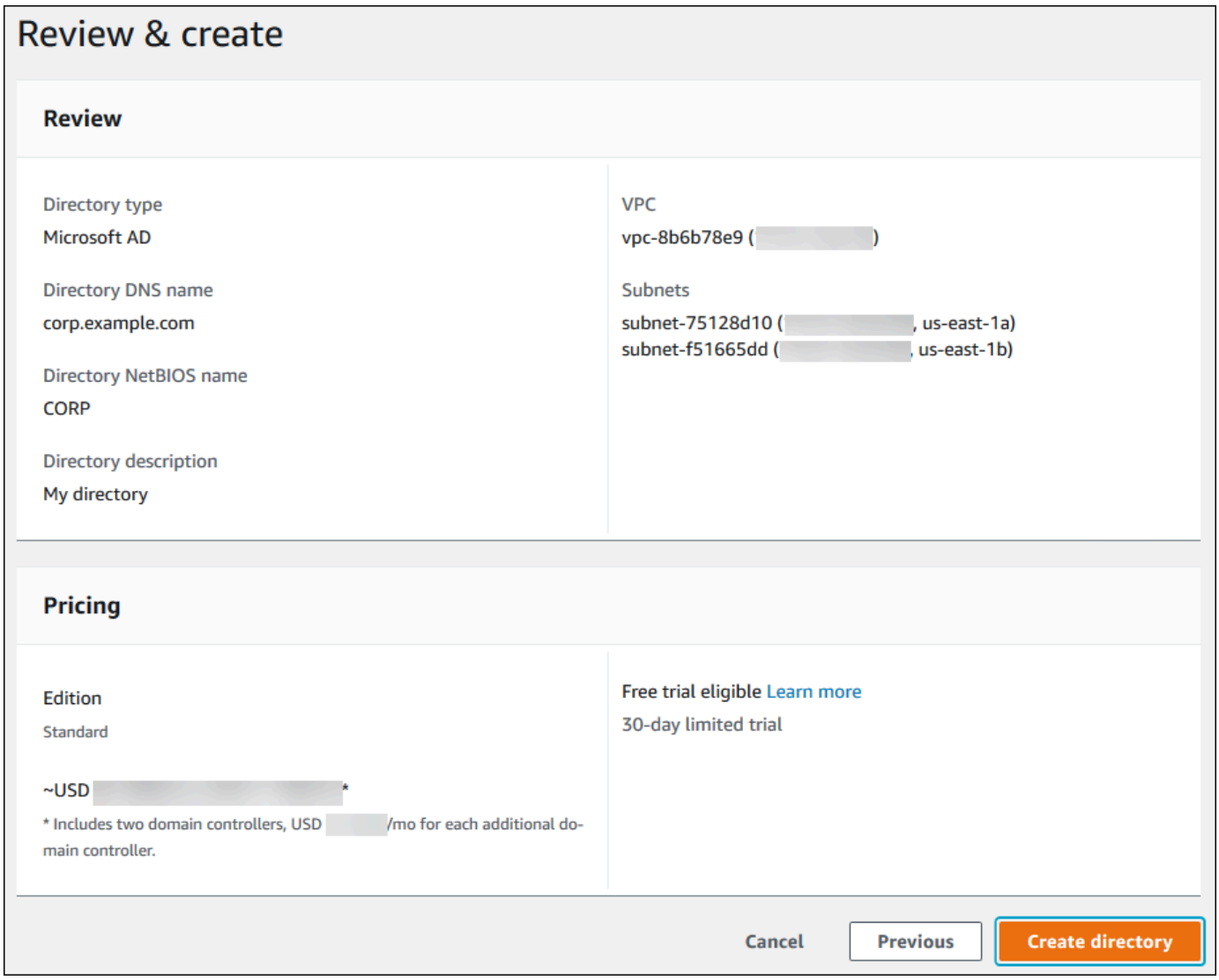
5. [Next] を選択します。
6. [VPC とサブネットの選択] ページで、以下の情報を指定します。

   VPC

   ディレクトリ用の VPC を選択します。PostgreSQL DB クラスターは、この同じ VPC または異なる VPC で作成できます。

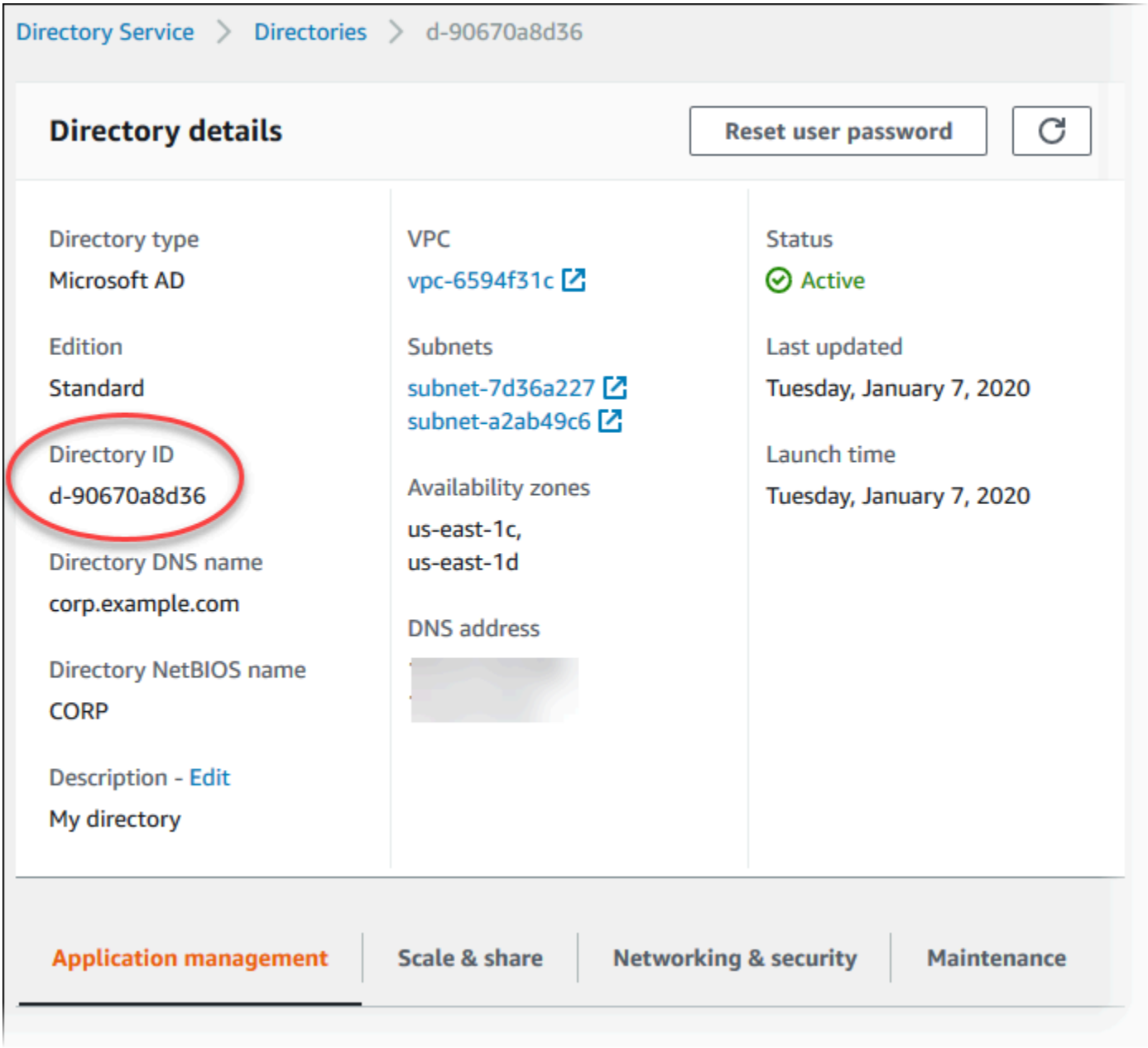
   Subnets

   ディレクトリサーバーのサブネットを選択します。2 つのサブネットは、異なるアベイラビリティーゾーンに存在している必要があります。
7. [Next] を選択します。
8. ディレクトリの情報を確認します。変更が必要な場合は、[戻る] を選択し、変更を行います。情報が正しい場合は、[Create directory (ディレクトリの作成)] を選択します。



ディレクトリが作成されるまで、数分かかります。正常に作成されると、[Status] 値が [Active] に変わります。

ディレクトリに関する情報を表示するには、ディレクトリの一覧で、そのディレクトリ ID を選択します。ディレクトリ ID 値を書き留めます。PostgreSQL DB インスタンスを作成または変更する場合は、この値が必要です。



ステップ 2: (オプション) オンプレミスの Active Directory と AWS Directory Service との間の信頼関係を作成する

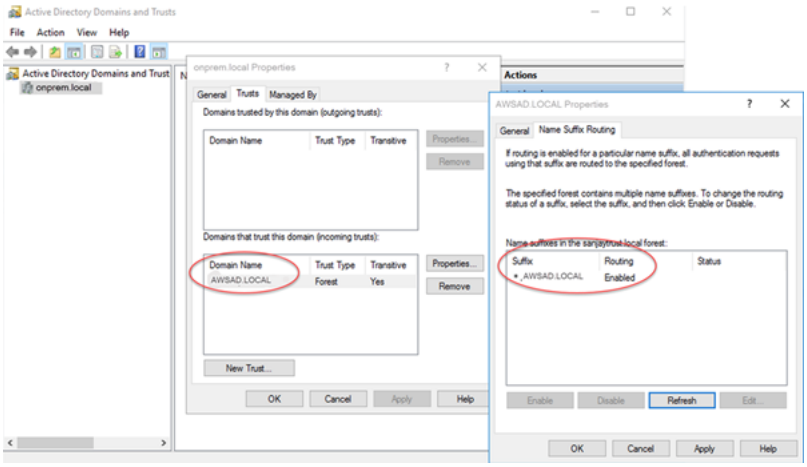
独自のオンプレミスの Microsoft Active Directory を使用する予定がない場合は、ステップ 3: Amazon Aurora が AWS Directory Service にアクセスするための IAM ロールを作成する に進みます。

オンプレミスの Active Directory を使用して Kerberos 認証を取得するには、オンプレミスの Microsoft Active Directory と (AWS Managed Microsoft AD で作成された) ステップ 1: AWS Managed Microsoft AD を使用してディレクトリを作成する ディレクトリとの間に、フォレストの信頼を使用して、信頼するドメイン関係を作成する必要があります。信頼は一方向にすることができます。こ

の場合、AWS Managed Microsoft AD ディレクトリはオンプレミスの Microsoft Active Directory を信頼します。信頼は、両方の Active Directory が相互に信頼する双方向にすることもできます。AWS Directory Service を使用して信頼関係を設定する方法の詳細については、「AWS Directory Service 管理ガイド」の「信頼関係を作成する場合」を参照してください。

> **Note**
>
> オンプレミスの Microsoft Active Directory を使用している場合:
>
> - Windows クライアントは、rds.amazonaws.com ではなく、エンドポイントの AWS Directory Service のドメイン名を使用して接続する必要があります。詳細については、「PostgreSQL を Kerberos 認証と接続する」を参照してください。
> - Windows クライアントは、Aurora カスタムエンドポイントを使用すると接続できません。詳細については、「Amazon Aurora 接続管理」を参照してください。
> - グローバルデータベースの場合:
>   - Windows クライアントは、グローバルデータベースのプライマリ AWS リージョン リージョンにあるインスタンスエンドポイントまたはクラスターエンドポイントを使用する場合に限り接続できます。
>   - Windows クライアントは、セカンダリ AWS リージョン のクラスターエンドポイントを使用して接続できません。

オンプレミスの Microsoft Active Directory ドメイン名に、新しく作成された信頼関係に対応する DNS サフィックスルーティングが含まれていることを確認してください。次のスクリーンショットは、例を示しています。



ステップ 3: Amazon Aurora が AWS Directory Service にアクセスするための IAM ロールを作成する

Amazon Aurora が AWS Directory Service を呼び出すには、AWS アカウントにマネージド IAM ポリシー `AmazonRDSDirectoryServiceAccess` を使用する IAM ロールが必要です。このロールにより、Amazon Aurora は AWS Directory Service を呼び出すことが可能になります。(AWS Directory Service にアクセスするためのこの IAM ロールは、 の IAM データベース認証 に使用される IAM ロールとは異なることに注意してください)

AWS Management Console を使用して DB インスタンスを作成し、コンソールユーザーが `iam:CreateRole` アクセス許可を持っている場合、コンソールは必要な IAM ロールを自動的に作成します。この場合、ロール名は `rds-directoryservice-kerberos-access-role` です。それ以外の場合は、IAM ロールを手動で作成する必要があります。IAM ロールを作成する場合、[`Directory Service`] を選択し、それに AWS マネージドポリシー `AmazonRDSDirectoryServiceAccess` をアタッチします。

サービス用の IAM ロールを作成する方法の詳細については、「IAM ユーザーガイド」の「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。

> **Note**
>
> RDS for Microsoft SQL Server の Windows 認証に使用される IAM ロールは、Amazon Aurora に使用できません。

`AmazonRDSDirectoryServiceAccess` マネージドポリシーを使用する代わりに、必要なアクセス許可を使用してポリシーを作成することもできます。これを行うには、IAM ロールに次の IAM 信頼ポリシーが必要です。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.amazonaws.com",
          "rds.amazonaws.com"
```

```
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

また、ロールには、以下の IAM ロールポリシーも必要です。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ds:DescribeDirectories",
        "ds:AuthorizeApplication",
        "ds:UnauthorizeApplication",
        "ds:GetAuthorizedApplicationDetails"
      ],
    "Effect": "Allow",
    "Resource": "*"
    }
  ]
}
```

ステップ 4: ユーザーを作成して設定する

Active Directory ユーザーとコンピューターツールを使用してユーザーを作成できます。これは Active Directory Domain Services ツールおよび Active Directory Lightweight Directory Services ツールの 1 つです。詳細については、Microsoft のドキュメントの「ユーザーとコンピュータを Active Directory ドメインに追加する」を参照してください。この場合、ユーザーは個人またはその他のエンティティです。例えば、ドメインの一部であり、その ID がディレクトリで管理されているコンピュータなどです。

AWS Directory Service ディレクトリにユーザーを作成するには、AWS Directory Service ディレクトリのメンバーである Windows ベースの Amazon EC2 インスタンスに接続している必要があります。同時に、ユーザーを作成する権限を持つユーザーとしてログインしていなければなりません。詳細については、AWS Directory Service 管理ガイドの「ユーザーの作成」を参照してください。

ステップ 5: ディレクトリと DB インスタンスの間のクロス VPC トラフィックを有効にする

同じ VPC 内にディレクトリおよび DB クラスターを配置する場合は、このステップをスキップして ステップ 6: PostgreSQL DB クラスターを作成または変更する に進みます。

ディレクトリと DB インスタンスを別の VPC に配置する場合は、VPC ピア接続または AWS Transit Gateway を使用してクロス VPC トラフィックを設定します。

次の手順では、VPC ピア接続を使用して VPC 間のトラフィックを有効にします。Amazon Virtual Private Cloud ピアリング接続ガイドの「VPC ピア機能とは」の手順に従います。

VPC ピア接続を使用してクロス VPC トラフィックを有効にするには

1. 適切な VPC ルーティングを設定し、ネットワークトラフィックが双方向にフローするようにします。
2. DB インスタンスのセキュリティグループが、ディレクトリのセキュリティグループからインバウンドトラフィックを受信できることを確認します。
3. トラフィックをブロックするネットワークのアクセス制御リスト (ACL) ルールがないことを確認します。

別の AWS アカウントがディレクトリを所有している場合は、ディレクトリを共有する必要があります。

AWS アカウント間でディレクトリを共有するには

1. DB インスタンスを作成する AWS アカウントとの間でディレクトリの共有をスタートするには、AWS 管理ガイドの「チュートリアル: AWS Directory Service マネージド Microsoft AD ディレクトリを共有して、シームレスに EC2 ドメインを結合する」の手順を実行します。
2. DB インスタンスのアカウントを使用して、AWS Directory Service コンソールにサインインし、続行する前にドメインが必ず SHARED ステータスであることを確認します。
3. DB インスタンスのアカウントを使用して AWS Directory Service コンソールにサインインしている間に、[ディレクトリ ID] の値を書き留めておきます。このディレクトリ ID は、DB インスタンスをドメインに結合するために使用します。

ステップ 6: PostgreSQL DB クラスターを作成または変更する

ディレクトリで使用する PostgreSQL DB クラスターを作成または変更します。コンソール、CLI、RDS API を使用して DB クラスターとディレクトリを関連付けることができます。これには以下の 2 つの方法があります。

- コンソール、create-db-cluster CLI コマンド、または CreateDBCluster RDS API オペレーションを使用して、新しい PostgreSQL DB クラスターを作成します。手順については、Aurora PostgreSQL DB クラスターの作成と接続 を参照してください。
- コンソール、modify-db-cluster CLI コマンド、または ModifyDBCluster RDS API オペレーションを使用して、既存の PostgreSQL DB クラスターを変更します。手順については、Amazon Aurora DB クラスターの変更 を参照してください。
- コンソール、restore-db-cluster-from-db-snapshot CLI コマンド、または RestoreDBClusterFromDBSnapshot RDS API オペレーションを使用して、DB スナップショットから PostgreSQL DB クラスターを復元します。手順については、DB クラスターのスナップショットからの復元 を参照してください。
- コンソール、restore-db-instance-to-point-in-time CLI コマンド、RestoreDBClusterToPointInTime RDS API オペレーションを使用して、PostgreSQL DB クラスターをポイントインタイムに復元します。手順については、DB クラスターを指定の時点の状態に復元する を参照してください。

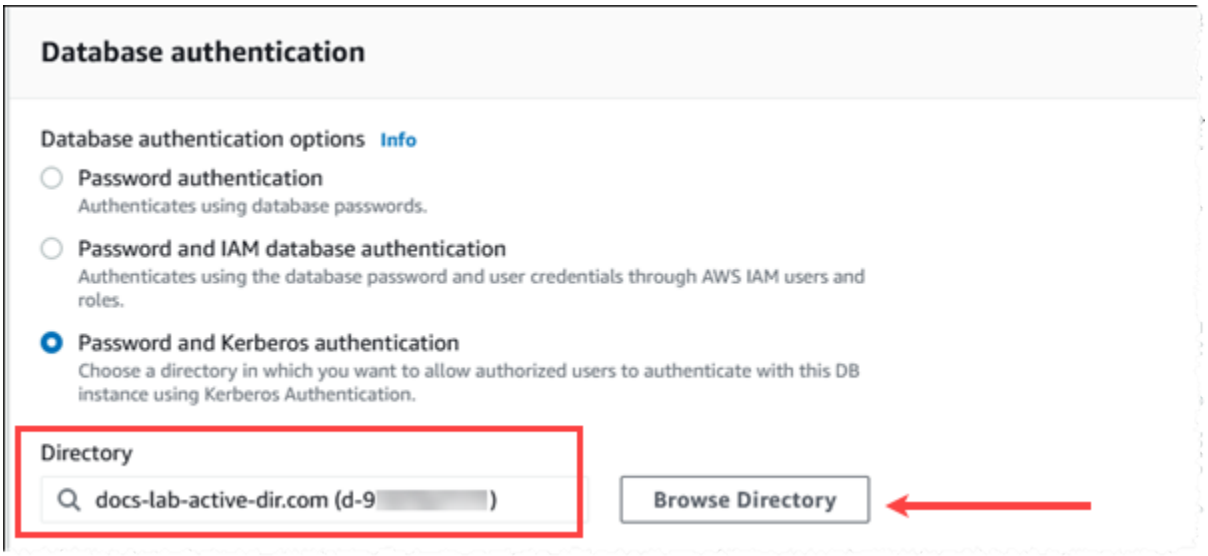
Kerberos 認証は、VPC 内の PostgreSQL DB クラスターでのみサポートされています。DB クラスターは、ディレクトリと同じ VPC または異なる VPC 内にあります。DB クラスターがディレクトリと通信する場合、そのクラスターは、ディレクトリの VPC 内での送受信を許可するセキュリティグループを使用する必要があります。

> **Note**
>
> RDS から PostgreSQL への移行中、Aurora PostgreSQL DB クラスターでは Kerberos 認証の有効化は現在サポートされていません。Kerberos 認証は、スタンドアロンの Aurora PostgreSQL DB クラスターでのみ有効にできます。

コンソール

DB クラスターを作成、変更または復元するためにコンソールを使用する場合は、[データベースの認証] セクションの [Kerberos 認証] を選択します。次に、[ディレクトリのブラウジング] を選択しま

す。Directory Service を使用するには、ディレクトリを選択するか、[新しいディレクトリの作成] を選択します。



AWS CLI

AWS CLI を使用する場合は、DB クラスターが、作成したディレクトリを使用できるように、以下のパラメータが必要です。

- `--domain` パラメータには、ディレクトリの作成時に生成されたドメイン識別子 ("d-*" 識別子) を使用します。
- `--domain-iam-role-name` パラメータには、マネージド IAM ポリシー `AmazonRDSDirectoryServiceAccess` を使用する作成済みのロールを使用します。

例えば、以下の CLI コマンドはディレクトリを使用するように DB クラスターを変更します。

```
aws rds  modify-db-cluster --db-cluster-identifier mydbinstance --domain d-Directory-ID
 --domain-iam-role-name role-name
```

> **Important**
>
> DB クラスターを変更して Kerberos 認証を有効にした場合、変更後、その DB クラスターを再起動します。

ステップ 7: Kerberos プリンシパル用の PostgreSQL ユーザーを作成する

この時点で、Aurora PostgreSQL DB クラスター が AWS Managed Microsoft AD ドメインに参加しました。ステップ 4: ユーザーを作成して設定する のディレクトリに作成したユーザーを PostgreSQL データベースユーザーとして設定し、データベースにログインする権限を付与する必要があります。そのためには、rds_superuser 権限を持つデータベースユーザーとしてサインインします。例えば、Aurora PostgreSQL DB クラスター、の作成時にデフォルト値を受け入れた場合は、次のステップに示すように postgres を使用します。

Kerberos プリンシパル用の PostgreSQL データベースユーザーを作成するには

1. psql を使用して、 Aurora PostgreSQL DB クラスターの DB インスタンスのエンドポイントに psql を使用して接続します。次の例では、postgres ロールにデフォルトの rds_superuser アカウントを使用しています。

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password
```

2. データベースにアクセスする Kerberos プリンシパル (Active Directory ユーザー名) ごとにデータベースユーザー名を作成します。Active Directory インスタンスで定義されている正規のユーザー名 (ID) を使用します。つまり、そのユーザー名には、小文字 alias (Active Directory 内のユーザー名) と Active Directory ドメインの大文字の名前を使用します。Active Directory ユーザー名は外部認証されたユーザーなので、次に示すように名前を引用符で囲んでください。

```
postgres=> CREATE USER "username@CORP.EXAMPLE.COM" WITH LOGIN;
CREATE ROLE
```

3. データベースユーザーに rds_ad ロールを付与します。

```
postgres=> GRANT rds_ad TO "username@CORP.EXAMPLE.COM";
GRANT ROLE
```

Active Directory ユーザー ID のすべての PostgreSQL ユーザーの作成が完了すると、ユーザーは Kerberos 認証情報を使用して Aurora PostgreSQL DB クラスター にアクセスできます。

Kerberos を使用して認証するデータベースユーザーは、Active Directory ドメインのメンバーであるクライアントマシンから認証を行う必要があります。

rds_ad ロールを付与されたデータベースユーザーもその rds_iam ロールを持つことはできません。これは、ネストされたメンバーシップにも適用されます。詳細については、「 の IAM データベース認証」を参照してください。

Aurora PostgreSQL DB クラスターの大文字と小文字を区別しないユーザー名を設定する

Aurora PostgreSQL バージョン 14.5、13.8、12.12、11.17 は krb_caseins_users PostgreSQL パラメータをサポートしています。このパラメータは、大文字と小文字を区別しない Active Directory ユーザー名をサポートします。デフォルトでは、このパラメータは false に設定されているため、Aurora PostgreSQL はユーザー名の大文字と小文字を区別して解釈します。これは、Aurora PostgreSQL のすべての古いバージョンでのデフォルトの動作です。ただし、カスタム DB クラスターパラメータグループでこのパラメータを true に設定し、Aurora PostgreSQL DB クラスターがユーザー名の大文字と小文字を区別しなくても解釈できるようにします。データベースユーザーが Active Directory を使用して認証するときに、ユーザー名の大文字と小文字を間違えて入力することがあるため、この方法を検討してください。

krb_caseins_users パラメータを変更するには、Aurora PostgreSQL DB クラスターがカスタム DB クラスターパラメータグループを使用している必要があります。カスタム DB クラスターパラメータグループの作成の詳細については、「「パラメータグループを使用する」」を参照してください。

AWS CLI または AWS Management Console を使用して設定を変更できます。詳細については、「DB クラスターパラメータグループのパラメータの変更」を参照してください。

ステップ 8: PostgreSQL クライアントを設定する

PostgreSQL クライアントを設定するには、次のステップを実行します。

- ドメインを指す krb5.conf ファイル (または同等) を作成します。
- クライアントホストと AWS Directory Service 間でトラフィックが流れることを確認します。次の目的で Netcat などのネットワークユーティリティを使用します。
  - ポート 53 の DNS 経由のトラフィックを確認します。
  - ポート 53 および Kerberos の TCP/UDP 上のトラフィックを確認します。これには、AWS Directory Service の場合ポート 88 および 464 が含まれます。
- データベースポートを介してクライアントホストと DB インスタンス間でトラフィックが流れることを確認します。例えば、psql を使用してデータベースに接続し、アクセスします。

以下は、AWS Managed Microsoft AD 向けの krb5.conf の内容のサンプルです 。

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
```

以下は、オンプレミスの Microsoft Active Directory 向けの krb5.conf の内容のサンプルです。

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
 ONPREM.COM = {
  kdc = onprem.com
  admin_server = onprem.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
 .onprem.com = ONPREM.COM
 onprem.com = ONPREM.COM
 .rds.amazonaws.com = EXAMPLE.COM
 .amazonaws.com.cn = EXAMPLE.COM
 .amazon.com = EXAMPLE.COM
```

ドメイン内の DB クラスターの管理

コンソール、CLI、RDS API を使用して、DB クラスターと Microsoft Active Directory との関係を管理できます。例えば、Kerberos 認証を有効化するために、Microsoft Active Directory を関連付けることができます。また、Microsoft Active Directory の関連付けを解除して、Kerberos 認証を無効化することもできます。さらに、1 つの Microsoft Active Directory によって外部に認証される DB クラスターをもう 1 つの Microsoft Active Directory に移動することもできます。

例えば、CLI を使用して次を実行できます。

- メンバーシップが失敗した Kerberos 認証を再度有効化するには、 modify-db-cluster CLI コマンドを使用します。--domain オプションに現在のメンバーシップのディレクトリ ID を指定します。
- DB インスタンスの Kerberos 認証を無効にするには、 modify-db-cluster CLI コマンドを使用します。none オプションで、--domain を指定します。
- DB インスタンスをあるドメインから別のドメインに移動するには、 modify-db-cluster CLI コマンドを使用します。--domain オプションの新しいドメインのドメイン識別子を指定します。

ドメインのメンバーシップを理解する

DB クラスターを作成または変更すると、DBインスタンスがドメインのメンバーになります。DB インスタンスのドメインメンバーシップのステータスは、コンソールで表示したり、describe-db-instances CLI コマンドを実行して表示したりすることができます。DB インスタンスのステータスは、以下のいずれかです。

- kerberos-enabled - DB インスタンスは Kerberos 認証を有効化しました。
- enabling-kerberos - AWS は、この DB インスタンスで Kerberos 認証を有効化中です。
- pending-enable-kerberos - この DB インスタンスでは、Kerberos 認証の有効化が保留中になっています。
- pending-maintenance-enable-kerberos - AWS は、次にスケジュールされたメンテナンスウィンドウで、DB インスタンスでの Kerberos 認証の有効化を試みます。
- pending-disable-kerberos - この DB インスタンスでは、Kerberos 認証の無効化が保留中になっています。
- pending-maintenance-disable-kerberos - AWS は、次にスケジュールされたメンテナンスウィンドウで、DB インスタンスでの Kerberos 認証の無効化を試みます。
- enable-kerberos-failed - 設定の問題により、AWS が DB インスタンスで Kerberos 認証を有効化できませんでした。DB インスタンスを変更するコマンドを再発行する前に、設定の問題を修正します。
- disabling-kerberos - AWS は、この DB インスタンスで Kerberos 認証を無効化中です。

ネットワーク接続の問題や正しくない IAM ロールのために、Kerberos 認証を有効化するリクエストは失敗する可能性があります。場合によっては、DB クラスターを作成または変更するときに、Kerberos 認証を有効にしようとすると失敗する可能性があります。その場合、正しい IAM ロールを使用していることを確認してから、DB クラスターを変更し、ドメインに接続します。

PostgreSQL を Kerberos 認証と接続する

pgAdmin インターフェイスまたは psql などのコマンドラインインターフェイスを使用して、Kerberos 認証で PostgreSQL に接続できます。接続の詳細については、「Amazon Aurora PostgreSQL DB クラスターへの接続」を参照してください。エンドポイント、ポート番号、および接続に必要なその他の詳細情報を取得する方法について詳細は、「Aurora クラスターのエンドポイントの表示」を参照してください。

pgAdmin

pgAdmin を使用して、PostgreSQL を Kerberos 認証に接続するには、以下のステップを実行します。

1. クライアントコンピュータの pgAdmin アプリケーションを起動します。
2. [Dashboard] (ダッシュボード) タブで、[Add New Server] (新しいサーバーの追加) を選択します。
3. [Create - Server (作成 - サーバー)] ダイアログボックスで、[General (全般)] タブに名前を入力し、pgAdmin のサーバーを特定します。
4. [Connection] (接続) タブで、Aurora PostgreSQL データベースから次の情報を入力します。
   - [Host] (ホスト) では、Aurora PostgreSQL DB クラスターのライターインスタンスのエンドポイントを入力します。エンドポイントは次のようになります。

     ```
     AUR-cluster-instance.111122223333.aws-region.rds.amazonaws.com
     ```

     Windows クライアントからオンプレミスの Microsoft Active Directory に接続するには、ホストエンドポイントで `rds.amazonaws.com` の代わりに AWS Managed Active Directory のドメイン名を使用します。例えば、AWS Managed Active Directory のドメイン名が `corp.example.com` であるとします。この場合、[Host] (ホスト) では、エンドポイントは次のように指定されます。

     ```
     AUR-cluster-instance.111122223333.aws-region.corp.example.com
     ```

   - [Port (ポート)] に、割り当てられたポートを入力します。
   - [Maintenance database (メンテナンスデータベース)] に、クライアントが接続する初期データベースの名前を入力します。
   - [ユーザーネーム] に、「ステップ 7: Kerberos プリンシパル用の PostgreSQL ユーザーを作成する」で Kerberos 認証用に入力したユーザーネームを入力します。
5. [保存] を選択します。

Psql

psql を使用して、PostgreSQL を Kerberos 認証に接続するには、以下のステップを実行します。

1. コマンドプロンプトで、次のコマンドを実行します。

```
kinit username
```

   username をユーザー名で置き換えます。プロンプトで Microsoft Active Directory に保存されているユーザーのパスワードを入力します。

2. PostgreSQL DB クラスターがパブリックにアクセス可能な VPC を使用している場合は、EC2 クライアントの /etc/hosts ファイルに、DB クラスターエンドポイントの IP アドレスを記述します。例えば、次のコマンドは IP アドレスを取得し、それを /etc/hosts ファイルに入れます。

```
% dig +short PostgreSQL-endpoint.AWS-Region.rds.amazonaws.com
;; Truncated, retrying in TCP mode.
ec2-34-210-197-118.AWS-Region.compute.amazonaws.com.
34.210.197.118

% echo " 34.210.197.118  PostgreSQL-endpoint.AWS-Region.rds.amazonaws.com" >> /etc/
hosts
```

   Windows クライアントからオンプレミスの Microsoft Active Directory を使用している場合は、特別なエンドポイントを使用して接続する必要があります。ホストエンドポイントで Amazon ドメイン rds.amazonaws.com を使用する代わりに、AWS Managed Active Directory のドメイン名を使用します。

   例えば、AWS Managed Active Directory のドメイン名が corp.example.com であるとします。次に、エンドポイントの形式 *PostgreSQL-endpoint*.*AWS-Region*.*corp.example.com* を使用して、これを /etc/hosts ファイルに配置します。

```
% echo " 34.210.197.118  PostgreSQL-endpoint.AWS-Region.corp.example.com" >> /etc/
hosts
```

3. 次の psql コマンドを使用して、Active Directory と統合されている PostgreSQL DB クラスターにログインします。クラスターまたはインスタンスエンドポイントを使用します。

```
psql -U username@CORP.EXAMPLE.COM -p 5432 -h PostgreSQL-endpoint.AWS-
Region.rds.amazonaws.com postgres
```

オンプレミスの Active Directory を使用して、Windows クライアントから PostgreSQL DB クラスターにログインするには、前のステップ (`corp.example.com`) のドメイン名を指定して次の psql コマンドを使用します。

```
psql -U username@CORP.EXAMPLE.COM -p 5432 -h PostgreSQL-endpoint.AWS-
Region.corp.example.com postgres
```

Aurora PostgreSQL アクセスコントロールへの AD セキュリティグループの使用

Aurora PostgreSQL 14.10 および 15.5 バージョンから、Aurora PostgreSQL アクセスコントロールは AWS Directory Service for Microsoft Active Directory (AD) セキュリティグループを使用して管理できます。Aurora PostgreSQL の以前のバージョンでは、個人ユーザーに対してのみ AD による Kerberos ベースの認証がサポートされていました。各 AD ユーザーは、アクセスできるように DB クラスターに明示的にプロビジョニングされる必要がありました。

ビジネスニーズに基づいて各 AD ユーザーを DB クラスターに明示的にプロビジョニングする代わりに、以下で説明するように AD セキュリティグループを活用できます。

- AD ユーザーは、Active Directory のさまざまな AD セキュリティグループのメンバーです。これらは DB クラスター管理者によって指示されるのではなく、ビジネス要件に基づいており、AD 管理者によって処理されます。
- DB クラスター管理者は、ビジネス要件に基づいて DB インスタンスに DB ロールを作成します。これらの DB ロールには、異なるアクセス許可または権限がある場合があります。
- DB クラスター管理者は、DB クラスターごとに AD セキュリティグループから DB ロールへのマッピングを設定します。
- DB ユーザーは、AD 認証情報を使用して DB クラスターにアクセスできます。アクセスは AD セキュリティグループのメンバーシップに基づいています。AD ユーザーは、AD グループのメンバーシップに基づいて自動的にアクセスを取得または喪失します。

前提条件

AD セキュリティグループの 拡張機能を設定する前に、次のものがあることを確認してください。

- PostgreSQL DB クラスターの Kerberos 認証をセットアップします。詳細については、「PostgreSQL DB クラスターの Kerberos 認証のセットアップ」を参照してください。

  > **Note**
  >
  > AD セキュリティグループの場合は、この設定手順の「ステップ 7: Kerberos プリンシパルの PostgreSQL ユーザーを作成する」をスキップします。

- ドメイン内の DB クラスターの管理。詳細については、「ドメインでの DB クラスターの管理」を参照してください。

pg_ad_mapping 拡張機能のセットアップ

Aurora PostgreSQL では、Aurora PostgreSQL クラスターの AD セキュリティグループと DB ロール間のマッピングを管理するための pg_ad_mapping 拡張機能が提供されるようになりました。pg_ad_mapping で提供される関数の詳細については、pg_ad_mapping 拡張機能の関数の使用 を参照してください。

Aurora PostgreSQL DB クラスターに pg_ad_mapping 拡張機能を設定するには、Aurora PostgreSQL DB クラスター用のカスタム DB クラスターのパラメータグループの共有ライブラリに pg_ad_mapping を追加します。カスタム DB クラスターパラメータグループの作成の詳細については、「「パラメータグループを使用する」」を参照してください。次に、pg_ad_mapping 拡張機能をインストールします。このセクションの手順で、方法を示します。AWS Management Console または AWS CLI を使用できます。

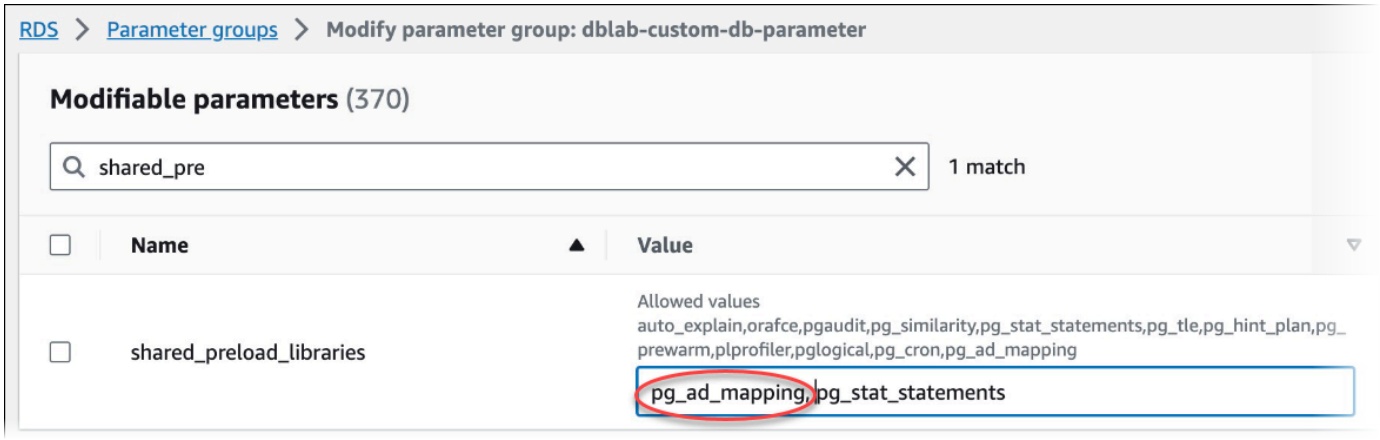
これらすべてのタスクを実行するには、rds_superuser ロールとして権限が必要です。

以下のステップでは、Aurora PostgreSQL DB クラスター がカスタム DB クラスターパラメータグループに関連付けられていることを前提としています。

コンソール

pg_ad_mapping 拡張機能をセットアップするには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、Aurora PostgreSQL DB クラスターのライターインスタンスを選択します。
3. Aurora PostgreSQL DB クラスターライターインスタンスの [設定] タブを開きます。インスタンスの詳細の中から、パラメータグループのリンクを見つけてください。
4. リンクを選択して、Aurora PostgreSQL DB クラスターに関連するカスタムパラメータを開きます。
5. パラメータ検索フィールドに、shared_pre を入力して shared_preload_libraries パラメータを検索します。
6. プロパティ値にアクセスするには、[Edit parameters] (パラメータの編集) を選択します。
7. [Values] (値) フィールドのリストに pg_ad_mapping を追加します。値のリスト内の項目を区切るにはカンマを使用します。



8. Aurora PostgreSQL DB クラスターのライターインスタンスを再起動して、shared_preload_libraries パラメータの変更を有効にします。
9. インスタンスが使用可能になったら、pg_ad_mapping が初期化されていることを確認します。psql を使用して Aurora PostgreSQL DB クラスターのライターインスタンスに接続し、次のコマンドを実行します。

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_ad_mapping
(1 row)
```

10. pg_ad_mapping を初期化すると、拡張機能を作成できるようになりました。この拡張機能が提供する関数の使用を開始するには、ライブラリを初期化した後に拡張機能を作成する必要があります。

```
CREATE EXTENSION pg_ad_mapping;
```

11. psql セッションを終了します。

```
labdb=> \q
```

AWS CLI

pg_ad_mapping をセットアップするには

AWS CLI を使用して pg_ad_mapping をセットアップするには、次の手順に示すように、modify-db-parameter-group オペレーションを呼び出して、カスタムパラメータグループにこのパラメータを追加します。

1. 次の AWS CLI コマンドを使用してshared_preload_librariesパラメータに pg_ad_mapping を追加します。

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pg_ad_mapping,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. 次の AWS CLI コマンドを使用して Aurora PostgreSQL DB クラスターのライターインスタンスを再起動し、pg_ad_mapping ライブラリを初期化します。

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

3. インスタンスが使用可能になると、pg_ad_mapping が初期化されていることを確認できます。psql を使用して Aurora PostgreSQL DB クラスターのライターインスタンスに接続し、次のコマンドを実行します。

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_ad_mapping
(1 row)
```

   pg_ad_mapping を初期化すると、拡張機能を作成できるようになりました。

```
CREATE EXTENSION pg_ad_mapping;
```

4. AWS CLI を使用できるように psql セッションを終了します。

```
labdb=> \q
```

PowerShell での Active Directory グループ SID の取得

セキュリティ識別子 (SID) は、セキュリティプリンシパルまたはセキュリティグループを一意に識別するために使用されます。Active Directory でセキュリティグループまたはアカウントを作成するたびに、SID が割り当てられます。Active Directory から AD セキュリティグループ SID を取得するには、その Active Directory ドメインに参加している Windows クライアントマシンから Get-ADGroup コマンドレットを使用できます。ID パラメータは、対応する SID を取得するための Active Directory グループ名を指定します。

次の例では、AD グループ *adgroup1* の SID を返します。

```
C:\Users\Admin> Get-ADGroup -Identity adgroup1 | select SID

             SID
-----------------------------------------------
S-1-5-21-3168537779-1985441202-1799118680-1612
```

AD セキュリティグループによる DB ロールのマッピング

データベース内の AD セキュリティグループを PostgreSQL DB ロールとして明示的にプロビジョニングする必要があります。少なくとも 1 つのプロビジョニングされた AD セキュリティグループに属する AD ユーザーは、データベースにアクセスできます。AD グループのセキュリティベースの DB ロールに `rds_ad role` を付与しないでください。セキュリティグループの Kerberos 認証は、*user1@example.com* などのドメイン名サフィックスを使用してトリガーされます。この DB ロールは、パスワードまたは IAM 認証を使用してデータベースにアクセスすることはできません。

> **Note**
>
> `rds_ad` ロールが付与されたデータベースに対応する DB ロールを持つ AD ユーザーは、AD セキュリティグループの一部としてログインできません。個人ユーザーとして DB ロールを通じてアクセスできるようになります。

例えば、accounts-group は、Aurora PostgreSQL でこのセキュリティグループを accounts-role としてプロビジョニングする AD のセキュリティグループです。

AD セキュリティグループ	PosgreSQL DB ロール
accounts-group	accounts-role

DB ロールを AD セキュリティグループにマッピングするときは、DB ロールに LOGIN 属性が設定され、必要なログインデータベースへの CONNECT 権限があることを確認する必要があります。

```
postgres => alter role accounts-role login;

ALTER ROLE
postgres => grant connect on database accounts-db to accounts-role;
```

管理者は AD セキュリティグループと PostgreSQL DB ロール間のマッピングの作成に進むことができるようになりました。

```
admin=>select pgadmap_set_mapping('accounts-group', 'accounts-role', <SID>, <Weight>);
```

AD セキュリティグループの SID の取得については、「PowerShell での Active Directory グループ SID の取得」を参照してください。

AD ユーザーが複数のグループに属している場合があります。その場合、AD ユーザーは DB ロールの権限を継承します。この権限は、最も高い重みでプロビジョニングされます。2 つのロールの重みが同じ場合、AD ユーザーは最近追加されたマッピングに対応する DB ロールの権限を継承します。個々の DB ロールの相対的なアクセス許可/権限を反映する重みを指定することをお勧めします。DB ロールのアクセス許可または権限が高いほど、マッピングエントリに関連付ける必要がある重みが高くなります。これにより、同じ重みを持つ 2 つのマッピングのあいまいさを回避できます。

次の表は、AD セキュリティグループから Aurora PostgreSQL DB ロールへのマッピングの例を示しています。

AD セキュリティグループ	PosgreSQL DB ロール	(重量)
accounts-group	accounts-role	7

AD セキュリティグループ	PosgreSQL DB ロール	(重量)
sales-group	sales-role	10
dev-group	dev-role	7

次の例では、user1 はより高い重みをもつため sales-role の権限を継承します。一方、user2 はこのコードのマッピングが accounts-role と同じ重みを共有する accounts-role の後に作成されたため、dev-role の権限を継承します。

ユーザーネーム	セキュリティグループのメンバーシップ
user1	accounts-group sales-group
user2	accounts-group dev-group

マッピングを確立、一覧表示、およびクリアするための psql コマンドを以下に示します。現在、単一のマッピングエントリを変更することはできません。既存のエントリを削除し、マッピングを再作成する必要があります。

```
admin=>select pgadmap_set_mapping('accounts-group', 'accounts-role', 'S-1-5-67-890',
 7);
admin=>select pgadmap_set_mapping('sales-group', 'sales-role', 'S-1-2-34-560', 10);
admin=>select pgadmap_set_mapping('dev-group', 'dev-role', 'S-1-8-43-612', 7);

admin=>select * from pgadmap_read_mapping();

ad_sid       | pg_role        | weight | ad_grp
-------------+----------------+--------+---------------
S-1-5-67-890 | accounts-role | 7      | accounts-group
S-1-2-34-560 | sales-role    | 10     | sales-group
S-1-8-43-612 | dev-role      | 7      | dev-group
(3 rows)
```

AD ユーザー ID のログ記録/監査

現在のユーザーまたはセッションユーザーによって継承されたデータベースロールを確認するには、次のコマンドを使用します。

```
postgres=>select session_user, current_user;

session_user | current_user
-------------+--------------
dev-role     | dev-role

(1 row)
```

AD セキュリティプリンシパル ID を確認するには、次のコマンドを使用します。

```
postgres=>select principal from pg_stat_gssapi where pid = pg_backend_pid();

 principal
-------------------------
 user1@example.com

(1 row)
```

現在、AD ユーザー ID は監査ログに表示されません。log_connections パラメータを有効にすると、DB セッションの確立をログに記録できます。詳細については、「log_connections」を参照してください。この出力には、次に示すように AD ユーザー ID が含まれます。この出力に関連付けられたバックエンド PID は、属性アクションを実際の AD ユーザーに戻すのに役立ちます。

```
pgrole1@postgres:[615]:LOG: connection authorized: user=pgrole1
 database=postgres application_name=psql GSS (authenticated=yes, encrypted=yes,
 principal=Admin@EXAMPLE.COM)
```

制限事項

- Azure Active Directory として知られる Microsoft Entra ID はサポートされていません。

pg_ad_mapping 拡張機能の関数の使用

提供される pg_ad_mapping 拡張機能は、次の関数をサポートしています。

pgadmap_set_mapping

この関数は、AD セキュリティグループとデータベースロール間のマッピングを関連する重みで確立します。

構文

```
pgadmap_set_mapping(
ad_group,
db_role,
ad_group_sid,
weight)
```

引数

パラメータ	説明
ad_group	AD グループの名前。値を NULL または空の文字列にすることはできません。
db_role	指定された AD グループにマッピングされるデータベースロール。値を NULL または空の文字列にすることはできません。
ad_group_sid	AD グループを一意に識別するために使用されるセキュリティ識別子。値は「S-1-」で始まり、NULL または空の文字列にすることはできません。詳細については、「PowerShell での Active Directory グループ SID の取得」を参照してください。
weight	データベースロールに関連付けられた重み。重みが最も高いロールは、ユーザーが複数のグループのメンバーである場合に優先されます。重みのデフォルト値は 1 です。

戻り型

None

使用に関する注意事項

この関数は、AD セキュリティグループからデータベースロールに新しいマッピングを追加します。DB クラスターのプライマリ DB インスタンスで実行できるのは、rds_superuser 権限を持つユーザーのみです。

例

```
postgres=> select pgadmap_set_mapping('accounts-group','accounts-
role','S-1-2-33-12345-67890-12345-678',10);

pgadmap_set_mapping

(1 row)
```

pgadmap_read_mapping

この関数は、pgadmap_set_mapping 関数を使用して設定された AD セキュリティグループと DB ロール間のマッピングを一覧表示します。

構文

```
pgadmap_read_mapping()
```

引数

None

戻り型

パラメータ	説明
ad_group_sid	AD グループを一意に識別するために使用されるセキュリティ識別子。値は「S-1-」で始まり、NULL または空の文字列にすることはできません。詳細については、「PowerShell での Active Directory グループ SID の取得.accounts-role@example.com」を参照してください。
db_role	指定された AD グループにマッピングされるデータベースロール。値を NULL または空の文字列にすることはできません。

パラメータ	説明
weight	データベースロールに関連付けられた重み。重みが最も高いロールは、ユーザーが複数のグループのメンバーである場合に優先されます。重みのデフォルト値は 1 です。
ad_group	AD グループの名前。値を NULL または空の文字列にすることはできません。

使用に関する注意事項

この関数を呼び出して、AD セキュリティグループと DB ロール間で使用可能なすべてのマッピングを一覧表示します。

例

```
postgres=> select * from pgadmap_read_mapping();

ad_sid                               | pg_role       | weight | ad_grp
-------------------------------------+---------------+--------+------------------
S-1-2-33-12345-67890-12345-678       | accounts-role | 10     | accounts-group
(1 row)

(1 row)
```

pgadmap_reset_mapping

この関数は、pgadmap_set_mapping 関数を使用して設定された 1 つまたはすべてのマッピングをリセットします。

構文

```
pgadmap_reset_mapping(
ad_group_sid,
db_role,
weight)
```

引数

パラメータ	説明
ad_group_sid	AD グループを一意に識別するために使用されるセキュリティ識別子。
db_role	指定された AD グループにマッピングされるデータベースロール。
weight	データベースロールに関連付けられた重み。

引数を指定しない場合、AD グループと DB ロール間のマッピングはすべてリセットされます。すべての引数を指定するか、何も指定しないかのいずれかにします。

戻り型

None

使用に関する注意事項

この関数を呼び出して、特定の AD グループから DB ロールへのマッピングを削除するか、すべてのマッピングをリセットします。この関数は、DB クラスターのプライマリ DB インスタンスで実行できるのは、rds_superuser 権限を持つユーザーのみです。

例

```
postgres=> select * from pgadmap_read_mapping();

   ad_sid                         | pg_role      | weight      | ad_grp
----------------------------------+--------------+-------------+-------------------
 S-1-2-33-12345-67890-12345-678 | accounts-role| 10          | accounts-group
 S-1-2-33-12345-67890-12345-666 | sales-role   | 10          | sales-group

(2 rows)
postgres=> select pgadmap_reset_mapping('S-1-2-33-12345-67890-12345-678', 'accounts-
role', 10);

pgadmap_reset_mapping
(1 row)
```

```
postgres=> select * from pgadmap_read_mapping();

   ad_sid                         | pg_role      | weight      | ad_grp
----------------------------------+--------------+-------------+--------------
 S-1-2-33-12345-67890-12345-666 | sales-role   | 10          | sales-group

(1 row)
postgres=> select pgadmap_reset_mapping();

pgadmap_reset_mapping
(1 row)

postgres=> select * from pgadmap_read_mapping();

   ad_sid                         | pg_role      | weight      | ad_grp
----------------------------------+--------------+-------------+--------------
 (0 rows)
```

PostgreSQL と互換性がある Amazon Aurora にデータを移行する

既存のデータベースから Amazon Aurora PostgreSQL 互換エディション DB クラスターにデータを移行するための複数のオプションがあります。また、移行オプションは、移行元のデータベースおよび移行するデータのサイズによっても異なります。オプションは次のとおりです。

スナップショットを使用して RDS for PostgreSQL DB インスタンスを移行する

データは、RDS for PostgreSQL DB スナップショットから Aurora PostgreSQL DB クラスターに直接移行できます。

Aurora リードレプリカを使用して RDS for PostgreSQL DB インスタンスを移行する

また、RDS for PostgreSQL DB インスタンスの Aurora PostgreSQL リードレプリカを作成して、RDS for PostgreSQL DB インスタンスから移行することも可能です。RDS for PostgreSQL DB インスタンスと Aurora PostgreSQL リードレプリカの間の複製ラグがない状態であれば、レプリケーションを停止することができます。この時点で、Aurora リードレプリカを読み書き用のスタンドアロン Aurora PostgreSQL DB クラスターにすることができます。

Amazon S3 から Aurora PostgreSQL にデータをインポートする

データを Amazon S3 から Aurora PostgreSQL DB クラスターに属するテーブルにインポートすることで、データを移行できます。

PostgreSQL 互換ではないデータベースからの移行

PostgreSQL との互換性がないデータベースからのデータを移行するには、AWS Database Migration Service (AWS DMS) を使用できます。AWS DMS の詳細については、AWS Database Migration Serviceユーザーガイドの「AWS Database Migration Service とは?」を参照してください。

> **Note**
>
> RDS から PostgreSQL への移行中、Aurora PostgreSQL DB クラスターでは Kerberos 認証の有効化は現在サポートされていません。Kerberos 認証は、スタンドアロンの Aurora PostgreSQL DB クラスターでのみ有効にできます。

Aurora を使用できる AWS リージョン の一覧は、AWS 全般のリファレンス の「Amazon Aurora」を参照してください。

> **Important**
>
> 近い将来 RDS for PostgreSQL DB インスタンスを Aurora PostgreSQL DB クラスターに移行する予定がある場合は、移行計画の早い段階で DB インスタンスのマイナーバージョンの自動アップグレードをオフにすることを強くお勧めします。RDS for PostgreSQL バージョンが Aurora PostgreSQL でサポートされていない場合、Aurora PostgreSQL への移行が遅れる可能性があります。
> Aurora PostgreSQL バージョンについては、Amazon Aurora PostgreSQL のエンジンのバージョンを参照してください。

RDS for PostgreSQL DB インスタンスの スナップショットを Aurora PostgreSQL DB クラスターに移行する

RDS for PostgreSQL DB インスタンスの DB スナップショットを移行して、Aurora PostgreSQL DB クラスターを作成することができます。作成された Aurora PostgreSQL DB クラスターには、元の RDS for PostgreSQL DB インスタンスのデータが格納されます。DB スナップショットの作成については、「DB スナップショットの作成」を参照してください。

場合によっては、DB スナップショットが、データを配置したい AWS リージョン にない可能性があります。その場合は、Amazon RDS コンソールを使用して、DB スナップショットを対象の AWS

リージョン にコピーします。DB スナップショットのコピーについては、「DB スナップショットのコピー」を参照してください。

対象の AWS リージョン で利用可能な Aurora PostgreSQL バージョンと互換性のある、RDS for PostgreSQL スナップショットを移行できます。例えば、RDS for PostgreSQL 11.1 DB インスタンスからのスナップショットは、米国西部 (北カリフォルニア) リージョンで、Aurora PostgreSQL のバージョン 11.4、11.7、11.8、または 11.9 に移行できます。RDS PostgreSQL 10.11 スナップショットは、Aurora PostgreSQL 10.11、10.12、10.13、および 10.14 に移行できます。つまり、RDS for PostgreSQL スナップショットは、Aurora PostgreSQL と同じかそれ以下のマイナーバージョンを使用する必要があります。

また、新しい Aurora PostgreSQL DB クラスターが、AWS KMS key を使用して、保管中に暗号化されるよう設定することもできます。このオプションは、暗号化されていない DB スナップショットに対してのみ使用できます。

RDS for PostgreSQL DB スナップショットを Aurora PostgreSQL DB クラスターに移行するには、AWS Management Console、AWS CLI、または RDS API を使用できます。AWS Management Console を使用すると、DB クラスターとプライマリインスタンスの両方を作成するために必要なアクションがコンソールによって実行されます。

コンソール

RDS コンソールを使用して PostgreSQL DB スナップショットを移行するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. [Snapshots] を選択します。
3. [スナップショット] ページで、Aurora PostgreSQL DB クラスターに移行する RDS for PostgreSQL スナップショットを選択します。
4. [アクション]、[スナップショットの移行] の順に選択します。
5. [データベースの移行] ページで以下の値を設定します。

   - [DB エンジンバージョン]: 移行した新しいインスタンスに使用する DB エンジンのバージョンを選択します。
   - [DB instance identifier] (DB インスタンス識別子): DB クラスター名を入力します。これは、選択した AWS リージョン 内で自分のアカウントに対して一意である必要があります。この識別子は、DB クラスター内のインスタンスのエンドポイントアドレスで使用されます。分かり

やすくするために、選択した AWS リージョン と DB エンジンなど (例: **aurora-cluster1**) を名前に含めます。

DB インスタンス識別子には次の制約があります。

- 1 ~ 63 文字の英数字またはハイフンを使用する必要があります。
- 1 字目は文字である必要があります。
- 文字列の最後にハイフンを使用したり、ハイフンを 2 つ続けて使用したりすることはできません。
- 1 つの AWS アカウント、1 つの AWS リージョン につき、すべての DB インスタンスにおいて一意である必要があります。

- [DB インスタンスクラス]: データベースに必要なストレージと容量のある DB インスタンスクラス (db.r6g.large など) を選択します。Aurora クラスターボリュームは、データベースのデータ量が増えるにつれて自動的に増加します。そのため、現在のストレージ要件を満たしている DB インスタンスクラスを選択する必要があります。詳細については、「Amazon Aurora ストレージの概要」を参照してください。
- [Virtual Private Cloud (VPC)]: 既存の VPC がある場合は、その VPC 識別子 (vpc-a464d1c1 など) を選択することで、その VPC を Aurora PostgreSQL DB クラスターで使用できます。VPC の作成方法の詳細については、「チュートリアル: DB クラスターで使用する VPC を作成する (IPv4 専用)」を参照してください。

  それ以外の場合は、[新しい VPC の作成] を選択することにより、Amazon RDS で VPC を作成できます。
- DB サブネットグループ: 既存のサブネットグループがある場合は、そのサブネットグループ識別子 (gs-subnet-group1 など) を選択して、サブネットグループを Aurora PostgreSQL DB クラスターで使用できます。
- [パブリックアクセス]: DB クラスターのインスタンスが VPC 内のリソースからのみアクセスできることを指定するには、[いいえ] を選択します。DB クラスターのインスタンスがパブリックネットワーク上のリソースからアクセスできることを指定するには、[はい] を選択します。

> **Note**
>
> 本番稼働用の DB クラスターは、お客様のアプリケーションサーバーのみがアクセスするため、パブリックサブネット内に配置する必要がない場合があります。DB クラ

スターをパブリックサブネットに配置する必要がない場合は、[パブリックアクセス可能] を [いいえ] に設定します。

- [VPC セキュリティグループ]: データベースへのアクセスを許可する VPC セキュリティグループを選択します。
- [アベイラビリティーゾーン]: Aurora PostgreSQL DB クラスターのプライマリインスタンスをホストするアベイラビリティーゾーンを選択します。Amazon RDS でアベイラビリティーゾーンが選択されるようにするには、[指定なし] を選択します。
- [データベースのポート]: Aurora PostgreSQL DB クラスターのインスタンスへの接続に使用されるデフォルトのポートを入力します。デフォルト: 5432。

> **Note**
>
> 会社のファイアウォールでデフォルトポート (PostgreSQL のデフォルトポート 5432 など) へのアクセスが許可されない場合があります。この場合は、会社のファイアウォールによって許可されるポート値を指定します。そのポート値を覚えておいてください。後で Aurora PostgreSQL DB クラスターに接続するときに使用します。

- [暗号を有効化]: 新しい Aurora PostgreSQL DB クラスターを保管時に暗号化するには、[暗号を有効化] を選択します。また、AWS KMS key 値としてKMS キーを選択します。
- [マイナーバージョン自動アップグレード]: PostgreSQL DB エンジンのマイナーバージョンアップグレードを、そのリリースと同時に Aurora PostgreSQL DB クラスターに自動的に適用するには、[マイナーバージョン自動アップグレードを有効にする] を選択します。

  [マイナーバージョン自動アップグレード] オプションは、Aurora PostgreSQL DB クラスターの PostgreSQL マイナーエンジンバージョンに対するアップグレードのみに適用されます。システム安定性を維持するために適用される定期的なパッチは適用されません。

6. [Migrate] を選択して、DB スナップショットを移行します。
7. [データベース] を選択して新しい DB クラスターを表示します。移行の進行状況を監視する新しい DB クラスターを選択します。移行が完了すると、クラスターのステータスは [Available] (使用可能) になります。[接続とセキュリティ] タブで、DB クラスターのプライマリライターインスタンスへの接続に使用するクラスターエンドポイントを見つけます。Aurora PostgreSQL DB クラスターとの接続の詳細については、「Amazon Aurora DB クラスターへの接続」を参照してください。

AWS CLI

AWS CLI を使用して RDS for PostgreSQL DB スナップショットを Aurora PostgreSQL に移行するには、2 つの別々の AWS CLI コマンドを使用します。まず、AWS CLI コマンド `restore-db-cluster-from-snapshot` で新しい Aurora PostgreSQL DB クラスターを作成します。次に、`create-db-instance` コマンドを使用して、新しいクラスターにプライマリ DB インスタンスを作成し、移行を完了します。次の手順では、スナップショットの作成に使用した DB インスタンスと同じ構成のプライマリ DB インスタンスで Aurora PostgreSQL DB クラスターを作成します。

RDS for PostgreSQL DB スナップショットを Aurora PostgreSQL DB クラスターに移行するには

1. describe-db-snapshots コマンドを使用して、移行する DB スナップショットに関する情報を取得します。コマンドで `--db-instance-identifier` パラメータまたは `--db-snapshot-identifier` を指定できます。これらのパラメータのいずれかを指定しないと、すべてのスナップショットを取得することになります。

   ```
   aws rds describe-db-snapshots --db-instance-identifier <your-db-instance-name>
   ```

2. このコマンドは、指定した DB インスタンスから作成されたスナップショットの設定の詳細をすべて返します。出力から移行するスナップショットを見つけ、その Amazon リソースネーム (ARN) を見つけます。Amazon RDS ARN の詳細については、「Amazon Relational Database Service (Amazon RDS)」を参照してください。ARN は次の出力例のようになります。

   ```
   “DBSnapshotArn": "arn:aws:rds:aws-region:111122223333:snapshot:<snapshot_name>"
   ```

   また、出力では、エンジンのバージョン、割り当てられたストレージ、DB インスタンスが暗号化されているかどうかなど、RDS for PostgreSQL DB インスタンスの設定の詳細を確認できます。

3. restore-db-cluster-from-snapshot コマンドを使用して、移行を開始します。以下のパラメータを指定します。

   - `--db-cluster-identifier` – Aurora PostgreSQL DB クラスターに割り当てる名前。この Aurora DB クラスターは、DB スナップショット移行のターゲットです。
   - `--snapshot-identifier` – 移行する DB スナップショットの Amazon リソースネーム (ARN)。
   - `--engine` – Aurora DB クラスターエンジンの `aurora-postgresql` を指定します。
   - `--kms-key-id` – このオプションパラメータを使用すると、暗号化されていない DB スナップショットから暗号化された Aurora PostgreSQL DB クラスターを作成できます。また、DB

スナップショットに使用されたキーとは異なる暗号化キーを DB クラスター用に選択することもできます。

> **Note**
>
> 暗号化された DB スナップショットから暗号化されていない Aurora PostgreSQL DB クラスターを作成することはできません。

`--kms-key-id` パラメータを以下に示すように指定しないと、restore-db-cluster-from-snapshot AWS CLI コマンドは、DB スナップショットと同じキーを使用して暗号化されているか、ソース DB スナップショットが暗号化されていない場合は暗号化されていない空の Aurora PostgreSQL DB クラスターを作成します。

Linux、macOS、Unix の場合:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier cluster-name \
    --snapshot-identifier arn:aws:rds:aws-region:111122223333:snapshot:your-snapshot-name \
    --engine aurora-postgresql
```

Windows の場合:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier new_cluster ^
    --snapshot-identifier arn:aws:rds:aws-region:111122223333:snapshot:your-snapshot-name ^
    --engine aurora-postgresql
```

4. このコマンドは、移行用に作成されている Aurora PostgreSQL DB クラスターの詳細を返します。Aurora PostgreSQL DB クラスターのステータスは、describe-db-clusters AWS CLI コマンドで確認できます。

```
aws rds describe-db-clusters --db-cluster-identifier cluster-name
```

5. DB クラスターが「使用可能」になったら、create-db-instance コマンドを使用して、Amazon RDS DB スナップショットに基づいて、Aurora PostgreSQL DB クラスターに DB インスタンスを入力します。以下のパラメータを指定します。

- `--db-cluster-identifier` – 前のステップで作成した新しい Aurora PostgreSQL DB クラスターの名前。
- `--db-instance-identifier` – DB インスタンスに付ける名前。このインスタンスは、Aurora PostgreSQL DB クラスターのプライマリノードになります。
- `----db-instance-class` – 使用する DB インスタンスクラスを指定します。移行先の Aurora PostgreSQL バージョンでサポートされている DB インスタンスクラスの中から選択します。詳細については、DB インスタンスクラスタイプおよびDB インスタンスクラスでサポートされている DB エンジンを参照してください。
- `--engine` – DB インスタンスに `aurora-postgresql` を指定します。

`create-db-instance` AWS CLI コマンドで適切なオプションを渡すことで、ソース DB スナップショットとは異なる設定で DB インスタンスを作成することもできます。詳細については、「create-db-instance」コマンドを参照してください。

Linux、macOS、Unix の場合:

```
aws rds create-db-instance \
    --db-cluster-identifier cluster-name \
    --db-instance-identifier --db-instance-class db.instance.class \
    --engine aurora-postgresql
```

Windows の場合:

```
aws rds create-db-instance ^
    --db-cluster-identifier cluster-name ^
    --db-instance-identifier --db-instance-class db.instance.class ^
    --engine aurora-postgresql
```

移行プロセスが完了すると、Aurora PostgreSQL クラスターでは、プライマリ DB インスタンスが設定されます。

Aurora リードレプリカを使用して RDS for PostgreSQL DB インスタンスから Aurora PostgreSQL DB クラスターにデータを移行する

移行プロセス用の Aurora リードレプリカにより、RDS for PostgreSQL DB インスタンスを新しい Aurora PostgreSQL DB クラスターのベースとして使用できます。Aurora リードレプリカオプションは、同じ AWS リージョン とアカウント内での移行にのみ使用でき、リージョンが RDS for PostgreSQL DB インスタンスに対して互換性のある Aurora PostgreSQL のバージョンを提供している場合にのみ使用できます。この場合、互換性があるとは、Aurora PostgreSQL のバージョンが RDS for PostgreSQL バージョンと同じであるか、同じメジャーバージョンファミリー内の上位マイナーバージョンであることを意味します。

例えば、この手法を使用して RDS for PostgreSQL 11.14 DB インスタンスを移行するには、リージョンで Aurora PostgreSQL バージョン 11.14 または PostgreSQL バージョン 11 ファミリーの上位マイナーバージョンが提供されている必要があります。

トピック

Aurora リードレプリカを使用したデータ移行の概要

RDS for PostgreSQL DB インスタンスから Aurora PostgreSQL DB クラスターに移行するには、複数のステップが必要です。まず、ソース RDS for PostgreSQL DB インスタンスの Aurora リードレプリカを作成します。これにより、RDS for PostgreSQL DB インスタンスから、レプリカクラスターと呼ばれる特殊目的 DB クラスターへのレプリケーションプロセスが開始されます。レプリカクラスターは 1 つの Aurora リードレプリカ (リーダーインスタンス) のみで構成されます。

レプリカクラスターができたら、レプリカクラスターとソース RDS for PostgreSQL DB インスタンスとの遅延時間をモニタリングします。レプリカ遅延がゼロ (0) の場合、レプリカクラスターを昇格させることができます。レプリケーションが停止し、レプリカクラスターはスタンドアロン Aurora DB クラスターに昇格し、リーダーはクラスターのライターインスタンスに昇格します。その後、Aurora PostgreSQL DB クラスターにインスタンスを追加して、ユースケースに合わせて Aurora PostgreSQL DB クラスターのサイズを設定できます。RDS for PostgreSQL DB インスタンスは、それ以上必要ない場合は、削除してもかまいません。

> **Note**
>
> 移行が完了するまでに、1 テラバイトのデータあたり数時間かかることがあります。

RDS for PostgreSQL DB インスタンスに既に Aurora リードレプリカがある場合、またはクロスリージョンリードレプリカがある場合は、Aurora リードレプリカを作成できません。

Aurora リードレプリカを使用してデータを移行するよう準備する

Aurora リードレプリカを使用した移行プロセス中に、ソース RDS for PostgreSQL DB インスタンスに加えられた更新は、レプリカクラスターの Aurora リードレプリカに非同期的にレプリケーションされます。このプロセスでは、ソースインスタンスにログ先行書き込み (WAL) セグメントを保存する PostgreSQL のネイティブのストリーミングレプリケーション機能を使用します。この移行プロセスを開始する前に、テーブルにリストされているメトリクスの値をチェックして、インスタンスに十分なストレージ容量があることを確認してください。

メトリクス	説明
`FreeStorageSpace`	使用可能なストレージ領域。 単位: バイト
`OldestReplicationSlotLag`	最も遅延の大きいレプリカの WAL データに関する遅延サイズ。 単位: メガバイト
`RDSToAuroraPostgreSQLReplicaLag`	Aurora PostgreSQL DB クラスターがソース RDS DB インスタンスより遅れる時間 (秒単位)。
`TransactionLogsDiskUsage`	トランザクションログで使用されているディスク容量。 単位: メガバイト

RDS インスタンスのモニタリングの詳細については、Amazon RDS ユーザーガイド の「モニタリング」を参照してください。

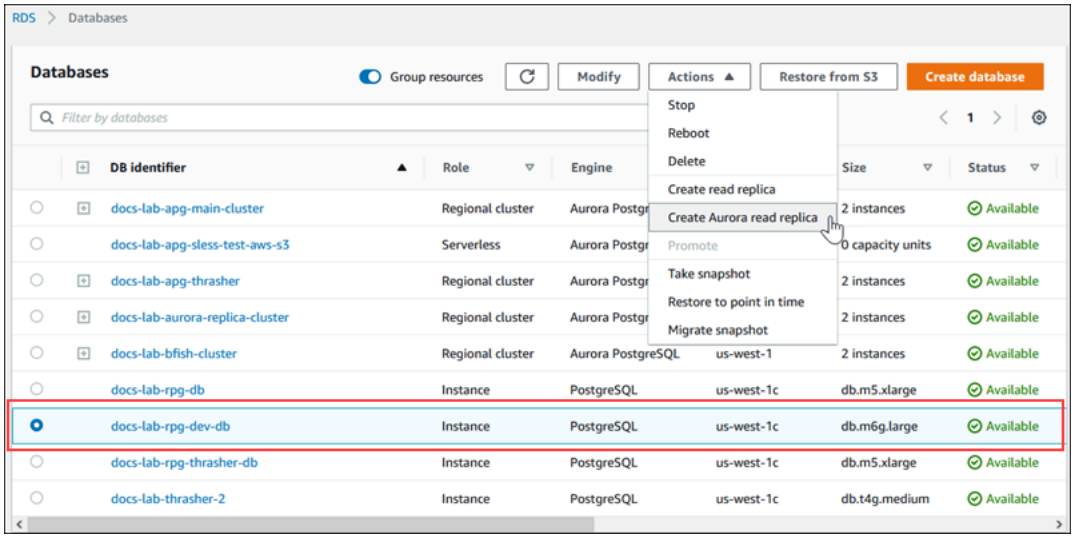
Aurora リードレプリカの作成

AWS Management Console または AWS CLI を使用して、RDS for PostgreSQL DB インスタンスの Aurora リードレプリカを作成できます。AWS Management Console を使用して Aurora リードレプリカを作成するオプションは、AWS リージョン が互換性のある Aurora PostgreSQL バージョンを提供している場合のみ使用可能です。つまり、このオプションは、RDS for PostgreSQL バージョンと同じか、同じメジャーバージョンファミリーの上位マイナーバージョンの Aurora PostgreSQL バージョンがある場合にのみ使用できます。

コンソール

ソース PostgreSQL DB インスタンスから Aurora リードレプリカを作成するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、データベースを選択します。
3. Aurora リードレプリカのソースとして使用する RDS for PostgreSQL DB インスタンスを選択します。[アクション] で [Aurora リードレプリカの作成] を選択します。この選択肢が表示されない場合は、互換性のあるバージョンの Aurora PostgreSQL がリージョンで使用できないことを意味します。



4. Aurora リードレプリカの設定の作成ページで、次の表に示すように、Aurora PostgreSQL DB クラスターのプロパティを設定します。レプリカ DB クラスターは、ソースと同じ「マスター」ユーザー名およびパスワードを使用してソース DB インスタンスのスナップショットから作成されるため、この時点では変更できません。

オプション	説明
DB インスタンスクラス	DB クラスターのプライマリインスタンスに対する処理要件やメモリ要件に適合する DB インスタンスクラスを選択します。詳細については、「Aurora DB インスタンスクラス」を参照してください。
マルチ AZ 配置	移行中は使用できません
DB インスタンス識別子	DB インスタンスに付ける名前を入力します。この識別子は、新しい DB クラスターのプライマリインスタンスのエンドポイントアドレスで使用されます。 DB インスタンス識別子には次の制約があります。 • 1 ~ 63 文字の英数字またはハイフンを使用する必要があります。 • 1 字目は文字である必要があります。 • 文字列の最後にハイフンを使用したり、ハイフンを 2 つ続けて使用したりすることはできません。 • これは、AWS リージョン ごと、AWS アカウントごとにすべての DB インスタンスに対して一意にする必要があります。
仮想プライベートクラウド (VPC)	DB クラスターをホストする VPC を選択します。[新しい VPC の作成] を選択して、Amazon RDS で VPC を作成します。詳細については、「DB クラスターの前提条件」を参照してください。
DB サブネットグループ	DB クラスターで使用する DB サブネットグループを選択します。[新しい DB サブネットグループの作成] を選択すると、Amazon RDS によって DB サブネットグループが作成されます。詳細については、「DB クラスターの前提条件」を参照してください。

オプション	説明
パブリックアクセシビリティ	DB クラスターにパブリック IP アドレスを指定するには [はい] を選択します。それ以外の場合は [いいえ] を選択します。DB クラスターのインスタンスでは、パブリック DB インスタンスとプライベート DB インスタンスの両方を混在させることができます。パブリックアクセスからインスタンスを隠す方法については、「VPC 内の DB クラスターをインターネットから隠す」を参照してください。
アベイラビリティーゾーン	特定のアベイラビリティーゾーンを指定するかどうかを指定します。利用可能ゾーンについての詳細は、リージョンとアベイラビリティーゾーン を参照してください。
VPC セキュリティグループ	DB クラスターへのネットワークアクセスの保護用に 1 つ以上の VPC セキュリティグループを選択します。[新しい VPC セキュリティグループの作成] を選択すると、Amazon RDS によって VPC セキュリティグループが作成されます。詳細については、「DB クラスターの前提条件」を参照してください。
データベースポート	データベースのアクセスに使用するために、アプリケーションやユーティリティのポートを指定します。Aurora PostgreSQL DB クラスターのデフォルトの PostgreSQL ポートは 5432 になります。会社のファイアウォールでは、このポートへの接続がブロックされます。会社のファイアウォールがデフォルトのポートをブロックする場合は、新しい DB クラスター用に別のポートを選択します。

オプション	説明
DB パラメータグループ	Aurora PostgreSQL DB クラスターの DB パラメータグループを選択します。Aurora にはデフォルトの DB パラメータグループが用意されています。また、独自の DB パラメータグループを作成することもできます。DB パラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。
DB クラスターのパラメータグループ	Aurora PostgreSQL DB クラスターに使用する DB クラスターでパラメータグループを選択します。Aurora にはデフォルトの DB クラスターパラメータグループが用意されています。また、独自の DB クラスターパラメータグループを作成することもできます。DB クラスターパラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。
暗号化	[暗号を有効化] を選択すると、新しい Aurora DB クラスターが保管時に暗号化されます。[Enable encryption] (暗号化を有効化) を選択する場合は、KMS キーも AWS KMS key 値として選択してください。
優先度	DB クラスターのフェイルオーバー優先度を選択します。値を選択しない場合、デフォルト値は [tier-1] になります。この優先度により、プライマリインスタンスの障害からの復旧時に、Aurora レプリカを昇格する順序が決まります。詳細については、「Aurora DB クラスターの耐障害性」を参照してください。
バックアップの保存期間	Aurora がデータベースのバックアップコピーを保持する期間 (1 ~ 35 日) を選択します。バックアップコピーは、2 番目のデータベースに対するポイントインタイム復元 (PITR) で使用できます。

オプション	説明
拡張モニタリング	DB クラスターが実行されているオペレーティングシステムに対してリアルタイムでのメトリクスの収集を有効にするには、[拡張モニタリングを有効にする] を選択します。詳細については、「拡張モニタリングを使用した OS メトリクスのモニタリング」を参照してください。
モニタリングロール	[拡張モニタリングを有効にする] を選択した場合にのみ使用できます。拡張モニタリングに使用する AWS Identity and Access Management (IAM) ロールです。詳細については、「拡張モニタリングの設定と有効化」を参照してください。
詳細度	[拡張モニタリングを有効にする] を選択した場合にのみ使用できます。DB クラスターのメトリクスを収集する間隔を秒単位で設定します。
マイナーバージョン自動アップグレード	Aurora PostgreSQL DB クラスターで PostgreSQL DB エンジンのマイナーバージョンアップグレードをリリースと同時に自動的に受信するには、[Yes (はい)] を選択します。 [マイナーバージョン自動アップグレード] オプションは、Aurora PostgreSQL DB クラスターの PostgreSQL マイナーエンジンバージョンに対するアップグレードのみに適用されます。システム安定性を維持するために適用される定期的なパッチは適用されません。
メンテナンスウィンドウ	週 1 回のシステムメンテナンスを実行できる時間帯を選択します。

5. [Create read replica] を選択します。

AWS CLI

AWS CLI を使用してソース RDS for PostgreSQL DB インスタンスから Aurora リードレプリカを作成するには、まず、create-db-cluster CLI コマンドを使用して空の Aurora DB クラスターを作成し

ます。DB クラスターが作成されたら、create-db-instance コマンドを使用して、DB クラスターのプライマリインスタンスを作成します。プライマリインスタンスは、Aurora DB クラスターで作成される最初のインスタンスです。この場合、最初に RDS for PostgreSQL DB インスタンスの Aurora リードレプリカとして作成されます。プロセスが終了すると、RDS for PostgreSQL DB インスタンスは、Aurora PostgreSQL DB クラスターに効果的に移行されました。

メインユーザーアカウント (通常は `postgres`)、パスワード、またはデータベース名を指定する必要はありません。Aurora リードレプリカは、AWS CLI コマンドの呼び出し時に識別するソース RDS for PostgreSQL DB インスタンスからこれらを自動的に取得します。

Aurora PostgreSQL DB クラスターと DB インスタンスに使用するエンジンのバージョンを指定する必要があります。指定するバージョンは、ソース RDS for PostgreSQL DB インスタンスと一致する必要があります。ソース RDS for PostgreSQL DB インスタンスが暗号化されている場合は、Aurora PostgreSQL DB クラスタープライマリインスタンスの暗号化を指定する必要があります。暗号化されていない Aurora DB クラスターへの暗号化されたインスタンスの移行はサポートされていません。

以下の例では、`my-new-aurora-cluster` という名前の Aurora PostgreSQL DB クラスターを作成し、暗号化されていない RDS DB ソースインスタンスが使用されます。まず、`create-db-cluster` CLI コマンドを呼び出して、Aurora PostgreSQL DB クラスターを作成します。この例は、オプションの `--storage-encrypted` パラメータを使用して、DB クラスターを暗号化することを指定する方法を示しています。ソース DB は暗号化されていないため、`--kms-key-id` を使用して、使用するキーを指定します。必須のパラメータとオプションのパラメータの詳細については、例に続くリストを参照してください。

Linux、macOS、Unix の場合:

```
aws rds create-db-cluster  \
    --db-cluster-identifier my-new-aurora-cluster \
    --db-subnet-group-name my-db-subnet
    --vpc-security-group-ids sg-11111111
    --engine aurora-postgresql \
    --engine-version same-as-your-rds-instance-version \
    --replication-source-identifier arn:aws:rds:aws-region:111122223333:db/rpg-source-
db \
    --storage-encrypted \
    --kms-key-id arn:aws:kms:aws-
region:111122223333:key/11111111-2222-3333-444444444444
```

Windows の場合:

```
aws rds create-db-cluster  ^
    --db-cluster-identifier my-new-aurora-cluster ^
    --db-subnet-group-name my-db-subnet ^
    --vpc-security-group-ids sg-11111111 ^
    --engine aurora-postgresql ^
    --engine-version same-as-your-rds-instance-version ^
    --replication-source-identifier arn:aws:rds:aws-region:111122223333:db/rpg-source-
db ^
    --storage-encrypted ^
    --kms-key-id arn:aws:kms:aws-
region:111122223333:key/11111111-2222-3333-444444444444
```

次のリストに、例に示されているいくつかのオプションの詳細を示します。特に指定がない限り、これらのパラメータは必須です。

- `--db-cluster-identifier` — 新しい Aurora PostgreSQL DB クラスターに名前を付ける必要があります。
- `--db-subnet-group-name` — ソース DB インスタンスと同じ DB サブネットに Aurora PostgreSQL DB クラスターを作成します。
- `--vpc-security-group-ids` — Aurora PostgreSQL DB クラスターのセキュリティグループを指定します。
- `--engine-version` — Aurora PostgreSQL DB クラスターで使用するバージョンを指定します。これは、ソース RDS for PostgreSQL DB インスタンスで使用されるバージョンと同じである必要があります。
- `--replication-source-identifier` — Amazon リソースネーム (ARN) を使用して RDS for PostgreSQL DB インスタンスを特定します。Amazon RDS ARN の詳細については、DB クラスターの AWS 全般のリファレンス の「Amazon Relational Database Service (Amazon RDS)」を参照してください。
- `--storage-encrypted` - オプション。暗号化を次のように指定する必要がある場合にのみ使用します。
  - このパラメータは、ソース DB インスタンスに暗号化ストレージがある場合に使用します。暗号化ストレージを持つソース DB インスタンスでこのパラメータを使用しなかった場合、`create-db-cluster` の呼び出しは失敗します。ソース DB インスタンスで使用されるキーとは異なるキーを Aurora PostgreSQL DB クラスターに使用する場合は、`--kms-key-id` も指定する必要があります。

  - ソース DB インスタンスのストレージは暗号化されていないが、Aurora PostgreSQL DB クラスターで暗号化を使用する場合に使用します。その場合は、使用する暗号化キーも `--kms-key-id` パラメータで特定する必要があります。
- `--kms-key-id` - オプション。使用すると、キーの ARN、ID、エイリアス ARN、またはそのエイリアス名を使用して、ストレージの暗号化 (`--storage-encrypted`) に使用するキーを指定できます。このパラメータは、以下の状況でのみ必要です。
  - Aurora PostgreSQL DB クラスターに、ソース DB インスタンスで使用されるキーとは異なるキーを選択する。
  - 暗号化されていないソースから暗号化されたクラスターを作成する。この場合、Aurora PostgreSQL が暗号化に使用するキーを指定する必要があります。

Aurora PostgreSQL DB クラスターを作成した後、以下に示すように、`create-db-instance` CLI コマンドを使用してプライマリインスタンスを作成します。

Linux、macOS、Unix の場合:

```
aws rds create-db-instance \
    --db-cluster-identifier my-new-aurora-cluster \
    --db-instance-class db.x2g.16xlarge \
    --db-instance-identifier rpg-for-migration \
    --engine aurora-postgresql
```

Windows の場合:

```
aws rds create-db-instance ^
    --db-cluster-identifier my-new-aurora-cluster ^
    --db-instance-class db.x2g.16xlarge ^
    --db-instance-identifier rpg-for-migration ^
    --engine aurora-postgresql
```

次のリストに、例に示されているいくつかのオプションの詳細を示します。

- `--db-cluster-identifier` – 前のステップで `create-db-instance` コマンドで作成した Aurora PostgreSQL DB クラスターの名前を指定します。
- `--db-instance-class` - db.r4.xlarge、db.t4g.medium、db.x2g.16xlarge など、プライマリインスタンスに使用する DB インスタンスクラスの名前。使用可能な DB インスタンスクラスのリストについては、「DB インスタンスクラスタイプ」を参照してください。

- `--db-instance-identifier` — プライマリ・インスタンスに付ける名前を指定します。
- `--engine aurora-postgresql` — エンジンの `aurora-postgresql` を指定します。

RDS API

ソース RDS for PostgreSQL DB インスタンスから Aurora リードレプリカを作成するには、まず、RDS API オペレーション `CreateDBCluster` を使用して、ソース RDS for PostgreSQL DB インスタンスから作成される Aurora リードレプリカ用の新しい Aurora DB クラスターを作成します。Aurora PostgreSQL DB クラスターが使用可能になったら、`CreateDBInstance` を使用して、Aurora DB クラスターのプライマリインスタンスを作成します。

メインユーザーアカウント (通常は `postgres`)、パスワード、またはデータベース名を指定する必要はありません。Aurora リードレプリカは、`ReplicationSourceIdentifier` で指定されたソース RDS for PostgreSQL DB インスタンスからこれらを自動的に取得します。

Aurora PostgreSQL DB クラスターと DB インスタンスに使用するエンジンのバージョンを指定する必要があります。指定するバージョンは、ソース RDS for PostgreSQL DB インスタンスと一致する必要があります。ソース RDS for PostgreSQL DB インスタンスが暗号化されている場合は、Aurora PostgreSQL DB クラスタープライマリインスタンスの暗号化を指定する必要があります。暗号化されていない Aurora DB クラスターへの暗号化されたインスタンスの移行はサポートされていません。

Aurora リードレプリカ用の Aurora DB クラスターを作成するには、RDS API オペレーション `CreateDBCluster` を以下のパラメータを指定して使用します。

- `DBClusterIdentifier` - 作成する DB クラスターの名前。
- `DBSubnetGroupName` - この DB クラスターに関連付ける DB サブネットグループの名前。
- `Engine=aurora-postgresql` - 使用するエンジンの名前。
- `ReplicationSourceIdentifier` - ソース PostgreSQL DB インスタンスの Amazon リソースネーム (ARN)。Amazon RDS ARN の詳細については、Amazon Web Services 全般のリファレンス の「Amazon Relational Database Service (Amazon RDS)」を参照してください。`ReplicationSourceIdentifier` が暗号化されたソースを示している場合、`KmsKeyId` オプションを使用して別のキーを指定しない限り、Amazon RDS は デフォルトの KMS キーを使用します。
- `VpcSecurityGroupIds` - この DB クラスターに関連付ける Amazon EC2 VPC セキュリティグループのリスト。

- `StorageEncrypted` - DB クラスターが暗号化されるかどうかを示します。このパラメータを使用して、`ReplicationSourceIdentifier` を指定しなかった場合、Amazon RDS はデフォルトの KMS キーを使用します。
- `KmsKeyId` - 暗号化されたクラスターのキー。これを使用すると、キーの ARN、ID、エイリアス ARN、またはその エイリアス名を使用して、ストレージの暗号化に使用するキーを指定できます。

詳細については、Amazon RDS API リファレンスの「`CreateDBCluster`」を参照してください。

Aurora DB クラスターが使用可能鳴ったら、RDS API オペレーション `CreateDBInstance` を以下のパラメータを指定して使用することによって、プライマリインスタンスを作成できます。

- `DBClusterIdentifier` - DB クラスターの名前。
- `DBInstanceClass` - プライマリインスタンスに使用する DB インスタンスの名前。
- `DBInstanceIdentifier` - プライマリインスタンスの名前。
- `Engine=aurora-postgresql` - 使用するエンジンの名前。

詳細については、Amazon RDS API リファレンスの「*`CreateDBInstance`*」を参照してください。

Aurora リードレプリカの昇格

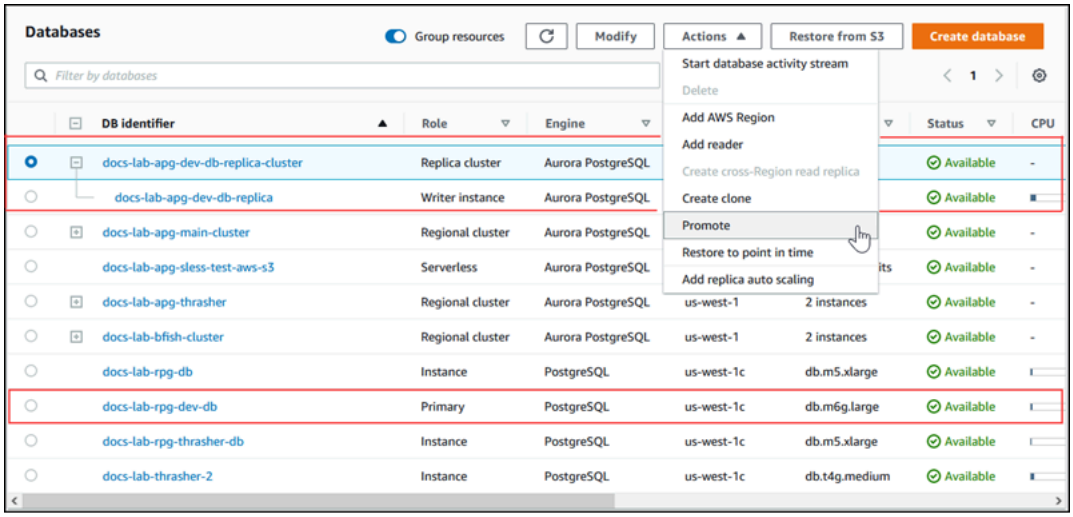
Aurora PostgreSQL への移行は、レプリカクラスターを昇格させるまで完了しないため、RDS for PostgreSQL ソース DB インスタンスをまだ削除しないようにする必要があります。

レプリカクラスターを昇格させる前に、RDS for PostgreSQL DB インスタンスに、処理中のトランザクションやデータベースへの書き込みを行うその他のアクティビティがないことを確認してください。Aurora リードレプリカのレプリカ遅延が 0 になると、レプリカクラスターを昇格させることができます。レプリカ遅延のモニタリングの詳細については、「Aurora PostgreSQL レプリケーションのモニタリング」と「Amazon Aurora のインスタンスレベルのメトリクス」を参照してください。

コンソール

Aurora リードレプリカを Aurora DB クラスターに昇格させるには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、データベースを選択します。
3. レプリカクラスターを選択します。



4. [アクション] で、[Promote (昇格)] を選択します。これには数分かかることがあり、ダウンタイムが発生する可能性があります。

プロセスが完了すると、Aurora レプリカクラスターはリージョンの Aurora PostgreSQL DB クラスターになっていて、ライターインスタンスには RDS for PostgreSQL DB インスタンスからのデータが含まれます。

AWS CLI

Aurora リードレプリカをスタンドアロン DB クラスターに昇格させるには、AWS CLI の promote-read-replica-db-cluster コマンドを使用します。

Example

Linux、macOS、Unix の場合:

```
aws rds promote-read-replica-db-cluster \
    --db-cluster-identifier myreadreplicacluster
```

Windows の場合:

```
aws rds promote-read-replica-db-cluster ^
    --db-cluster-identifier myreadreplicacluster
```

RDS API

Aurora リードレプリカをスタンドアロン DB クラスターに昇格するには、RDS API オペレーション PromoteReadReplicaDBCluster を使用します。

レプリカクラスターを昇格させたら、次のようにイベントログをチェックして、昇格が完了したことを確認できます。

Aurora レプリカクラスターが昇格したことを確認するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインの [Events] (イベント) を選択します。
3. [イベント] ページで、クラスターの名前を [ソース] リストから見つけます。各イベントには、ソース、タイプ、時間、およびメッセージがあります。AWS リージョン でアカウントに発生したすべてのイベントを確認できます。昇格が成功すると、次のメッセージが生成されます。

```
Promoted Read Replica cluster to a stand-alone database cluster.
```

昇格の完了後、ソース RDS for PostgreSQL DB インスタンスと Aurora PostgreSQL DB クラスター間のリンクは解除されます。Aurora リードレプリカのエンドポイントにクライアントアプリケーションを誘導できます。Aurora エンドポイントの詳細については、「Amazon Aurora 接続管理」を参照してください。この時点で、DB インスタンスを安全に削除できます。

Amazon Optimized Reads による Aurora PostgreSQL のクエリパフォーマンスの向上

Aurora Optimized Reads で Aurora PostgreSQL のクエリ処理を高速化できます。Aurora Optimized Reads を使用する Aurora PostgreSQL DB インスタンスは、DB インスタンスのメモリ容量を超える大規模なデータセットを持つアプリケーションのクエリレイテンシーを最大 8 倍改善し、コストを最大 30% 削減します。

トピック

- PostgreSQL の Aurora Optimized Reads の概要
- Aurora Optimized Reads の使用
- Aurora Optimized Reads のユースケース
- Aurora Optimized Reads を使用する DB インスタンスのモニタリング
- Aurora Optimized Reads のベストプラクティス

PostgreSQL の Aurora Optimized Reads の概要

Aurora Optimized Reads は、Graviton ベースの R6gd インスタンスと不揮発性メモリエクスプレス (NVMe) ストレージを備えた Intel ベースの R6id インスタンスを使用する DB クラスターを作成するときに、デフォルトで使用できます。以下の PostgreSQL バージョンから入手できます。

- 16.1 以降のすべてのバージョン
- 15.4 以降のバージョン
- 14.9 以降のバージョン

Aurora Optimized Reads は、階層型キャッシュと一時オブジェクトの 2 つの機能をサポートしています。

Optimized Reads 対応階層型キャッシュ - 階層型キャッシュを使用すると、DB インスタンスのキャッシュ容量をインスタンスメモリの 5 倍まで拡張できます。これにより、トランザクションが一貫した最新のデータがキャッシュに自動的に保持され、外部の結果セットベースのキャッシュソリューションによるデータ流通管理のオーバーヘッドからアプリケーションが解放されます。これまで Aurora ストレージからデータを取得していたクエリのレイテンシーが最大 8 倍向上します。

Aurora では、デフォルトパラメータグループでの `shared_buffers` の値は、通常、使用可能なメモリの約 75% に設定されます。ただし、r6gd および r6id インスタンスタイプの場合、Aurora は Optimized Reads キャッシュのメタデータをホストするため、`shared_buffers` スペースを 4.5% 削減します。

Optimized Reads 対応一時オブジェクト - 一時オブジェクトを使用すると、PostgreSQL によって生成された一時ファイルをローカルの NVMe ストレージに配置することで、クエリ処理を高速化できます。これにより、ネットワーク経由の Elastic Block Storage (EBS) へのトラフィックが減少します。DB インスタンスで使用可能なメモリ容量に収まらない大量のデータをソート、結合、またはマージする高度なクエリでは、レイテンシーとスループットが最大 2 倍向上します。

Aurora I/O 最適化クラスターでは、Optimized Reads は NVMe ストレージ上の階層型キャッシュと一時オブジェクトの両方を利用します。Optimized Reads 対応階層型キャッシュ機能により、Aurora はインスタンスメモリの 2 倍を一時オブジェクトに、ストレージの約 10% を内部オペレーションに、残りのストレージを階層型キャッシュとして割り当てます。Aurora スタンダードクラスターでは、Optimized Reads は一時オブジェクトのみを使用します。

エンジン	クラスターのストレージ設定	Optimized Reads 対応一時オブジェクト	Optimized Reads 対応階層型キャッシュ	サポートされるバージョン
Aurora PostgreSQL 互換エディション	規格	Yes	No	Aurora PostgreSQL バージョン 16.1 およびそれ以降のすべてのバージョン、15.4 以降、バージョン 14.9 以降
	I/O 最適化	Yes	Yes	

> **Note**
>
> NVMe ベースの DB インスタンスクラスで IO 最適化クラスターとスタンダードクラスターを切り替えると、データベースエンジンがすぐに再起動します。

Aurora PostgreSQL では、一時オブジェクトが格納されるテーブルスペースを設定するのに `temp_tablespaces` パラメータを使用します。

一時オブジェクトが設定されているかどうかを確認するには、次のコマンドを使用します。

```
postgres=> show temp_tablespaces;
temp_tablespaces
--------------------
aurora_temp_tablespace
(1 row)
```

`aurora_temp_tablespace` は、NVMe ローカルストレージを指す Aurora によって設定された表領域です。このパラメータは変更できません。また、Amazon EBS ストレージに切り替えることはできません。

Optimized Reads キャッシュがオンになっているかどうかを確認するには、次のコマンドを使用します。

```
postgres=> show shared_preload_libraries;
                shared_preload_libraries
-----------------------------------------------------
rdsutils,pg_stat_statements,aurora_optimized_reads_cache
```

Aurora Optimized Reads の使用

NVMe ベースの DB インスタンスを使用して Aurora PostgreSQL DB インスタンスをプロビジョニングすると、DB インスタンスは自動的に Aurora Optimized Reads を使用します。

RDS Optimized Reads をオンにするには、次のいずれかの操作を行います。

- NVMe ベースの DB インスタンスクラスの 1 つを使用して、Aurora PostgreSQL DB クラスターを作成します。詳細については、「Amazon Aurora DB クラスターの作成」を参照してください。
- NVMe ベースの DB インスタンスクラスの 1 つを使用して、既存の Aurora PostgreSQL DB クラスターを変更します。詳細については、「Amazon Aurora DB クラスターの変更」を参照してください。

Aurora Optimized Reads は、ローカル NVMe SSD ストレージのある DB インスタンスクラスの 1 つ以上がサポートされているすべての AWS リージョン で使用できます。詳細については、「Aurora DB インスタンスクラス」を参照してください。

最適化されていない読み取り Aurora インスタンスに戻すには、Aurora インスタンスの DB インスタンスクラスを、データベースワークロードの NVMe工フェメラルストレージがない同様のインスタンスクラスに変更します。例えば、現在の DB インスタンスクラスが db.r6gd.4xlarge の場合、db.r6g.4xlarge を選択して元に戻します。詳細については、「Aurora DB インスタンスを変更する」を参照してください。

Aurora Optimized Reads のユースケース

Optimized Reads 対応階層型キャッシュ

以下に、階層型キャッシュで Optimized Reads を使用することでメリットが得られるユースケースをいくつか紹介します。

- 支払い処理、請求、E コマースなど、厳格なパフォーマンス SLA を備えたインターネットスケールアプリケーション。
- メトリクスやデータ収集のために何百ものポイントクエリを実行するリアルタイムレポートダッシュボード。

- pgvector 拡張機能を備えた生成 AI アプリケーションにより、何百万ものベクター埋め込みから正確な近傍または最近傍を検索できます。

Optimized Reads 対応一時オブジェクト

以下に、一時オブジェクトで Optimized Reads を使用することでメリットが得られるユースケースをいくつか紹介します。

- テーブル共通式 (CTE)、派生テーブル、グループ化オペレーションを含む分析クエリ。
- アプリケーションの最適化されていないクエリを処理するリードレプリカ。
- GROUP BY や ORDER BY などの複雑な操作を伴うオンデマンドまたは動的なレポートクエリで、常に適切なインデックスを使用できるとは限らないもの。
- ソート用の `CREATE INDEX` または `REINDEX` 操作。
- 内部の一時テーブルを使用するその他のワークロード。

Aurora Optimized Reads を使用する DB インスタンスのモニタリング

Aurora Optimized Reads 対応階層型キャッシュを使用するクエリは、次の例のように EXPLAIN コマンドでモニタリングできます。

```
Postgres=> EXPLAIN (ANALYZE, BUFFERS) SELECT c FROM sbtest15 WHERE id=100000000


QUERY PLAN
-----------------------------------------------------------------------------------------
 Index Scan using sbtest15_pkey on sbtest15  (cost=0.57..8.59 rows=1 width=121) (actual
 time=0.287..0.288 rows=1 loops=1)
   Index Cond: (id = 100000000)
   Buffers: shared hit=3 read=2 aurora_orcache_hit=2
   I/O Timings: shared/local read=0.264
 Planning:
   Buffers: shared hit=33 read=6 aurora_orcache_hit=6
   I/O Timings: shared/local read=0.607
 Planning Time: 0.929 ms
 Execution Time: 0.303 ms
(9 rows)
Time: 2.028 ms
```

> **Note**
>
> 説明プランの Buffers セクションにある aurora_orcache_hit および aurora_storage_read フィールドは、Optimized Reads が有効で、値が 0 より大きい場合にのみ表示されます。読み取りフィールドは、aurora_orcache_hit と aurora_storage_read フィールドの合計です。

Aurora Optimized Reads を使用する DB インスタンスは、次の CloudWatch メトリクスでモニタリングできます。

- AuroraOptimizedReadsCacheHitRatio
- FreeEphemeralStorage
- ReadIOPSEphemeralStorage
- ReadLatencyEphemeralStorage
- ReadThroughputEphemeralStorage
- WriteIOPSEphemeralStorage
- WriteLatencyEphemeralStorage
- WriteThroughputEphemeralStorage

これらのメトリクスでは、利用可能なインスタンスストアストレージ、IOPS、スループットに関するデータを提供します。これらのメトリクスの詳細については、「Amazon Aurora のインスタンスレベルのメトリクス」を参照してください。

pg_proctab 拡張機能を使用して NVMe ストレージをモニタリングすることもできます。

```
postgres=>select * from pg_diskusage();

major | minor |       devname       | reads_completed | reads_merged | sectors_read |
 readtime | writes_completed | writes_merged | sectors_written | writetime | current_io
 | iotime  | totaliotime
------+-------+---------------------+-----------------+--------------+--------------
+----------+------------------+---------------+-----------------+-----------
+------------+---------+-------------
      |       | rdstemp             |           23264 |            0 |       191450 |
  11670 |          1750892 |             0 |        24540576 |    819350 |          0 |
 3847580 |      831020
```

```
      |       | rdsephemeralstorage |           23271 |            0 |       193098 |
   2620 |           114961 |             0 |        13845120 |    130770 |          0 |
  215010 |      133410
(2 rows)
```

Aurora Optimized Reads のベストプラクティス

Aurora Optimized Reads を使用するベストプラクティスは次のとおりです。

- CloudWatch メトリクスの `FreeEphemeralStorage` を使用して、インスタンスストアで使用可能なストレージ容量をモニタリングします。DB インスタンスのワークロードが原因でインスタンスストアが上限に達している場合は、同時実行数と一時オブジェクトを頻繁に使用するクエリを調整するか、より大きな DB インスタンスクラスを使用するように変更します。
- CloudWatch メトリクスをモニタリングして、Optimized Reads キャッシュヒットレートを確認します。VACUUM のようなオペレーションでは、多数のブロックを非常に素早く変更します。これにより、ヒットレートが一時的に低下する可能性があります。`pg_prewarm` 拡張機能を使用すると、データをバッファキャッシュにロードできます。これにより、Aurora はそれらのブロックの一部を Optimized Reads キャッシュに事前に書き込むことができます。
- クラスターキャッシュ管理 (CCM) を有効にすると、フェイルオーバーターゲットとして使用される Tier-0 リーダーのバッファキャッシュと階層型キャッシュをウォームアップできます。CCM を有効にすると、バッファキャッシュが定期的にスキャンされ、エビクションの対象となるページが階層型キャッシュに書き込まれます。CCM の詳細については、「Aurora PostgreSQL のクラスターキャッシュ管理によるフェイルオーバー後の高速リカバリ」を参照してください。

Babelfish for Aurora PostgreSQL の使用

Babelfish for Aurora PostgreSQL は、SQL Server クライアントからデータベース接続を受け入れる機能を使用して Aurora PostgreSQL DB クラスターを拡張します。Babelfish より、従来の移行に比べて少ないコード変更で、またデータベースドライバを変更せずに、元々 SQL Server 用に構築されているアプリケーションを直接 Aurora PostgreSQL で利用できるようになります。行に関する詳細については、SQL Server データベースを Babelfish for Aurora PostgreSQL に移行する を参照してください。

Babelfish は、Aurora PostgreSQL データベースクラスターに追加のエンドポイントを提供します。これにより Aurora PostgreSQL データベースクラスターは SQL Server ワイヤレベルプロトコルと一般的に使用される SQL Server ステートメントを理解することができます。テーブル式データストリーミング (TDS) ワイヤプロトコルを使用するクライアントアプリケーションは Aurora PostgreSQL の TDS リスナーポートにネイティブ接続できます。TDS の詳細については、Microsoft ウェブサイトの「[MS-TDS]: 表形式データストリームプロトコルを参照してください。

> **Note**
>
> Babelfish for Aurora PostgreSQL は TDS バージョン 7.1 から 7.4 をサポートしています。

Babelfish は、PostgreSQL 接続を使用してデータへのアクセスも提供します。デフォルトでは、Babelfish でサポートされている両方の SQL 言語は、次のポートでネイティブのワイヤプロトコルを通じて使用できます。

- SQL Server 言語 (T-SQL)、クライアントはポート 1433 に接続します。
- PostgreSQLg 言語 (PL/pgSQL) ダイアレクト、クライアントはポート 5432 に接続します。

Babelfish は Transact-SQL (T-SQL) 言語をいくつかの違いで実行します。詳しくは、「Babelfish for Aurora PostgreSQL と SQL Server の違い」を参照してください。

以下のセクションでは、Babelfish for Aurora PostgreSQL DB クラスターの設定と使用に関する情報を示します。

トピック

- Babelfish の制限事項
- Babelfish のアーキテクチャと構成を理解する

- Babelfish for Aurora PostgreSQL DB クラスターの作成
- SQL Server データベースを Babelfish for Aurora PostgreSQL に移行する
- Babelfish for Aurora PostgreSQL を使用したデータベース認証
- Babelfish DB クラスターへの接続
- Babelfish での作業
- Babelfish のトラブルシューティング
- Babelfish をオフにする
- Babelfish バージョンのアップデート
- Babelfish for Aurora PostgreSQL リファレンス

Babelfish の制限事項

Babelfish for Aurora PostgreSQL には、現在、以下のような制限事項があります。

- 現在、Babelfish は、以下の Aurora 機能をサポートしていません。
  - Amazon RDS ブルー/グリーンデプロイ
  - AWS Identity and Access Management
  - データベースアクティビティストリーム (DAS)
  - PostgreSQL 論理的なレプリケーション
  - Aurora PostgreSQL Serverless v2 とプロビジョニングされた RDS Data API
  - RDS for SQL Server を使用した RDS Proxy
  - SCRAM (Salted Challenge Response Authentication Mechanism)
  - クエリエディタ
- Babelfish は現在、Active Directory グループ向けの Kerberos ベースの認証をサポートしていません。
- Babelfish では、以下のクライアントドライバー API のサポートは提供していません。
  - Microsoft 分散トランザクションコーディネーター (MSDTC) に関連する接続属性を持つ API リクエストはサポートされていません。これには、SQL サーバー JDBC ドライバーの SQLServerXAResource クラスによる XA コールが含まれます。
  - Babelfish は、最新バージョンの TDS プロトコルを使用するドライバーとの接続プーリングをサポートしています。以前のドライバーでは、接続プールに関連する接続属性やメソッドによる API リクエストはサポートされていません。
- 現在、Babelfish は、以下の Aurora PostgreSQL 拡張機能をサポートしていません。
  - `bloom`
  - `btree_gin`
  - `btree_gist`
  - `citext`
  - `cube`
  - `hstore`
  - `hypopg`
  - `pglogical` を使用した論理的なレプリケーション
  - `ltree`

  - `pgcrypto`
  - `apg_plan_mgmt` を使用するクエリ計画管理

  PostgreSQL の拡張機能の詳細については、「エクステンションと外部データラッパーの使用」を参照してください。
- Microsoft JDBC ドライバーの代替として設計されたオープンソースの jTDS ドライバーはサポートされていません。

Babelfish のアーキテクチャと構成を理解する

Babelfish を実行する Aurora PostgreSQL 互換エディション DB クラスターは、Aurora DB クラスターと同様に管理します。つまり、スケーラビリティ、フェイルオーバーサポートによる高可用性、Aurora DB クラスターによって提供される組み込みレプリケーションを利用できます。これらの機能の詳細については、「Aurora DB クラスターのパフォーマンスとスケーリングの管理」、「Amazon Aurora の高可用性」、および「Amazon Aurora でのレプリケーション」を参照してください。次を含む、他の多くの AWS ツールおよびユーティリティにもアクセスできます。

- Amazon CloudWatch は、データと実用的なインサイトを提供するモニタリングおよび可観測性サービスです。詳しくは、「Amazon CloudWatch を使用した Amazon Aurora メトリクスのモニタリング」を参照してください。
- RDS Performance Insights は、データベースのロードを迅速に評価するのに役立つ、データベースのパフォーマンスチューニングおよびモニタリング特徴です。詳細については、「Amazon Aurora での Performance Insights を使用したDB 負荷のモニタリング」を参照してください。
- Aurora Global Database は複数の AWS リージョン にまたがり配置されます。これにより、低レイテンシーのグローバル読み取りを実現し、AWS リージョン 全体に影響が及ぶ可能性のある停止がまれに起きても、すばやい復旧を可能にします。詳しくは、「Amazon Aurora Global Database の使用」を参照してください。
- 自動ソフトウェアパッチを適用すると、利用可能な状態になったときに最新のセキュリティパッチと特徴パッチを適用して、データベースを最新の状態に保ちます。
- Amazon RDS イベントは、自動フェイルオーバーなどの重要なデータベースイベントを E メールまたは SMS メッセージで通知します。詳しくは、「Amazon Aurora イベントのモニタリング」を参照してください。

以下では、Babelfish アーキテクチャと、移行する SQL Server データベースが Babelfish によってどのように処理されるかについて学習します。Babelfish DB クラスターを作成するときは、1 つのデー

タベース、または複数のデータベース、照合順序、その他の詳細についていくつかの決定を下す必要があります。

トピック

Babelfish のアーキテクチャ

Babelfish を有効にして Aurora PostgreSQL クラスターを作成すると、Aurora は `babelfish_db` という名前の PostgreSQL データベースを使用してクラスターをプロビジョンします。このデータベースは、移行されたすべての SQL Server オブジェクトおよび構造が存在する場所です。

> **Note**
>
> Aurora PostgreSQL クラスターでは、`babelfish_db` データベース名は Babelfish 用に予約されています。Babelfish DB クラスターの Babelfish に独自の「babelfish_db」データベースを作成すると、Aurora は正常に Babelfish のプロビジョニングができなくなります。

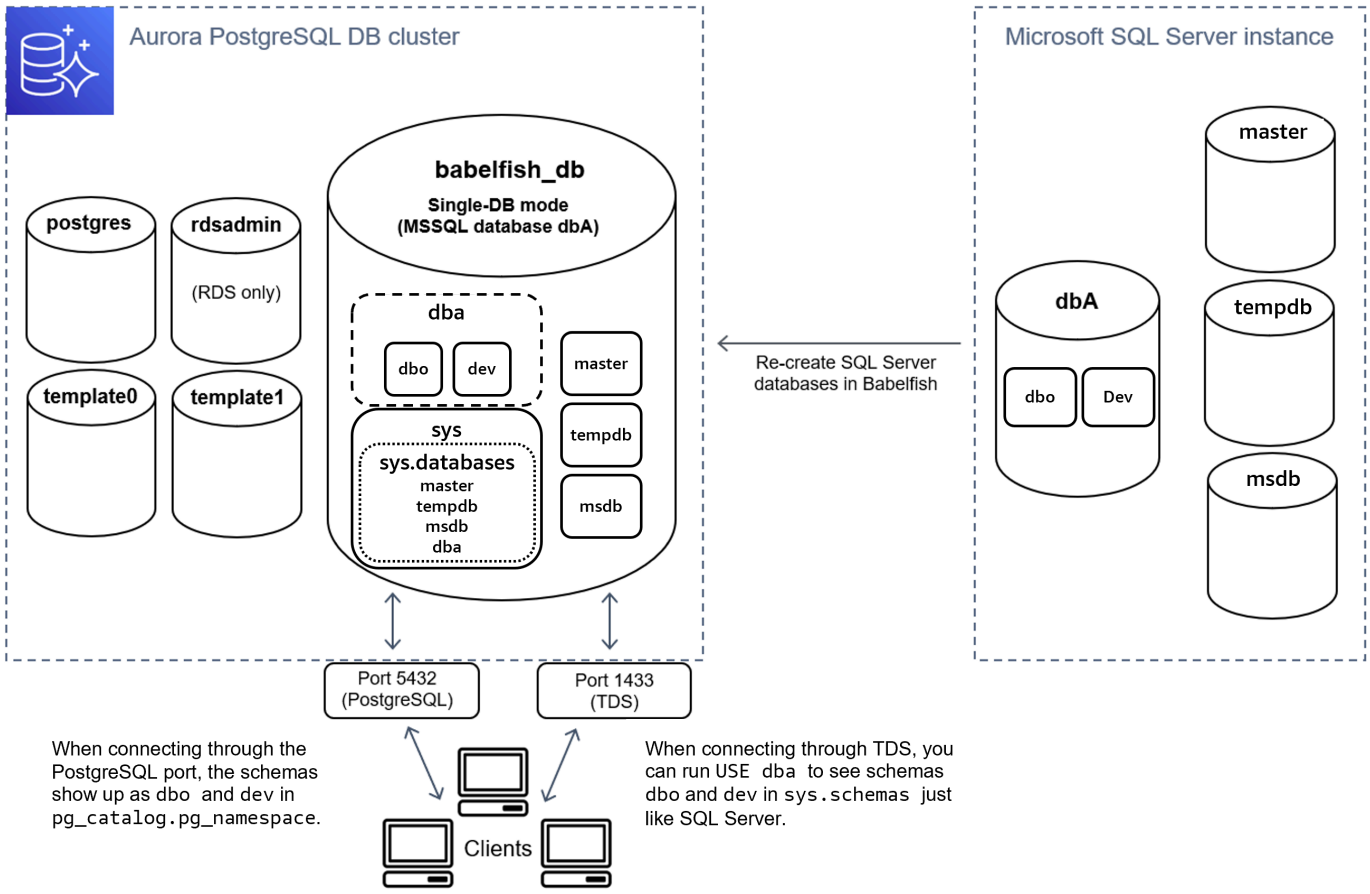
TDS ポートに接続すると、セッションは `babelfish_db` データベースに置かれます。T-SQL からは、構造は SQL Server インスタンスに接続されている状態と似た見た目になります。`master`、`msdb`、および `tempdb` データベース、そして `sys.databases` カタログが確認できます。USE ステートメントを使用して追加のユーザーデータベースを作成し、データベース間で切り替えることができます。SQL Server ユーザーデータベースを作成すると、そのデータベースは `babelfish_db` PostgreSQL データベースにフラット化されます。ユーザーのデータベースは、SQL Server が提供するものと等しい、またはそれに類するクロスデータベース構文およびセマンティクスを保持します。

1 つのデータベースまたは複数のデータベースでの babelfish の使用

Babelfish で使用する Aurora PostgreSQL クラスターを作成する場合は、単独で 1 つの SQL Server データベースを使用するか、複数の SQL Server データベースを一緒に使用するかを選択します。選択した内容は、`babelfish_db` データベース内の SQLServer スキーマの名前が Aurora PostgreSQL からどのように表示されるかに影響します。移行モードは `migration_mode` パラメータに保存され

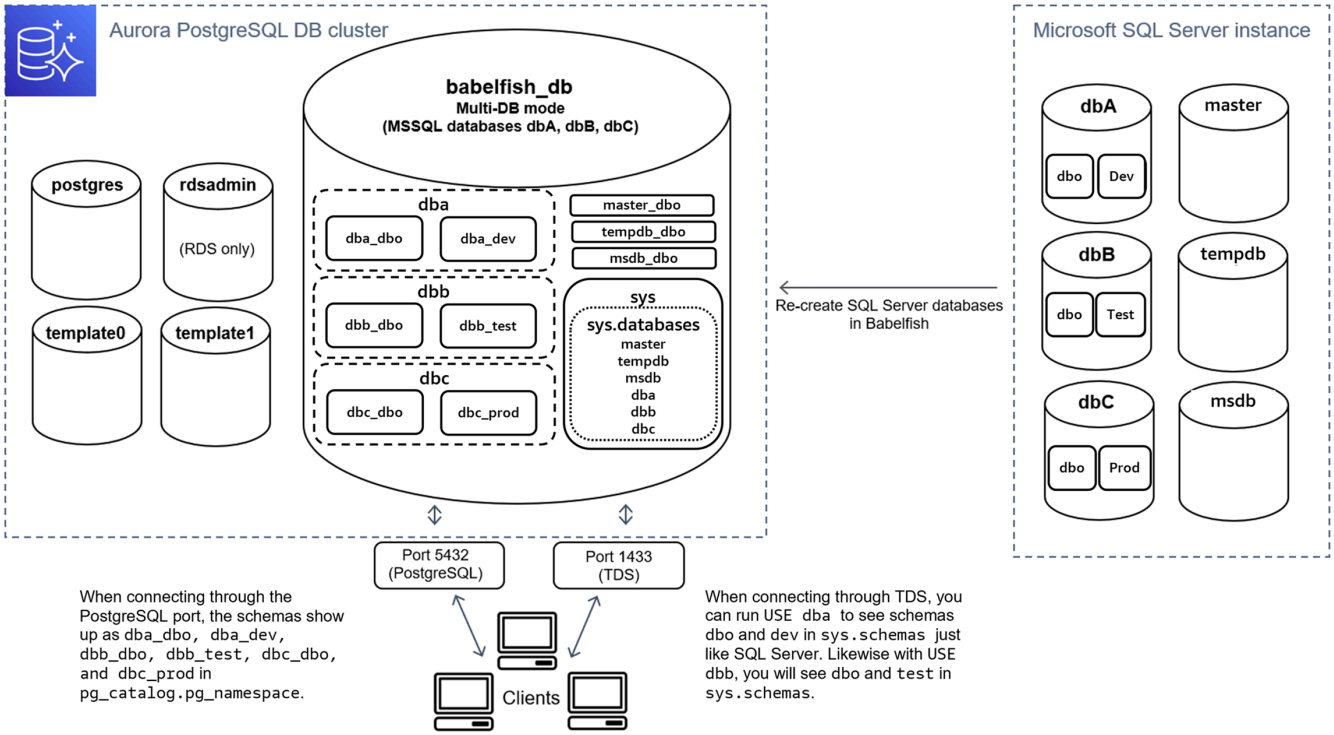
ます。以前に作成したすべての SQL オブジェクトにアクセスできなくなる可能性があるため、クラスターの作成後にこのパラメータを変更しないでください。

シングルデータベースモードでは、SQL Server データベースのスキーマ名は、PostgreSQL の babelfish_db データベースと同じままです。1 つのデータベースだけを移行する場合には、移行したユーザーデータベースのスキーマ名は、SQL Server で使用されているのと同じ名前の PostgreSQL で参照できます。例えば、dbo と smith スキーマは dbAデータベース内に存在します。



TDS 経由で接続すると、SQL Server の場合と同様に、USE dba を実行して T-SQL から スキーマ dbo と dev を表示することができます。変更されていないスキーマ名は PostgreSQL からも確認できます。

マルチデータベースモードでは、PostgreSQL からアクセスされたとき、ユーザーデータベースのスキーマ名は dbname_schemaname になります。T-SQL からアクセスされたときのスキーマ名は変わりません。



図のように、TDS ポート経由で接続し T-SQL を使用する場合、マルチデータベースモードとシングルデータベースモードは SQL Server と同じです。例えば、USE dbA はスキーマ dbo と dev を SQL Serverの場合と同じように列挙します。dba_dbo や dba_dev などのマップされたスキーマ名は、PostgreSQL から確認できます。

各データベースには引き続きスキーマが含まれます。各データベースの名前は、以下の例のようにアンダースコアを区切り文字として、SQL Server スキーマ名に付加されます。

- dba が dba_dbo と dba_dev を含む。
- dbb が dbb_dbo と dbb_test を含む。
- dbc が dbc_dbo と dbc_prod を含む。

babelfish_db データベース内で、T-SQL ユーザーは引き続き USE dbname を実行してデータベースコンテキストを変更する必要があります。そうすることで外観が SQL Server と似たままになります。

移行モードの選択

各移行モードにはメリットとデメリットがあります。使用しているユーザーデータベース数と移行計画に基づいて、移行モードを選択します。Babelfish で使用するクラスターの作成後は、以前に作成

したすべての SQL オブジェクトにアクセスできなくなる可能性があるため、移行モードを変更しないでください。移行モードを選択する際は、ユーザーデータベースとクライアントの要件を考慮してください。

Babelfish で使用するクラスターを作成すると、Aurora PostgreSQL はシステムデータベース、`master` および `tempdb` を作成します。システムデータベース (`master` または `tempdb`) のオブジェクトを作成または変更した場合、新しいクラスターにこれらのオブジェクトを必ず再作成してください。SQL Server と異なり、Babelfish は クラスターの再起動後、 `tempdb` を再初期化しません。

シングルデータベース移行モードは、以下の場合に使用します。

- 単一の SQL Server データベースを移行する場合。シングルデータベースモードでは、PostgreSQL からアクセスされたときの移行されたスキーマ名は、元の SQL Server スキーマ名と同じです。これにより、PostgreSQL 接続で実行するように最適化する場合に、既存の SQL クエリへのコード変更を減らすことができます。
- 最終目標がネイティブ Aurora PostgreSQL への完全移行である場合。移行する前にスキーマを 1 つのスキーマに統合 (dbo) し、それから単一のクラスターに移行すると、必要な変更が少なく済みます。

次の場合は、複数のデータベース移行モードを使用します。

- 同じインスタンス内の複数のユーザーデータベースでデフォルトの SQL Server エクスペリエンスを実現する場合。
- 複数のユーザーデータベースを一緒に移行する必要がある場合。

Babelfish の DB クラスターパラメータグループ設定

Aurora PostgreSQL DB クラスターを作成し、[Babelfish をオンにする] を選択した場合、[新規作成] を作成すると、DB クラスターパラメータグループが自動的に作成されます。この DB クラスターパラメータグループは、インストール用に選択された Aurora PostgreSQL DB バージョンの Aurora PostgreSQL DB クラスターパラメータグループ (例えば、Aurora PostgreSQL DB バージョン 14) に基づいています。これは、次の一般的なパターンを使用して命名されています。

```
custom-aurora-postgresql14-babelfish-compat-3
```

クラスター作成プロセス中に次の設定を変更できますが、カスタムパラメータグループに保存するとこれらの一部は変更できないため、慎重に選択してください。

- 単一データベースまたは複数のデータベース
- デフォルトの照合ロケール
- 照合順序名
- DB パラメータグループ

既存の Aurora PostgreSQL DB クラスターバージョン 13 以降のパラメータグループを使用するには、グループを編集して `babelfish_status` パラメータを on に設定します。Aurora PostgreSQL クラスターを作成する前に、Babelfish オプションを指定してください。詳細については、「「パラメータグループを使用する」」を参照してください。

次のパラメータは、Babelfish のプリファレンスを制御します。説明に特に明記されていない限り、パラメータは変更可能です。デフォルト値は説明に含まれています。任意のパラメータの許容値を表示するには、次の手順を実行します。

> **Note**
>
> 新しい DB パラメータグループを DB インスタンスに関連付ける場合、変更された静的パラメータと動的パラメータは、DB インスタンスが再起動された後にのみ適用されます。ただし、DB インスタンスに関連付けた後に DB パラメータグループの動的パラメータを変更すると、これらの変更は再起動せずに直ちに適用されます。

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。

2. ナビゲーションペインから [パラメータグループ] を選択します。
3. リストから、`default.aurora-postgresql14` DB クラスターのパラメータグループを選択します。
4. 検索フィールドにパラメータの名前を入力します。例えば、検索フィールドに `babelfishpg_tsql.default_locale` を入力して、このパラメータとそのデフォルト値、および許容される設定を表示します。

パラメータ	説明	適用タイプ	変更可能
babelfishpg_tds.tds_default_numeric_scale	エンジンがスケールを指定していない場合、TDS 列メタデータで送信される数値型のデフォルトスケールを設定します。(デフォルト: 8) (許容値: 0 ~ 38)	dynamic	true
babelfishpg_tds.tds_default_numeric_precision	エンジンが精度を指定していない場合、TDS 列のメタデータで送信される数値型のデフォルト精度を設定する整数。(デフォルト: 38) (許容値: 1 ~ 38)	dynamic	true
babelfishpg_tds.tds_default_packet_size	SQL Server クライアントを接続するためのデフォルトのパケットサイズを設定する整数。(デフォルト: 4096) (許容値: 512 ~ 32767)	dynamic	true

パラメータ	説明	適用タイプ	変更可能
babelfishpg_tds.tds_default_protocol_version	クライアントを接続するためのデフォルトの TDS プロトコルバージョンを設定する整数。(デフォルト: DEFAULT) (tdsv7.0、 tdsv7.1、tdsv7.1.1、 tdsv7.2、tdsv7.3a、 tdsv7.3b、tdsv7.4、DEFAULT)	dynamic	true
babelfishpg_tds.default_server_name	Babelfish サーバーのデフォルト名を識別する文字列。(デフォルト: Microsoft SQL Server) (許容値: null)	dynamic	true
babelfishpg_tds.tds_debug_log_level	TDS のロギングレベルを設定します。0 はロギングをオフにする整数。(デフォルト: 1) (許容値: 0、1、2、3)	dynamic	true
babelfishpg_tds.listen_addresses	TDS をリッスンするためのホスト名、IP アドレス、またはアドレスを設定する整数。Babelfish DB クラスターの作成後にこのパラメータを変更することはできません。(デフォルト: *) (許容値: null)	–	false

パラメータ	説明	適用タイプ	変更可能
babelfishpg_tds.port	SQL Server 構文でのリクエストに使用する TCP ポートを設定する整数。(デフォルト: 1433) (許容値: 1 ~ 65535)	static	true
babelfishpg_tds.tds_ssl_encrypt	TDS リスナーポートを通過するデータの暗号化をオン (0) またはオフ (1) にするブール値。SSL を使用したクライアント接続の詳細については、Babelfish SSL 設定とクライアント接続 を参照してください。(デフォルト: 00) (許容値: 0、1)	dynamic	true
babelfishpg_tds.tds_ssl_max_protocol_version	TDS セッションに使用する最高の SSL/TLS プロトコルバージョンを指定する文字列。(デフォルト: 'TLSv1.2') (許容値: 'TLSv1'、'TLSv1.1'、'TLSv1.2')	dynamic	true

パラメータ	説明	適用タイプ	変更可能
babelfishpg_tds.tds_ssl_min_protocol_version	TDS セッションに使用する最低の SSL/TLS プロトコルバージョンを指定する文字列。(デフォルト: Aurora PostgreSQL バージョン 16 の「TLSv1.2」、Aurora PostgreSQL バージョン 16 より古いバージョンの「TLSv1」) (許容可能:「TLSv1」、「TLSv1.1」、「TLSv1.2」)	dynamic	true
babelfishpg_tds.unix_socket_directories	TDS サーバーの Unix ソケットディレクトリを識別する文字列。Babelfish DB クラスターの作成後にこのパラメータを変更することはできません。(デフォルト: /tmp) (許容値: null)	–	false
babelfishpg_tds.unix_socket_group	TDS サーバーの Unix ソケットグループを識別する文字列。Babelfish DB クラスターの作成後にこのパラメータを変更することはできません。(デフォルト: rdsdb) (許容値: null)	–	false

パラメータ	説明	適用タイプ	変更可能
babelfishpg_tsql.default_locale	Babelfish 照合に使用されるデフォルトのロケールを指定する文字列。デフォルトのロケールはロケールで、修飾子は含まれません。 Babelfish DB クラスターをプロビジョニングするときに、このパラメータを設定します。DB クラスターのプロビジョニング後、このパラメータへの変更は無視されます。(デフォルト: en_US) (許容値: tables を参照)	static	true

パラメータ	説明	適用タイプ	変更可能
babelfishpg_tsql.migration_mode	単一または複数のユーザーデータベースのサポートを指定する変更不可能なリスト。Babelfish DB クラスターをプロビジョニングするときに、このパラメータを設定します。DB クラスターのプロビジョニング後は、このパラメータの値を変更できません。(デフォルト: Aurora PostgreSQL バージョン 16 の multi-db、Aurora PostgreSQL バージョン 16 以前のバージョンでは single-db) (許容値: single-db、multi-db、null)	static	true

パラメータ	説明	適用タイプ	変更可能
babelfishpg_tsql.server_collation_name	サーバーレベルのアクションに使用される照合の名前を指定する文字列。Babelfish DB クラスターをプロビジョニングするときに、このパラメータを設定します。DB クラスターのプロビジョニング後は、このパラメータの値を変更しないでください。(デフォルト: bbf_unicode_general_ci_as) (許容値: tables を参照)	static	true
babelfishpg_tsql.version	@@VERSION 変数の出力を設定する文字列。Aurora PostgreSQL DB クラスターのこの値は変更しないでください。(デフォルト: null) (許容値: デフォルト)	dynamic	true

パラメータ	説明	適用タイプ	変更可能
rds.babelfish_status	Babelfish 機能の状態を設定する文字列。このパラメータが `datatypesonly` に設定されている場合、Babelfish はオフになりますが、SQL Server のデータ型は引き続き使用できます。(デフォルト: off) (許容値: on、off、datatypesonly)	static	true
unix_socket_permissions	TDS サーバーの Unix ソケットパーミッションを設定する整数。Babelfish DB クラスターの作成後にこのパラメータを変更することはできません。(デフォルト: 0700) (許容値: 0 ~ 511)	–	false

Babelfish SSL 設定とクライアント接続

クライアントが TDS ポート (デフォルト 1433) に接続するとき、Babelfish はクライアントハンドシェイク中に送信された Secure Sockets Layer (SSL) 設定を Babelfish SSL パラメータ設定 (`tds_ssl_encrypt`) と比較します。その後 Babelfish は接続が許可されているかどうかを判断します。接続が許可されている場合、パラメータ設定およびクライアントが提供する暗号化のサポートに応じて、暗号化動作が強制されるかどうかが決まります。

次の表は、各組み合わせで Babelfish がどのように動作するかを示します。

クライアント SSL の設定	Babelfish SSL 設定	接続は許可されていますか？	クライアントに返された値
ENCRYPT_OFF	tds_ssl_encrypt=0	許可されている場合、ログインパケットは暗号化されます	ENCRYPT_OFF
ENCRYPT_OFF	tds_ssl_encrypt=1	許可されている場合、接続全体が暗号化されます	ENCRYPT_REQ
ENCRYPT_ON	tds_ssl_encrypt=0	許可されている場合、接続全体が暗号化されます	ENCRYPT_ON
ENCRYPT_ON	tds_ssl_encrypt=1	許可されている場合、接続全体が暗号化されます	ENCRYPT_ON

クライアント SSL の設定	Babelfish SSL 設定	接続は許可されていますか？	クライアントに返された値
ENCRYPT_NOT_SUP	tds_ssl_encrypt=0	はい	ENCRYPT_NOT_SUP
ENCRYPT_NOT_SUP	tds_ssl_encrypt=1	いいえ、接続は閉じました	ENCRYPT_REQ
ENCRYPT_REQ	tds_ssl_encrypt=0	許可されている場合、接続全体が暗号化されます	ENCRYPT_ON
ENCRYPT_REQ	tds_ssl_encrypt=1	許可されている場合、接続全体が暗号化されます	ENCRYPT_ON
ENCRYPT_CLIENT_CERT	tds_ssl_encrypt=0	いいえ、接続は閉じました	サポートされていません
ENCRYPT_CLIENT_CERT	tds_ssl_encrypt=1	いいえ、接続は閉じました	サポートされていません

Babelfish がサポートする照合順序

Babelfish で Aurora PostgreSQL DB クラスターを作成するときは、データの照合を選択します。照合順序は、特定の人間言語で書かれたテキストまたは文字を生成するソート順とビットパターンを指定します。照合順序には、特定のビットパターンのセットに対してデータを比較するルールが含まれます。照合順序はローカリゼーションに関連しています。ロケールが異なると、文字マッピング、ソート順などに影響します。照合順序の属性は、さまざまな照合順序の名前に反映されます。属性の詳細については、「Babelfish collation attributes table」を参照してください。

Babelfish は、Babelfish が提供する同等の照合に SQL Server 照合をマッピングします。Babelfish は、文化的に敏感な文字列比較とソート順で、 Unicode 照合を事前に定義しています。また Babelfish は、SQL Server DB 内の照合を最も近い Babelfish 照合に変換する方法も提供します。ロケール固有の照合順序は、異なる言語および地域に対して提供されます。

一部の照合では、クライアント側のエンコーディングに対応したコードページを指定します。Babelfish は、各出力列の照合順序に応じて、サーバーエンコーディングからクライアントエンコーディングに自動的に変換します。

Babelfish は、「Babelfish supported collations table」に列挙された照合順序をサポートしています。Babelfish は、Babelfish が提供する同等の照合に SQL Server 照合をマッピングします。

Babelfish は International Components for Unicode (ICU) 照合順序ライブラリのバージョン 153.80 を使用しています。ICU 照合の詳細については、ICU ドキュメントの「Collation」(照合順序) を参照してください。PostgreSQL と照合の詳細については、PostgreSQL ドキュメントの「照合順序のサポート」を参照してください。

トピック

照合順序と ロケールを制御する DB クラスターパラメータ

次のパラメータは、照合動作に影響します。

babelfishpg_tsql. default_locale

このパラメータは、照合で使用されるデフォルトの口ケールを指定します。このパラメータは、Babelfish collation attributes table に列挙された照合属性と組み合わせて使用されるので、特定の言語およびリージョンの照合順序をカスタマイズすることができます。このパラメータのデフォルト値は en-US です。

デフォルトの口ケールは、BBF 文字でスタートすべての Babelfish 照合順序名と、Babelfish 照合順序にマッピングされるすべての SQL Server 照合順序に適用されます。既存の Babelfish DB クラスターでこのパラメータの設定を変更しても、既存の照合順序の口ケールには影響しません。照合の一覧については、「Babelfish supported collations table」を参照してください。

babelfishpg_tsql.server_collation_name

このパラメータは、サーバー (Aurora PostgreSQL DB クラスターインスタンス) とデータベースのデフォルトの照合順序を指定します。デフォルト値は、「sql_latin1_general_cp1_ci_as」です。T-SQL では、サーバーの照合順序によって識別子の比較方法が決定されるため、CI_AS 照合順序は server_collation_name でなければならなりません。

Babelfish DB クラスターを作成するときは、選択リストから [照合順序名] を選択します。これらには、「Babelfish supported collations table」に列挙された照合順序が含まれます。belfish データベースの作成後は、server_collation_name を変更しないでください。

Babelfish for Aurora PostgreSQL DB クラスターの作成時に選択した設定は、これらのパラメータのクラスターに関連付けられた DB クラスターパラメータグループに格納され、照合動作を設定します。

決定論的照合および非決定論的照合と Babelfish

Babelfish は、決定論的照合と非決定論的照合をサポートしています:

- 決定的照合順序は、同じバイトシーケンスを持つ文字を等価として評価します。つまり、x と X は決定論的照合では等しくありません。確定的照合順序は、大文字と小文字の区別 (CS) とアクセントの区別 (AS) をします。
- 非決定照合順序 は完全な一致を必要としません。非決定性照合は x と X を 同等に評価します。非決定的照合は 大文字と小文字を区別せず (CI)、アクセントを無視 (AI) します。

次の表に、非決定的照合順序を使用する場合の Babelfish と PostgreSQL の動作の違いをいくつか示します。

Babelfish	PostgreSQL
CI_AS 照合順序の LIKE 句をサポートします。	非決定的照合順序に関する LIKE 句はサポートしていません。
AI 照合に関する LIKE 句をサポートしていません。	
非決定性照合順序に対するパターンマッチング操作もサポートしていません。	

SQL Server および PostgreSQL と比較した Babelfish の他の制限および動作の違いについては、「照合順序の制限と動作の違い」を参照してください。

Babelfish と SQL Server は次の表に示すように、照合属性を記述する照合順序の命名規則に従います。

属性	説明
AI	アクセントは区別されません。
AS	アクセントは区別されます。
BIN2	BIN2 は、データをコードポイント順に並べ替えるよう要求します。Unicode コードポイントの順序は、UTF-8、UTF-16、UCS-2 のエンコーディングで同じ文字順序になります。コードポイントの順序は、高速確定的な照合順序です。
CI	大文字と小文字を区別しない。
CS	大文字と小文字を区別する。
PREF	小文字の前に大文字を並べるには、PREF 照合を使用します。比較で大文字と小文字を区別しない場合、もし他の区別がないのであれば、大文字バージョンの文字は小文字バージョンの前にソートされます。ICU ライブラリでは、`colCaseFirst=upper` で大文字の設定がサポートされています。ただし、CI_AS 照合では使用できません。

属性	説明
	PREF は CS_AS 決定論的照合順序にのみ適用できます。

Babelfish がサポートする照合順序

サーバー照合またはオブジェクトの照合として、次の照合順序を使用します。

照合順序 ID	メモ
bbf_unicode_general_ci_as	大文字と小文字を区別しない比較と LIKE 演算子をサポートします。
bbf_unicode_cp1_ci_as	CP1252 とも呼ばれる 非決定照合順序。
bbf_unicode_CP1250_ci_as	ラテン文字を使用する中央ヨーロッパおよび東ヨーロッパ言語のテキストを表すために使用される非決定照合順序。
bbf_unicode_CP1251_ci_as	キリル文字を使用する言語の非決定照合順序。
bbf_unicode_cp1253_ci_as	現代ギリシャ語を表すのに使われる非決定照合順序。
bbf_unicode_cp1254_ci_as	トルコ語をサポートする非決定照合順序。
bbf_unicode_cp1255_ci_as	ヘブライ語をサポートする非決定照合順序。
bbf_unicode_cp1256_ci_as	アラビア語の文字を使う言語を書くのに使われる非決定照合順序。
bbf_unicode_cp1257_ci_as	エストニア語、ラトビア語、リトアニア語をサポートするために使用される非決定照合順序。

照合順序 ID	メモ
bbf_unicode_cp1258_ci_as	ベトナム語の文字を書くのに使われる非決定照合順序。
bbf_unicode_cp874_ci_as	タイ語の文字を書くのに使われる非決定照合順序
sql_latin1_general_cp1250_ci_as	ラテン文字を表すために使用される非決定的なシングルバイト文字エンコード。
sql_latin1_general_cp1251_ci_as	ラテン文字をサポートしている非決定照合順序。
sql_latin1_general_cp1_ci_as	ラテン文字をサポートしている非決定照合順序。
sql_latin1_general_cp1253_ci_as	ラテン文字をサポートしている非決定照合順序。
sql_latin1_general_cp1254_ci_as	ラテン文字をサポートしている非決定照合順序。
sql_latin1_general_cp1255_ci_as	ラテン文字をサポートしている非決定照合順序。
sql_latin1_general_cp1256_ci_as	ラテン文字をサポートしている非決定照合順序。
sql_latin1_general_cp1257_ci_as	ラテン文字をサポートしている非決定照合順序。
sql_latin1_general_cp1258_ci_as	ラテン文字をサポートしている非決定照合順序。

照合順序 ID	メモ
chinese_prc_ci_as	中国語 (PRC) をサポートする非決定的照合順序。
cyrillic_general_ci_as	キリル文字をサポートする非決定的照合順序。
finnish_swedish_ci_as	フィンランド語をサポートする非決定的照合順序。
french_ci_as	フランス語をサポートする非決定的照合順序。
japanese_ci_as	日本語をサポートする非決定的照合順序。Supported in Babelfish 2.1.0 and higher releases.
korean_wansung_ci_as	韓国語をサポートする非決定的照合順序 (辞書ソートあり)。
latin1_general_ci_as	ラテン文字をサポートする非決定的照合順序。
modern_spanish_ci_as	現代スペイン語をサポートする非決定的照合順序。
polish_ci_as	ポーランド語をサポートする非決定的照合順序。
thai_ci_as	タイ語をサポートする非決定的照合順序。
traditional_spanish_ci_as	スペイン語 (伝統的なソート) をサポートする非決定的な照合順序。
turkish_ci_as	トルコ語をサポートする非決定的な照合順序。
ukrainian_ci_as	ウクライナ語をサポートする非決定的な照合順序。
vietamese_ci_as	ベトナム語をサポートする非決定的照合順序。

次の照合順序をオブジェクト照合として使用できます。

言語	決定的なオプション	非決定的オプション
アラビア語	Arabic_CS_AS	Arabic_CI_AS, Arabic_CI_AI
中国語	Chinese_CS_AS	Chinese_CI_AS、Chinese_CI_AI
Cyrillic_General	Cyrillic_General_CS_AS	Cyrillic_General_CI_AS、Cyrillic_General_CI_AI
エストニア語	Estonian_CS_AS	Estonian_CI_AS, Estonian_CI_AI
Finnish_Swedish	Finnish_Swedish_CS_AS	Finnish_Swedish_CI_AS、Finnish_Swedish_CI_AI
フランス語	French_CS_AS	French_CI_AS, French_CI_AI
ギリシャ語	Greek_CS_AS	Greek_CI_AS、Greek_CI_AI
ヘブライ語	Hebrew_CS_AS	Hebrew_CI_AS, Hebrew_CI_AI
日本語 (Babelfish 2.1.0 以上)	Japanese_CS_AS	Japanese_CI_AI、Japanese_CI_AS
Korean_Wamsung	Korean_Wamsung_CS_AS	Korean_Wamsung_CI_AS、Korean_Wamsung_CI_AI
Modern_Spanish	modern_Spanish_CS_AS	Modern_Spanish_CI_AS、Modern_Spanish_CI_AI
モンゴル語	Mongolian_CS_AS	Mongolian_CI_AS, Mongolian_CI_AI

言語	決定的なオプション	非決定的オプション
ポーランド語	Polish_CS_AS	Polish_CI_AS, Polish_CI_AI
タイ語	Thai_CS_AS	Thai_CI_AS、Thai_CI_AI
Tradition al_Spanish	Traditional_Spanish_CS_AS	Traditional_Spanish_CI_AS、T raditional_Spanish_CI_AI
トルコ語	Turkish_CS_AS	Turkish_CI_AS、Turkish_CI_AI
ウクライナ語	Ukranian_CS_AS	Ukranian_CI_AS, Ukranian_CI_AI
ベトナム語	Vietnamese_CS_AS	Vietnamese_CI_AS、Vietnamese _CI_AI

Babelfish でのデフォルト照合

以前は、照合可能なデータ型のデフォルトの照合は pg_catalog.default でした。これらのデータ型に依存するデータ型とオブジェクトは、大文字と小文字を区別して照合されます。この条件は、大文字と小文字を区別しない照合でデータセットの T-SQL オブジェクトに影響を与える可能性があります。Babelfish 2.3.0 以降、照合可能なデータ型 (TEXT と NTEXT を除く) のデフォルトの照合は、babelfishpg_tsql.server_collation_name パラメータの照合順序と同じです。Babelfish 2.3.0 にアップグレードすると、DB クラスターの作成時にデフォルトの照合が自動的に選択されるため、目に見える影響はありません。

照合順序の管理

ICU ライブラリは照合バージョン追跡機能を提供し、新しいバージョンの ICU が使用可能になった際、照合順序に依存するインデックスを再インデックスできるようにします。現在のデータベースに更新が必要な照合順序があるかどうかを確認するには、psql または pgAdmin を使用して接続した後に次のクエリを使用します。

```
SELECT pg_describe_object(refclassid, refobjid,
    refobjsubid) AS "Collation",
```

```
    pg_describe_object(classid, objid, objsubid) AS "Object"
    FROM pg_depend d JOIN pg_collation c ON refclassid = 'pg_collation'::regclass
    AND refobjid = c.oid WHERE c.collversion <> pg_collation_actual_version(c.oid)
    ORDER BY 1, 2;
```

このクエリは、次のような出力を返します。

```
 Collation | Object
-----------+--------
(0 rows)
```

この例では、照合順序を更新する必要はありません。

Babelfish データベース内の事前定義された照合順序のリストを取得するには、次のクエリで psql または pgAdmin を使用できます。

```
SELECT * FROM pg_collation;
```

定義済みの照合順序は、sys.fn_helpcollations テーブルに保存されます。次のコマンドを使用して、照合に関する情報 (lcid、スタイル、照合フラグなど) を表示できます。sqlcmd を使用してすべての照合のリストを取得するには、T-SQL ポート (デフォルトでは 1433) に接続し、次のクエリを実行します。

```
1> :setvar SQLCMDMAXVARTYPEWIDTH 40
2> :setvar SQLCMDMAXFIXEDTYPEWIDTH 40
3> SELECT * FROM fn_helpcollations()
4> GO
name                                    description
--------------------------------------- ----------------------------------------
arabic_cs_as                            Arabic, case-sensitive, accent-sensitive
arabic_ci_ai                            Arabic, case-insensitive, accent-insensi
arabic_ci_as                            Arabic, case-insensitive, accent-sensiti
bbf_unicode_bin2                        Unicode-General, case-sensitive, accent-
bbf_unicode_cp1250_ci_ai                Default locale, code page 1250, case-ins
bbf_unicode_cp1250_ci_as                Default locale, code page 1250, case-ins
bbf_unicode_cp1250_cs_ai                Default locale, code page 1250, case-sen
bbf_unicode_cp1250_cs_as                Default locale, code page 1250, case-sen
bbf_unicode_pref_cp1250_cs_as           Default locale, code page 1250, case-sen
bbf_unicode_cp1251_ci_ai                Default locale, code page 1251, case-ins
bbf_unicode_cp1251_ci_as                Default locale, code page 1251, case-ins
bbf_unicode_cp1254_ci_ai                Default locale, code page 1254, case-ins
```

```
...
(124 rows affected)
```

例に示す 1 行目と 2 行目では、文書の読みやすさのためだけに出力を絞り込みます。

```
1> SELECT SERVERPROPERTY('COLLATION')
2> GO
serverproperty
--------------------------------------------------------------------------------
sql_latin1_general_cp1_ci_as

(1 rows affected)
1>
```

照合順序の制限と動作の違い

Babelfish は ICU ライブラリを使用して照合をサポートします。PostgreSQL は特定のバージョンの ICU で構築されており、照合の 1 つのバージョンにだけ一致させることができます。バージョン間のバリエーションは避けられません。言語は時間と共に進化するため、マイナーなバリエーションも同様です。次のセクションでは、Babelfish 照合順序の既知制限と動作のバリエーションをいくつか示しています。

- インデックスおよび照合順序タイプ依存関係 – International Components for Unicode (ICU) 照合ライブラリ (Babelfish が使用するライブラリ) に依存するユーザー定義型のインデックスは、ライブラリのバージョンを変更しても無効になりません。
- 照合関数 – 照合プロパティは、サポートされている Babelfish BBF 照合に対してのみ実装されます。詳細については、「Babelfish supported collations table」を参照してください。
- Unicode ソートルールの違い — SQL Server の SQL 照合順序は、Unicode エンコードされたデータ (ncharそしてnvarchar) を、Unicode でエンコードされていないデータ (charそしてvarchar) とは異なるやり方でソートします。Babelfish データベースは常に UTF-8 でエンコードされ、データ型に関係なく Unicode ソートルールを常に適用します。したがって、char または varchar のソート順は、nchar または nvarchar と同じです。
- 二次等照合順序およびソート動作 - デフォルトの ICU Unicode 二次等 (CI_AS) 照合では、句読点やその他の英数字以外の文字を数字の前にソートし、英字の前に数字をソートします。ただし、句読点やその他の特殊文字の順序は異なります。
- 三次等照合順序、ORDER BY の回避策 - SQL_Latin1_General_Pref_CP1_CI_AS などのSQL照合順序は TERTIARY_WEIGHTS 関数 をサポートし、CI_AS 照合で等しく比較される文字列が初期に大文字でソートされる機能をサポートします: ABC、ABc、AbC、Abc、aBC、aBc、abC、そ

して最後に abc。よって、DENSE_RANK OVER (ORDER BY column) 分析関数は、これらの文字列を同じランクとして評価しますが、パーティション内でまず大文字で並べ替えます。

@colCaseFirst=upper を指定する三次 CS_AS 照合 を明記した ORDER BY 句に COLLATE 句を追加することで、Babelfish でも同様の結果が得られます。ただし、colCaseFirst 修飾子は 3 次と等しい (CI_AS 照合順序のような二次等照合順序ではなく) 文字列にのみ適用されます。したがって、1 つの ICU 照合を使用して 3 次 SQL 照合をエミュレートすることはできません。

回避策として、BBF_SQL_Latin1_General_CP1_CI_AS 照合順序を使う前に SQL_Latin1_General_Pref_CP1_CI_AS 照合を使用するアプリケーションを変更することをお勧めします。その後、このカラムの ORDER BY 句に COLLATE BBF_SQL_Latin1_General_Pref_CP1_CS_AS を追加します。

- 文字拡張 - 文字拡張では、1 文字を 1 次レベルの文字シーケンスと等しく扱います。SQL Server のデフォルト CI_AS 照合では、文字拡張がサポートされています。ICU 照合では、アクセントが区別されない照合順序に対してのみ文字拡張がサポートされます。

  文字拡張が必要な場合は、AI 照合を使って比較してください。ただし、このような照合は LIKE 演算子では現在サポートされていません。

- char および varchar エンコード - SQL で始まる照合を char または varchar データ型に使用する場合、ASCII 127 より前の文字のソート順は、その SQL 照合の特定のコードページによって決まります。SQL 照合では、char または varchar として宣言された文字列は、nchar または nvarchar として宣言された文字列とは異なるソートされることがあります。

  PostgreSQL はすべての文字列をデータベースエンコーディングでエンコードするため、すべての文字を UTF-8 に変換し、Unicode ルールを使用してソートします。

  SQL 照合では Unicode ルールを使用して nchar データ型と nvarchar データ型がソートされるため、Babelfish はサーバー上のすべての文字列を UTF-8 を使用してエンコードします。Babelfish は、Unicode ルールを使用して char 文字列と varchar 文字列をソートするのと同じ方法で、nchar 文字列と nvarchar 文字列をソートします。

- 補足文字 - SQL Server 関数 NCHAR、UNICODE、および LEN は Unicode 基本多言語平面 (BMP) 外のコードポイントの文字をサポートします。対照的に、SC 以外の照合では、サロゲートペア文字を使用して補足文字を処理します。Unicode データ型の場合、SQL Server は UCS-2 を使用して最大 65,535 文字を表すことができます。補足文字を使用する場合は Unicode の全範囲 (1,114,114 文字) を表すことができます。

- かなを区別する (KS) 照合順序 — かなを区別する (KS) 照合とは、Hiragana と Katakana 日本語のかな文字を異なるやり方で処理する照合です。ICU は日本語の照合スタンダードJIS X 4061

をサポートしています。現在非推奨される `colhiraganaQ [on | off]` ロケール修飾子は KS 照合と同じ機能を提供する場合があります。ただし、SQL Server と同じ名前の KS 照合は現在 Babelfish でサポートされていません。

- 幅を区別する (WS) 照合順序 - 同じ文字が 半角文字 (半角) と 全角文字 (全角) で別扱いされる場合、照合は幅を区別する (WS) と呼ばれます。SQL Server と同じ名前の WS 照合は、現在 Babelfish でサポートされていません。
- バリエーションセレクタを区別する (VSS) 照合 — バリエーションセレクタを区別する (VSS) 照合では、日本語の照合順序 `Japanese_Bushu_Kakusu_140` と `Japanese_XJIS_140` における表意文字バリエーションセレクタを区別します。バリエーションシーケンスは、基本文字と追加のバリエーションセレクタで構成されます。`_VSS` オプションを選択しない場合、バリエーションセレクタは比較には考慮されません。

  VSS 照合は、現在 Babelfish はサポートされていません。
- BIN および BIN2 照合 – BIN2 照合では、コードポイントの順序に従って文字がソートされます。UTF-8 のバイト単位のバイナリ順序は Unicode コードポイントの順序を保持するため、これが最もパフォーマンスの高い照合になる可能性もあります。Unicode コードポイントの順序がアプリケーションで機能する場合は、BIN2 照合の使用を検討してください。ただし、BIN2 照合を使用すると、文化的に予期しない順序でデータがクライアントに表示されることがあります。小文字への新しいマッピングは時間の経過とともに Unicode に追加されるため、ICU のバージョンによって、`LOWER` 関数の動作が異なる場合があります。これは BIN2 照合に固有なケースではなく、より一般的な照合バージョニング管理の特殊な不具合ケースです。

  Babelfish は Unicode コードポイント順に照合すため、 Babelfish ディストリビューションで `BBF_Latin1_General_BIN2` 照合を提供します。BIN 照合では、初期の文字だけが wchar としてソートされます。残りの文字はバイト単位でソートされ、エンコーディングに従ってコードポイント順に効率的にソートされます。このアプローチは Unicode 照合ルールに従わず、Babelfish ではサポートされていません。
- 非決定的照合順序と CHARINDEX の制限 — バージョン 2.1.0 より古いバージョンの Babelfish では、非決定的照合で CHARINDEX を使用することはできません。デフォルトでは、Babelfish は大文字と小文字を区別しない (非決定的) 照合を使用します。古いバージョンの Babelfish で CHARINDEX を使用すると、次のランタイムエラーが発生します。

```
nondeterministic collations are not supported for substring searches
```

> **Note**
>
> この制限と回避策は、Babelfish バージョン 1.x (Aurora PostgreSQL 13.x バージョン) にのみ適用されます。Babelfish 2.1.0 以降のリリースでは、この問題は発生しません。

この問題は、次のいずれかの方法で回避できます。

- 式を大文字と小文字を区別する照合に明示的に変換し、LOWER または UPPER を適用して両方の引数で大文字と小文字を区別します。例えば、`SELECT charindex('x', a) FROM t1` は次のようになります。

```
SELECT charindex(LOWER('x'), LOWER(a COLLATE sql_latin1_general_cp1_cs_as)) FROM t1
```

- SQL 関数 f_charindex を作成し、CHARINDEX 呼び出しを次の関数の呼び出しに置き換えます。

```
CREATE function f_charindex(@s1 varchar(max), @s2 varchar(max)) RETURNS int
AS
BEGIN
declare @i int = 1
WHILE len(@s2) >= len(@s1)
BEGIN
  if LOWER(@s1) = LOWER(substring(@s2,1,len(@s1))) return @i
  set @i += 1
  set @s2 = substring(@s2,2,999999999)
END
return 0
END
go
```

エスケープハッチ処理時の Babelfish のエラー処理の管理

Babelfish は可能な限り、制御フローとトランザクションの状態から SQL の動作を模倣します。Babelfish でエラーが発生するとSQL Server エラーコードに似たエラーコードを返します。Babelfish がエラーを SQL Server のコードにマッピングできない場合、修正されたエラーコード (33557097) が返され、エラーの種類に基づいて特定のアクションを実行します。

- コンパイル時エラーの場合、Babelfish はトランザクションをロールバックします。
- ランタイム時エラーの場合、Babelfish はバッチを終了し、トランザクションをロールバックします。
- クライアントとサーバー間のプロトコルエラーの場合、トランザクションはロールバックされません。

エラーコードを同等のコードにマッピングできず、同様のエラーのコードが使用可能な場合、エラーコードは代替コードにマッピングされます。例えば、SQL Server コード 8143 や 8144 の原因となる動作などは、両方とも 8143 にマップされます。

マッピングできないエラーは、`TRY... CATCH` 構成を配慮しません。

`@@ERROR` を使って SQL Server のエラーコードを返すか、`@@PGERROR` 関数 を使って PostgreSQL エラーコードを返すことができます。また `fn_mapped_system_error_list` 関数を使用して、マッピングされたエラーコードのリストを返しますことも可能です。PostgreSQL のエラーコードについては、「PostgreSQL ウェブサイト」を参照してください。

Babelfish エスケープのハッチ設定を変更する

失敗する可能性のあるステートメントを処理するために、Babelfish はエスケープハッチと呼ばれる特定のオプションを定義しています。エスケープハッチングは、サポートされていない特徴や構文に遭遇した際に、 Babelfish の動作を指定するオプションです。

`sp_babelfish_configure` ストアドプロシージャを使用して、エスケープハッチングの設定を制御することができます。スクリプトを使用して、エスケープハッチングを `ignore` または `strict` に設定します。`strict` に設定されている場合、Babelfish は修正する必要のあるエラーを続行する前に返します。

現在のセッションおよびクラスターレベルに変更を適用するには、`server` キーワードを含めます。

使用方法は次のようになります:

- エスケープハッチングとそのステータス、および使用状況をすべて一覧表示するには、sp_babelfish_configure を実行します。
- 現在のセッションまたはクラスター全体の名前付きエスケープハッチングとその値を一覧表示するには、*hatch_name* が 1 つ以上のエスケープハッチングの識別子になっている sp_babelfish_configure '*hatch_name*' コマンドを実行します。*hatch_name* は「%」などの SQL ワイルドカードを使用できます。
- 1 つまたは複数のエスケープハッチングを指定した値に設定するには、sp_babelfish_configure ['*hatch_name*' [, 'strict'|'ignore' [, 'server']] を実行します。クラスター全体のレベルで設定を永続化するには、次に示されているように、server キーワードを含めます。

```
EXECUTE sp_babelfish_configure 'escape_hatch_unique_constraint', 'ignore', 'server'
```

  現在のセッションに対してのみ設定する場合、server は使用しないでください。
- すべてのエスケープハッチをデフォルト値にリセットするには、sp_babelfish_configure 'default' を実行します (Babelfish 1.2.0 以上)。

ハッチ (または複数のハッチ) を識別する文字列に SQL ワイルドカードが含まれている場合があります。例えば以下では、すべての構文エスケープハッチングを Aurora PostgreSQL クラスターの ignore に設定しています。

```
EXECUTE sp_babelfish_configure '%', 'ignore', 'server'
```

次の表に、Babelfish の定義済みエスケープハッチの説明とデフォルト値を示します。

エスケープハッチ	説明	デフォルト値
escape_hatch_checkpoint	手続き型コードで CHECKPOINT ステートメントを使用できますが、CHECKPOINT ステートメントは現在実装されていません。	ignore
escape_hatch_constraint_name_for_default	デフォルトの制約名に関連する Babelfish の動作を制御します。	ignore

エスケープハッチ	説明	デフォルト値
escape_hatch_database_misc_options	CREATE ないし ALTER DATABASE のオプションに関連した以下の Babelfish 動作を管理します: CONTAINMENT、DB_CHAINING、TRUSTWORTHY、 PERSISTENT_LOG_BUFFER。	ignore
escape_hatch_for_replication	テーブルを作成または変更する際、 [NOT] FOR REPLICATION 句に関連する Babelfish の動作を制御します。	strict
escape_hatch_fulltext	CREATE/ALTER DATABASE の DEFAULT_FULLTEXT_LANGUAGE や CREATE FULLTEXT INDEX、 また sp_fulltext_database などの FULLTEXT 機能に関連する Babelfish の動作を制御します。	ignore
escape_hatch_ignore_dup_key	CREATE/ALTER TABLE および CREATE INDEX に関する Babelfish の動作を制御します。IGNORE_DUP_KEY=ON の場合、strict (デフォルト) に設定するとエラーが発生し、ignore に設定するとエラーを無視します (Babelfish バージョン 1.2.0 以上)。	strict

エスケープハッチ	説明	デフォルト値
escape_hatch_index_clustering	インデックス、プライマリキー、または UNIQUE 制約のクラスタ化キーワードまたは非クラスタ化キーワードに関連する Babelfish の動作を制御します。CLUSTERED が無視された場合も、インデックスまたは制約は NONCLUSTERED が指定されているかのように作成されます。	ignore
escape_hatch_index_columnstore	COLUMNSTORE 句に関連する Babelfish の動作を制御します。ignore を指定した場合、Babelfish は通常の B-tree インデックスを作成します。	strict
escape_hatch_join_hints	JOIN 演算子のキーワードの動作を制御します: LOOP、HASH、MERGE、REMOTE、REDUCE、REDISTRIBUTE、REPLICATE。	ignore
escape_hatch_language_non_english	画面上のメッセージの英語以外の言語に関する Babelfish の動作を制御します。Babelfish は現在 画面上のメッセージのみで us_english をサポートされます。SET LANGUAGE は言語名を含む可変を使用する場合があるため、設定されている実際の言語はランタイムでのみ検出できます。	strict

エスケープハッチ	説明	デフォルト値
escape_hatch_login_hashed_password	無視された場合は、CREATE LOGIN とALTER LOGIN の HASHED キーワードに対するエラーが抑制されます。	strict
escape_hatch_login_misc_options	無視された場合は、HASHED、MUST_CHANGE 、OLD_PASSWORD 、および UNLOCK 以外の他のキーワード、そして CREATE LOGIN と ALTER LOGIN に対するエラーが抑制されます。	strict
escape_hatch_login_old_password	無視された場合は、CREATE LOGIN とALTER LOGIN の OLD_PASSWORD キーワードに対するエラーが抑制されます。	strict
escape_hatch_login_password_must_change	無視された場合は、CREATE LOGIN とALTER LOGIN の MUST_CHANGE キーワードに対するエラーが抑制されます。	strict
escape_hatch_login_password_unlock	無視された場合は、CREATE LOGIN とALTER LOGIN の UNLOCK キーワードに対するエラーが抑制されます。	strict

エスケープハッチ	説明	デフォルト値
escape_hatch_nocheck_add_constraint	制約の WITH CHECK 句または NOCHECK 句に関連する Babelfish の動作を制御します。	strict
escape_hatch_nocheck_existing_constraint	FOREIGN KEY および CHECK 制約に関連する Babelfish の動作を制御します。	strict
escape_hatch_query_hints	クエリヒントに関連する Babelfish の動作を制御します。このオプションが 無視するよう設されていると、サーバーは OPTION (...) 句を使用してクエリ処理の側面を指定するヒントを無視します。例に含まれるのは SELECT FROM… OPTION(MERGE JOIN HASH, MAXRECURSION 10))	ignore
escape_hatch_rowversion	ROWVERSION および TIMESTAMP のデータ型の動作を制御します。使用に関する情報については、「実装が制限されている機能」を参照してください。	strict
escape_hatch_schemabinding_function	WITH SCHEMABINDING 句に関連する Babelfish の動作を制御します。デフォルトでは、CREATE または ALTER FUNCTION コマンドで指定した場合は、WITH SCHEMABINDING 句は無視されます。	ignore

エスケープハッチ	説明	デフォルト値
escape_hatch_schemabinding_procedure	WITH SCHEMABINDING 句に関連する Babelfish の動作を制御します。デフォルトでは、CREATE または ALTER PROCEDURE コマンドで指定した場合は、WITH SCHEMABINDING 句は無視されます。	ignore
escape_hatch_rowguidcol_column	テーブルを作成または変更する際、 ROWGUIDCOL 句に関連する Babelfish の動作を制御します。	strict
escape_hatch_schemabinding_trigger	WITH SCHEMABINDING 句に関連する Babelfish の動作を制御します。デフォルトでは、CREATE または ALTER TRIGGER コマンドで指定した場合は、WITH SCHEMABINDING 句は無視されます。	ignore
escape_hatch_schemabinding_view	WITH SCHEMABINDING 句に関連する Babelfish の動作を制御します。デフォルトでは、CREATE または ALTER VIEW コマンドで指定した場合は、WITH SCHEMABINDING 句は無視されます。	ignore

エスケープハッチ	説明	デフォルト値
escape_hatch_session_settings	サポートされていないセッションレベルの SET ステートメントに対する Babelfish の動作を制御します。	ignore
escape_hatch_showplan_all	SET SHOWPLAN_ALL と SET STATISTICS PROFILE に関連する Babelfish の動作を制御します。ignore に設定すると、SET BABELFISH_SHOWPLAN_ALL と SET BABELFISH_STATISTICS PROFILE のように動作します。strict に設定すると、何も通知せずに無視されます。	strict
escape_hatch_storage_on_partition	パーティショニングの定義時に ON partition_scheme column 句に関連した Babelfish の動作を制御します。Babelfish は現在、パーティショニングを実装していません。	strict

エスケープハッチ	説明	デフォルト値
escape_hatch_storage_options	CREATE、ALTER DATABASE、TABLE、INDEXで使用されるストレージオプションのハッチングをエスケープします。これには、テーブル、インデックス、制約、およびデータベースの格納場所 (パーティション、ファイルグループ) を定義する句 (LOG) ON、TEXTIMAGE_ON、FILESTREAM_ON が含まれます。このエスケープハッチの設定は、これらすべての句 (ON [PRIMARY] および ON [DEFAULT] を含む) に適用されます。例外は、ON partition_scheme (カラム) を持つテーブルまたはインデックスにパーティションが指定されている場合です。	ignore
escape_hatch_table_hints	WITH (...) 句を使って指定されたテーブルヒントの動作を制御します。	ignore

エスケープハッチ	説明	デフォルト値
escape_hatch_unique_constraint	strict に設定すると、インデックス付き列の NULL 値の処理における SQL Server と PostgreSQL のあいまいな意味の違いにより、エラーが発生することがあります。意味の違いは非現実的なユースケースでのみ発生するため、このエスケープハッチを「無視」に設定して、エラーが表示されないようにすることができます。	strict

Babelfish for Aurora PostgreSQL DB クラスターの作成

Babelfish for Aurora PostgreSQL のこのリリースは Aurora PostgreSQL バージョン 13.4 以降のリリースでサポートされています。

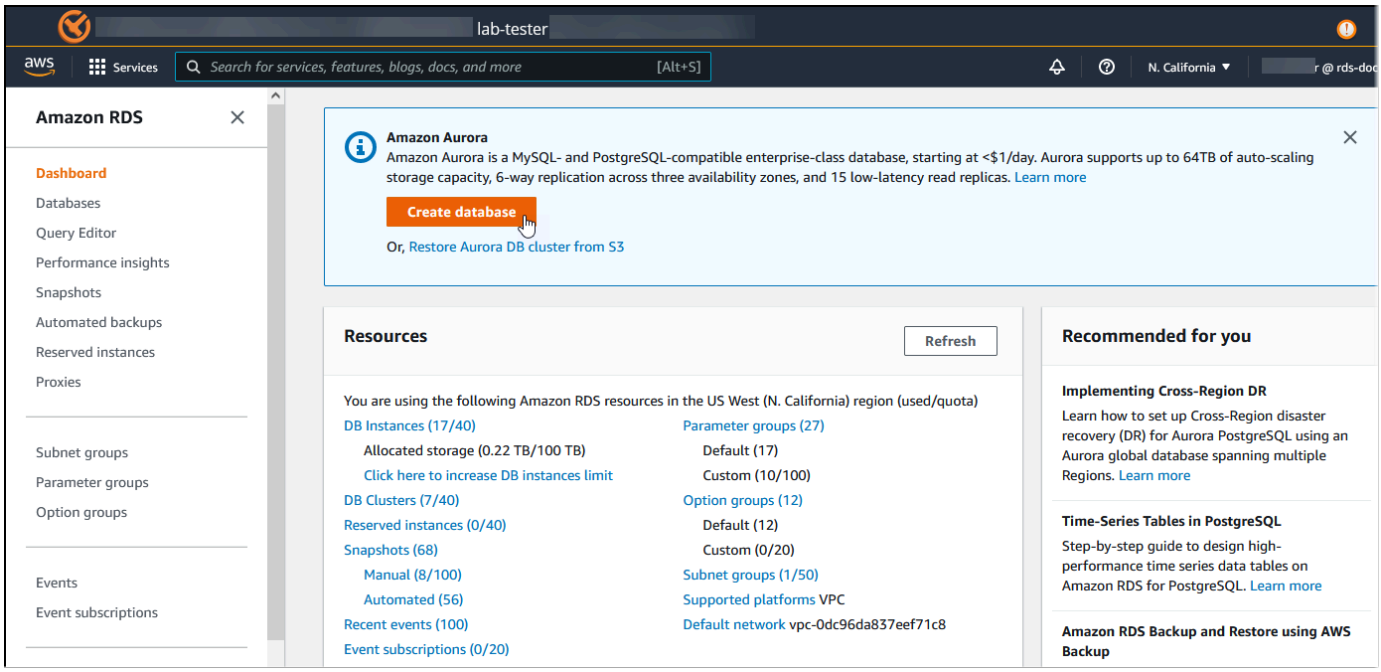
AWS Management Console または AWS CLI を使用して、Babelfish で Aurora PstgreSQL クラスターを作成できます。

> **Note**
>
> Aurora PostgreSQL クラスターでは、`babelfish_db` データベース名は Babelfish 用に予約されています。Aurora PostgreSQL の Babelfish に独自の「babelfish_db」データベースを作成すると、Aurora は正常に Babelfish のプロビジョニングができなくなります。

コンソール

AWS Management Console で実行している Babelfish を使ってクラスターを作成する方法

1. https://console.aws.amazon.com/rds/ で Amazon RDS コンソールを開き、データベースを作成を選択します。

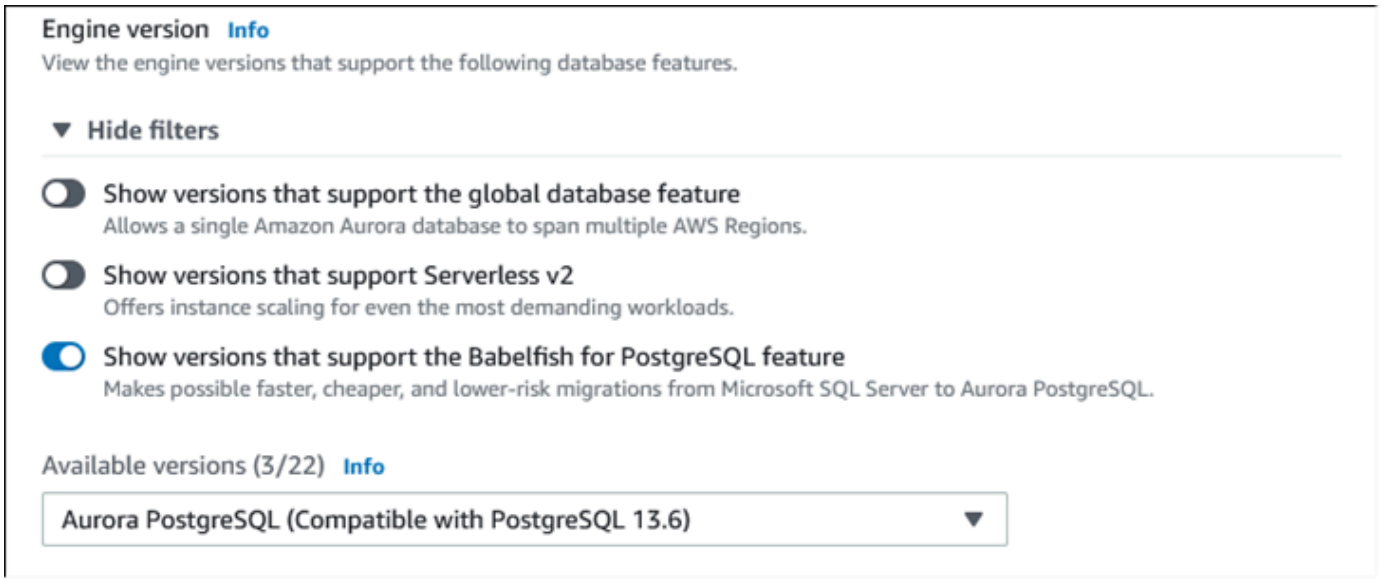


2. データベース作成方法を選択するには、次のいずれかを実行します:

   - 詳細なエンジンオプションを指定するには、スタンダード作成を選択します。

- Aurora クラスターのベストプラクティスをサポートする事前設定済みのオプションを使用するには、Easy Create (簡単作成)を選択します

3. [エンジンタイプには] として、[Aurora (PostgreSQL 互換)] を選択します。
4. [フィルターを表示] を選択した後 [Babelfish for PostgreSQL 特徴をサポートしているバージョンを表示する] を選択し、Babelfish をサポートするエンジンの種類をリストアップします。Babelfish は現在 Aurora PostgreSQL バージョン 13.4 以降でサポートされています。
5. [使用できるバージョン] で、Aurora PostgreSQL のバージョンを選択します。Babelfish の最新機能を入手するには、Aurora PostgreSQL の最高メジャーバージョンを選択してください。

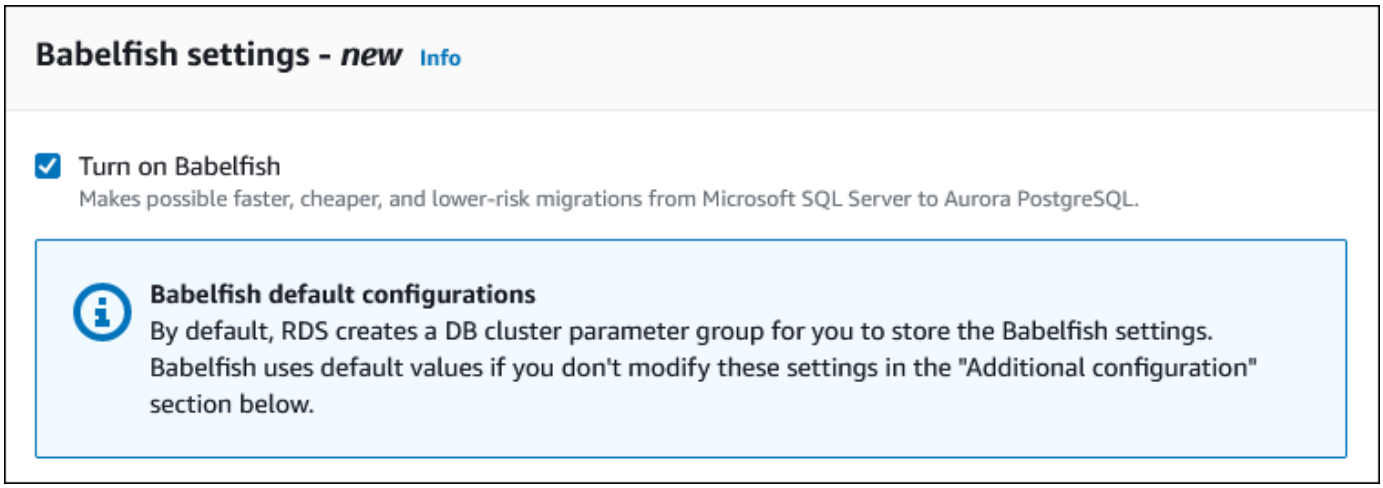


6. テンプレート で、ユースケースに合うテンプレートを選択します。
7. DB cluster identifier (DB クラスター識別子)で、後ほど DB クラスターリストで簡単に見つけられる名前を入力します。
8. [マスターユーザーネーム] で、管理者ユーザーネームを入力します。Aurora PostgreSQL のデフォルト値は `postgres` です。デフォルトを使用するか、別の名前を選択します。例えば、SQL Server データベースで使用される命名規則に従うには、マスターユーザー名に `sa` (システム管理者) と入力します。

   この時点で `sa` という名前のユーザーを作成しない場合、選択したクライアントを使用して後ほど作成できます。ユーザーを作成したら、`ALTER SERVER ROLE` コマンドを使用してクラスターの `sysadmin` グループ (ロール) に追加します。

> **Warning**
>
> マスターユーザー名は、常に小文字を使用する必要があり、そうしないと、DB クラスターは TDS ポート経由で Babelfish に接続できません。

9. [マスターパスワード] には、強力なパスワードを作成し、パスワードを確認します。
10. [Bablfish の設定] セクションまで続くオプションについては、DB クラスター設定を指定します。各設定の詳細については、「Aurora DB クラスターの設定」を参照してください。
11. Babelfish 機能を利用可能にするには、Babelfish を有効にするボックスを選択します。



12. DB クラスターのパラメータグループで、次のいずれかを実行します:

    - 新規作成を選択し、Babelfish が有効な新しいパラメータグループを作成します。
    - 既存のものを選択を選択して、既存のパラメータグループを使用します。既存のグループを使用する場合は、クラスターを作成する前にグループを変更し、Babelfish パラメータの値を追加してください。Babelfish パラメータの詳細については、Babelfish の DB クラスターパラメータグループ設定 を参照してください。

    既存のグループを使用する場合は、次のボックスにグループ名を入力します。
13. データベース移行モードの場合は、次のいずれかを選択します。

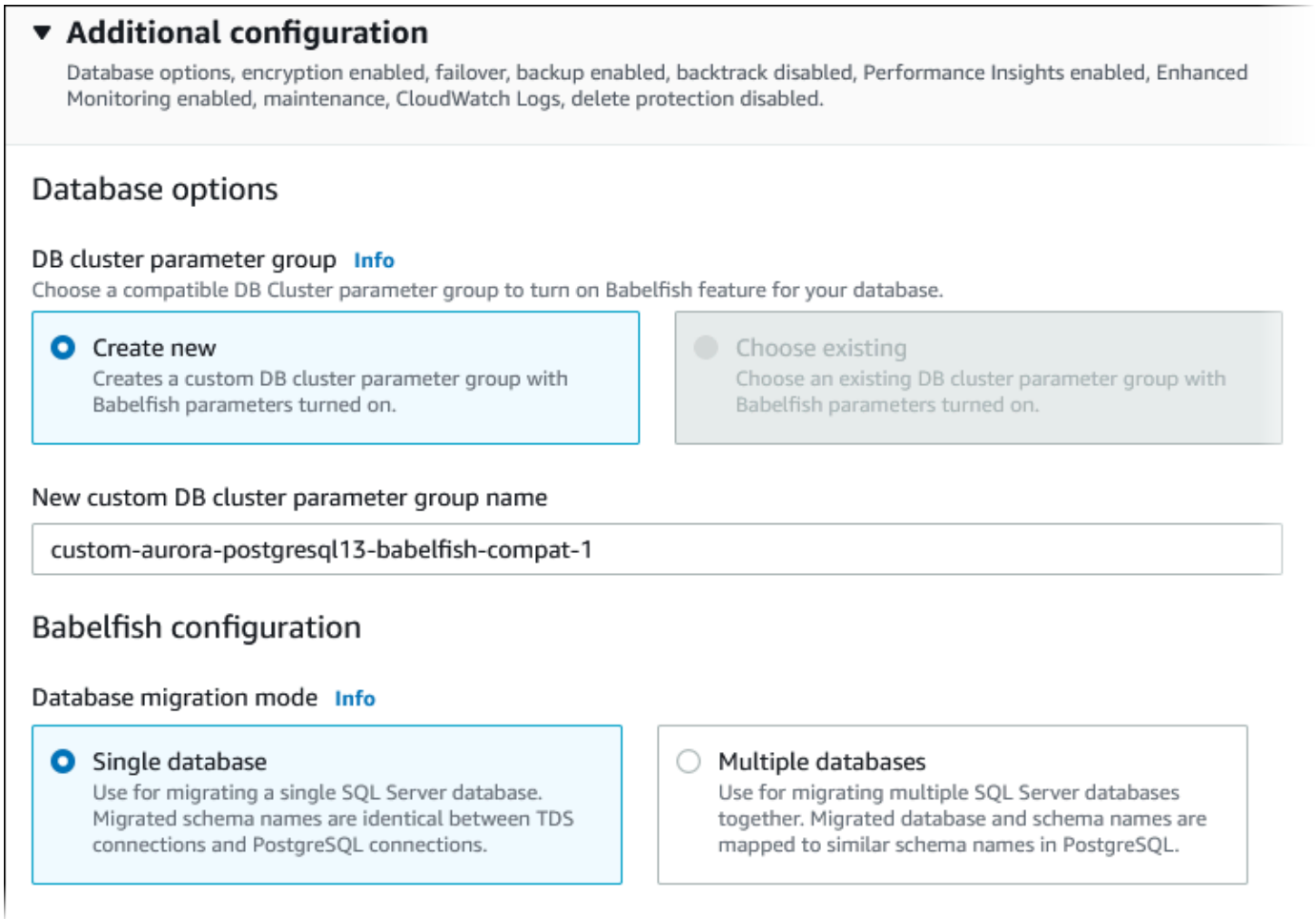
    - 単一データベースを選択して、単一の SQL Server データベースを移行する。

    Aurora PostgreSQL への完全な移行が最終目標の場合などには、Babelfish を使用せずにネイティブ Aurora PostgreSQL へ複数のユーザーデータベースを一緒に移行することもできます。最終的なアプリケーションで統合スキーマ (単一の dbo スキーマ) が必要な場合は、ま

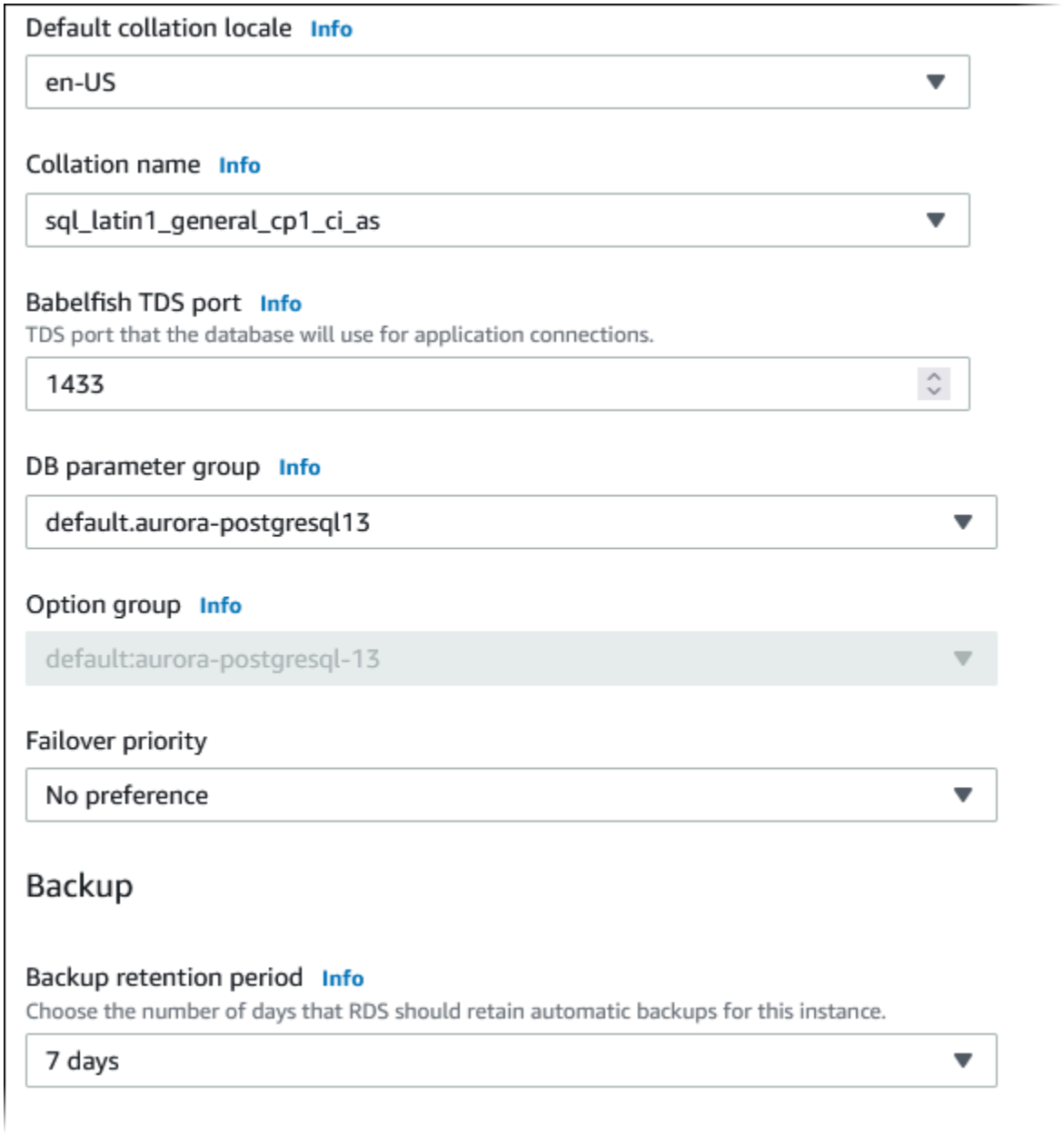
ず SQL Server データベースを単一の SQL Server データベースに統合してください。その後、単一データベースモードを使用して Babelfish に移行します。

- マルチデータベースによる、複数の SQL Server データベース (単一の SQL Server インストールを発生元とする) の移行。マルチデータベースモードでは、単一の SQL Server インストールを発生元としないマルチデータベースは統合されません。マルチデータベースの移行については、1 つのデータベースまたは複数のデータベースでの babelfish の使用 を参照してください。

> **Note**
>
> Aurora PostgreSQL 16 バージョンから、デフォルトでデータベース移行モードとして複数のデータベースが選択されています。



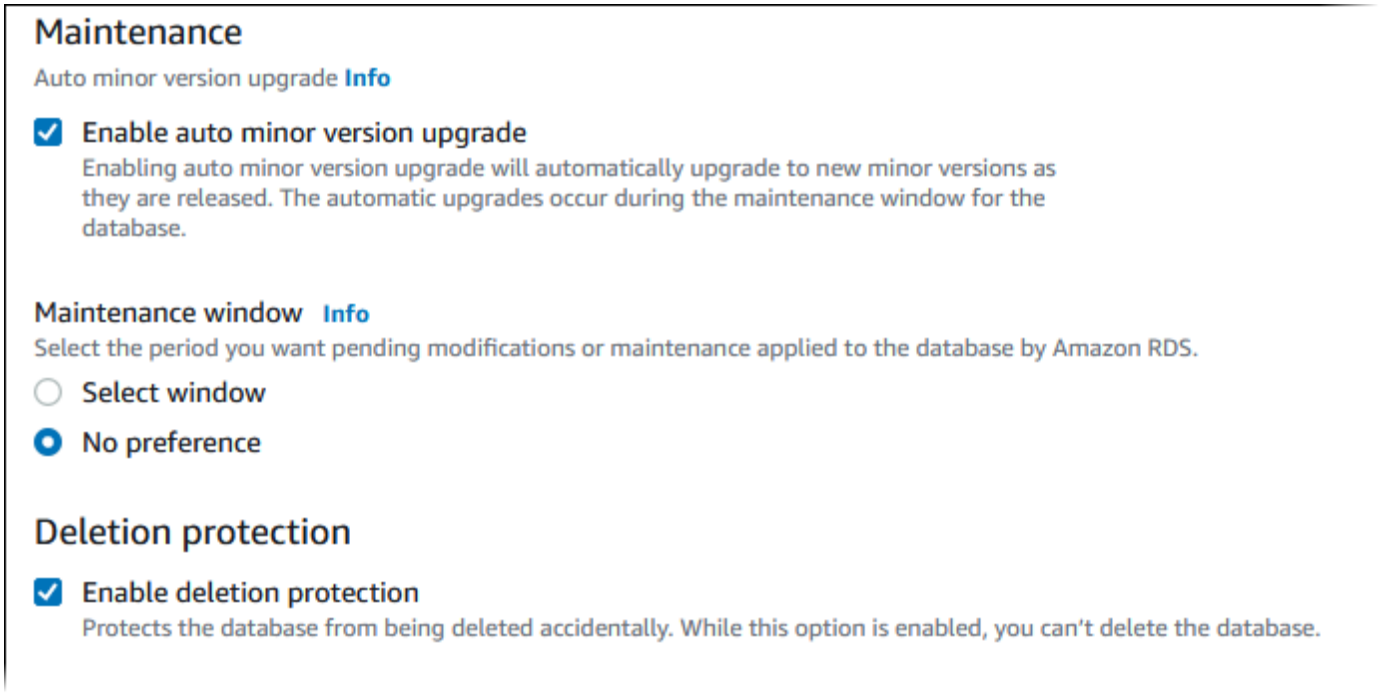
14. デフォルトの照合ロケール で、サーバーロケールを入力します。デフォルトは en-US です。照合の詳細については、Babelfish がサポートする照合順序 を参照してください。
15. 照合順名フィールドは、デフォルトの照合順序を入力します。デフォルトは sql_latin1_general_cp1_ci_as です。詳細については、「Babelfish がサポートする照合順序」を参照してください。
16. [Babelfish TDS ポート] に、デフォルトポート 1433 を入力します。現在、Babelfish は、DB クラスターについてはポート 1433 のみをサポートしています。
17. [DB パラメータグループ] で、パラメータグループを選択するか、Aurora にデフォルト設定で新しいグループを作成させます。
18. フェイルオーバー優先順位で、インスタンスのフェイルオーバー優先度を選択します。値を選択しない場合、デフォルト値は tier-1 になります。この優先度により、プライマリインスタンスの障害からの復旧時に、 レプリカを昇格する順序が決まります。詳しくは、「Aurora DB クラスターの耐障害性」を参照してください。
19. Backup retention period (バックアップ保持期間) で、Aurora がデータベースのバックアップコピーを保持する期間 (1 ~ 35 日) を選択します。バックアップコピーは、2 番目のデータベースに対するポイントインタイム復元 (PITR) で使用できます。デフォルトの保持期間は 7 日間です。



20. スナップショットにタグをコピーを選択し、スナップショット作成時に DB インスタンスタグを DB スナップショットにコピーします。
21. Enable encryption (暗号化の有効)をクリックして、この DB クラスター保存時の暗号化 (Aurora ストレージ暗号化) を有効にします。
22. Performance Insights の有効をクリックして Amazon RDS Performance Insights を有効にします。
23. DB クラスターが実行されているオペレーティングシステムに対してリアルタイムでのメトリクスの収集をスタートするには、Enable enhanced monitoring (高度なモニタリングの有効) を選択します。

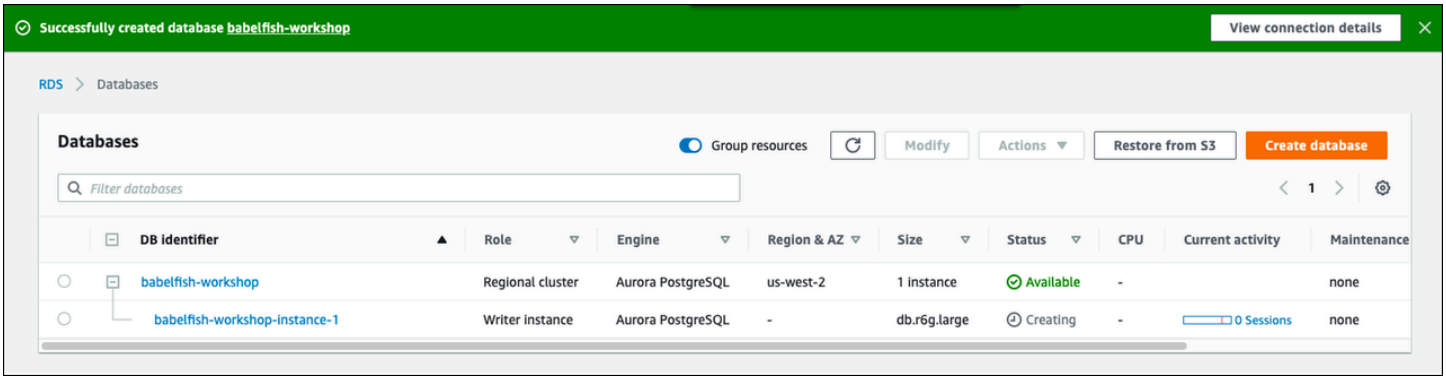
24. PostgreSQL ログを選択して、Amazon CloudWatch Logs にログファイルを発行します。
25. マイナーバージョン自動アップグレードの有効化を選択して、マイナーバージョンのアップグレードが利用可能な際に Aurora DB クラスターを自動的に更新します。
26. Maintenance window (メンテナンスウィンドウ) で、次の作業を行います。
    - Amazon RDS が変更やメンテナンスを実行する時間を設定するには、ウィンドウを選択を選びます。
    - 予定外の時刻に Amazon RDS メンテナンスを実行するには、指定なしを選択します。
27. 削除保護の有効化ボックスを選択して、データベースが誤って削除されるのを防ぎます。

    この特徴を有効にすると、データベースを直接削除することができなくなります。代わりに、データベースを削除する前にデータベースクラスターを変更し、この特徴を無効にする必要があります。



28. [データベースの作成] を選択します。

Babelfish 用にセットアップされた新しいデータベースは、データベースリスティングから確認できます。デプロイが完了した際、Staus (ステータス) 列に Available (使用可能) が表示されます。



AWS CLI

Babelfish for Aurora PostgreSQL を作成する場合、AWS CLI を使用して、クラスターに使用する DB クラスターパラメータグループの名前をコマンドに渡す必要があります。詳しくは、「DB クラスターの前提条件」を参照してください。

AWS CLI を使って Babelfish で Aurora PostgreSQL クラスターを作成するには、以下の操作を行います。

- Amazon Aurora エンドポイントとクォータのサービスのリストから、エンドポイント URL を選択します。
- クラスターのパラメータグループを作成します。パラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。
- パラメータグループを変更し、Babelfish を有効にするパラメータを追加します。

AWS CLI を使用して Babelfish で Aurora PostgreSQL DB クラスターを作成する方法

以下の例では、デフォルトのマスターユーザー名 `postgres` を使用しています。必要に応じて、`sa` または、デフォルトを受け入れなかった場合に選択したユーザー名など、DB クラスター用に作成したユーザー名に置き換えます。

1. パラメータグループを作成します。

   Linux、macOS、Unix の場合:

```
aws rds create-db-cluster-parameter-group \
--endpoint-url endpoint-url \
--db-cluster-parameter-group-name parameter-group \
--db-parameter-group-family aurora-postgresql14 \
--description "description"
```

Windows の場合:

```
aws rds create-db-cluster-parameter-group ^
--endpoint-url endpoint-URL ^
--db-cluster-parameter-group-name parameter-group ^
--db-parameter-group-family aurora-postgresql14 ^
--description "description"
```

2. パラメータグループを変更して Babelfish を有効にします。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster-parameter-group \
--endpoint-url endpoint-url \
--db-cluster-parameter-group-name parameter-group \
--parameters
 "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot"
```

Windows の場合:

```
aws rds modify-db-cluster-parameter-group ^
--endpoint-url endpoint-url ^
--db-cluster-parameter-group-name paramater-group ^
--parameters
 "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot"
```

3. 新しい DB クラスターの DB サブネットグループと Virtual Private Cloud (VPC) セキュリティグループ ID を認証し、create-db-cluster コマンドを呼び出します。

Linux、macOS、Unix の場合:

```
aws rds create-db-cluster \
--db-cluster-identifier  cluster-name\
--master-username postgres \
--manage-master-user-password \
--engine aurora-postgresql \
--engine-version 14.3            \
--vpc-security-group-ids security-group \
--db-subnet-group-name subnet-group-name \
--db-cluster-parameter-group-name parameter-group
```

Windows の場合:

```
aws rds create-db-cluster ^
--db-cluster-identifier cluster-name ^
--master-username postgres ^
--manage-master-user-password ^
--engine aurora-postgresql ^
--engine-version 14.3 ^
--vpc-security-group-ids security-group ^
--db-subnet-group-name subnet-group ^
--db-cluster-parameter-group-name parameter-group
```

この例では、マスターユーザーパスワードを生成して Secrets Manager で管理する `--manage-master-user-password` オプションを指定しています。詳細については、「Amazon Aurora および AWS Secrets Manager によるパスワード管理」を参照してください。または、`--master-password` オプションを使用して、自分でパスワードを指定して管理することもできます。

4. DB クラスターにプライマリインスタンスを明示的に作成します。次のように、create-db-instance コマンドを呼び出すときに `--db-cluster-identifier` 引数に対してステップ 3 で作成したクラスターの名前を使用します。

   Linux、macOS、Unix の場合:

   ```
   aws rds create-db-instance \
   --db-instance-identifier instance-name \
   --db-instance-class db.r6g \
   --db-subnet-group-name subnet-group \
   --db-cluster-identifier cluster-name \
   --engine aurora-postgresql
   ```

   Windows の場合:

   ```
   aws rds create-db-instance ^
   --db-instance-identifier instance-name ^
   --db-instance-class db.r6g ^
   --db-subnet-group-name subnet-group ^
   --db-cluster-identifier cluster-name ^
   --engine aurora-postgresql
   ```

SQL Server データベースを Babelfish for Aurora PostgreSQL に移行する

Aurora 用の Babelfish PostgreSQL を使用すると、SQL Server データベースから Amazon Aurora PostgreSQL DB クラスターへの移行が可能です。移行前に「1 つのデータベースまたは複数のデータベースでの babelfish の使用」を確認します。

トピック

- データ移行プロセスの概要
- SQL Server と Babelfish の違いを評価して処理する
- SQL Server から Babelfish に移行するためのインポート/エクスポートツール

データ移行プロセスの概要

以下の概要では、SQL Server アプリケーションを移行して Babelfish で使用するために必要なステップを説明します。エクスポートおよびインポートプロセスに使用できるツールの詳細については、「SQL Server から Babelfish に移行するためのインポート/エクスポートツール」を参照してください。

1. Babelfish をオンにした状態で、新しい Aurora PostgreSQL DBクラスターを作成します。この方法の詳細は、Babelfish for Aurora PostgreSQL DB クラスターの作成を参照してください。

   SQL Server データベースからエクスポートされたさまざまな SQL アーティファクトをインポートするには、sqlcmd などの SQL Server ツールを使用して Babelfish クラスターに接続します。詳細については、「SQL Server クライアントを使用した DB クラスターへの接続」を参照してください。
2. 移行したい SQL Server データベースで、データ定義言語 (DDL) をエクスポートします。DDL は、ユーザーデータ (テーブル、インデックス、ビューなど) とユーザーが作成したデータベースコード (保存された手順、ユーザー定義の関数、トリガーなど) を含むデータベースオブジェクトを記述する SQL コードです。

   詳細については、「SQL Server Management Studio (SSMS) を使用して Babelfish に移行する」を参照してください。
3. 評価ツールを実行して、Babelfish が SQL Server で実行されているアプリケーションを効果的にサポートできるように、必要な変更の範囲を評価します。詳細については、「SQL Server と Babelfish の違いを評価して処理する」を参照してください。
4. データをロードするには、移行要件に応じて、Babelfish または Aurora PostgreSQL をターゲットエンドポイントとして AWS DMS を使用することをお勧めします。必ず、推奨される Babelfish

データ型で列を変更してください。そのためには、「Babelfish を AWS DMS のターゲットとして使用するための前提条件」を参照してください。

5. 新しい Babelfish DB クラスターで、指定された T-SQL データベース内で DDL を実行して、プライマリキー制約を持つスキーマ、ユーザー定義データ型、およびテーブルのみを作成します。
6. AWS DMS を使用して、SQL Server から Babelfish テーブルにデータを移行します。SQL Server 変更データキャプチャまたは SQL レプリケーションを使用する連続レプリケーションの場合は、エンドポイントとして Babelfish の代わりに Aurora PostgreSQL を使用してください。これを行うには、「AWS Database Migration Service のターゲットとして Babelfish for Aurora PostgreSQL を使用する」を参照してください。
7. データのロードが完了したら、アプリケーションをサポートする残りのすべての T-SQL オブジェクトを Babelfish クラスター上に作成します。
8. SQL Server データベースではなく Babelfish エンドポイントに接続するようにクライアントアプリケーションを再設定します。詳細については、「Babelfish DB クラスターへの接続」を参照してください。
9. 必要に応じてアプリケーションを変更し、再テストします。詳しくは、「Babelfish for Aurora PostgreSQL と SQL Server の違い」を参照してください。

ただし、引き続きクライアント側の SQL クエリを評価する必要があります。SQL Server インスタンスから生成されたスキーマは、サーバー側 SQL コードのみを変換します。次の手順を実行することをお勧めします。

- TSQL_Replay 定義済みテンプレートで SQL Server プロファイラーを使用して、クライアント側のクエリをキャプチャします。このテンプレートは T-SQL ステートメント情報をキャプチャします。この情報は、反復的なチューニングとテストのために再生できます。プロファイラーは、SQL Server Management Studio 内の [Tools] (ツール) メニューから起動できます。[SQL Server Profiler] (SQL Server プロファイラー) をクリックしてプロファイラーを開き、TSQL_Replay テンプレートを選択します。

  Babelfish の移行に使用するには、トレースを開始し、機能テストを使用してアプリケーションを実行します。プロファイラーは T-SQL ステートメントをキャプチャします。テストが完了したら、以下のようにトレースを停止します。クライアント側のクエリを使用して、結果を XML ファイルに保存します ([ファイル] > [名前を付けて保存] > [再生のために XML ファイルをトレース])。

  詳細については、Microsoft ドキュメントの「SQL Server プロファイラー」を参照してください。TSQL_Replay テンプレートの詳細については、「SQL Server プロファイラーテンプレート」を参照してください。

- 複雑なクライアント側 SQL クエリを使用するアプリケーションでは、Babelfish の互換性のために、Babelfish Compass を使用してこれらのクエリを分析することをお勧めします。分析により、クライアント側 SQL 文にサポートされていない SQL 機能が含まれていることが示された場合、クライアントアプリケーションの SQL の側面を確認し、必要に応じて変更を加えます。
- SQL クエリを拡張イベント (.xel 形式) としてキャプチャすることもできます。これを行うには、SSMS XEvent プロファイラーを使用します。.xel ファイルを生成した後、SQL ステートメントを Compass で処理できる .xml ファイルに抽出します。詳細については、Microsoft ドキュメントの「SSMS XEvent プロファイラーの使用」を参照してください。

移行アプリケーションに必要なテスト、分析、および変更がすべて完了したら、本番環境で Babelfish データベースの使用を開始できます。そのためには、元のデータベースを停止し、ライブクライアントアプリケーションをリダイレクトして Babelfish TDS ポートを使用します。

SQL Server と Babelfish の違いを評価して処理する

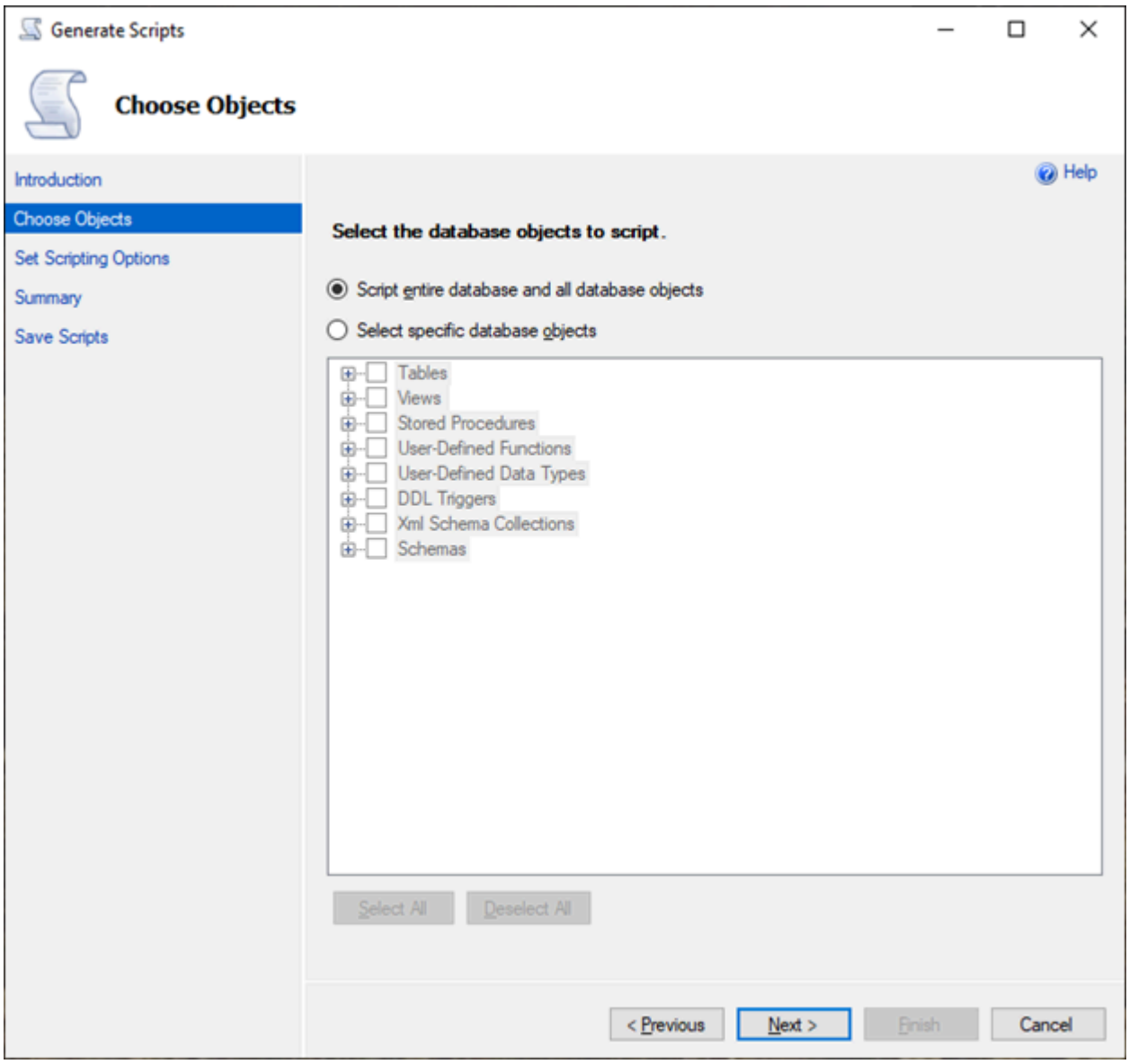
最良の結果を得るには、SQL Server データベースアプリケーションを実際に Babelfish に移行する前に、生成された DDL/DML とクライアントクエリコードを評価することをお勧めします。Babelfish のバージョンと、アプリケーションが実装する SQL Server の特定の機能によっては、アプリケーションをリファクタリングするか、Babelfish でまだ完全にはサポートされていない機能の代替手段を使用する必要がある場合があります。

- SQL Server アプリケーションコードを評価するには、生成された DDL に対して Babelfish Compass を使用し、T-SQL コードが Babelfish でサポートされる範囲を調べます。Babelfish で実行する前に、変更が必要となる可能性がある T-SQL コードを特定します。このツールの詳細については、GitHub で「Babelfish Compass ツール」を参照してください。

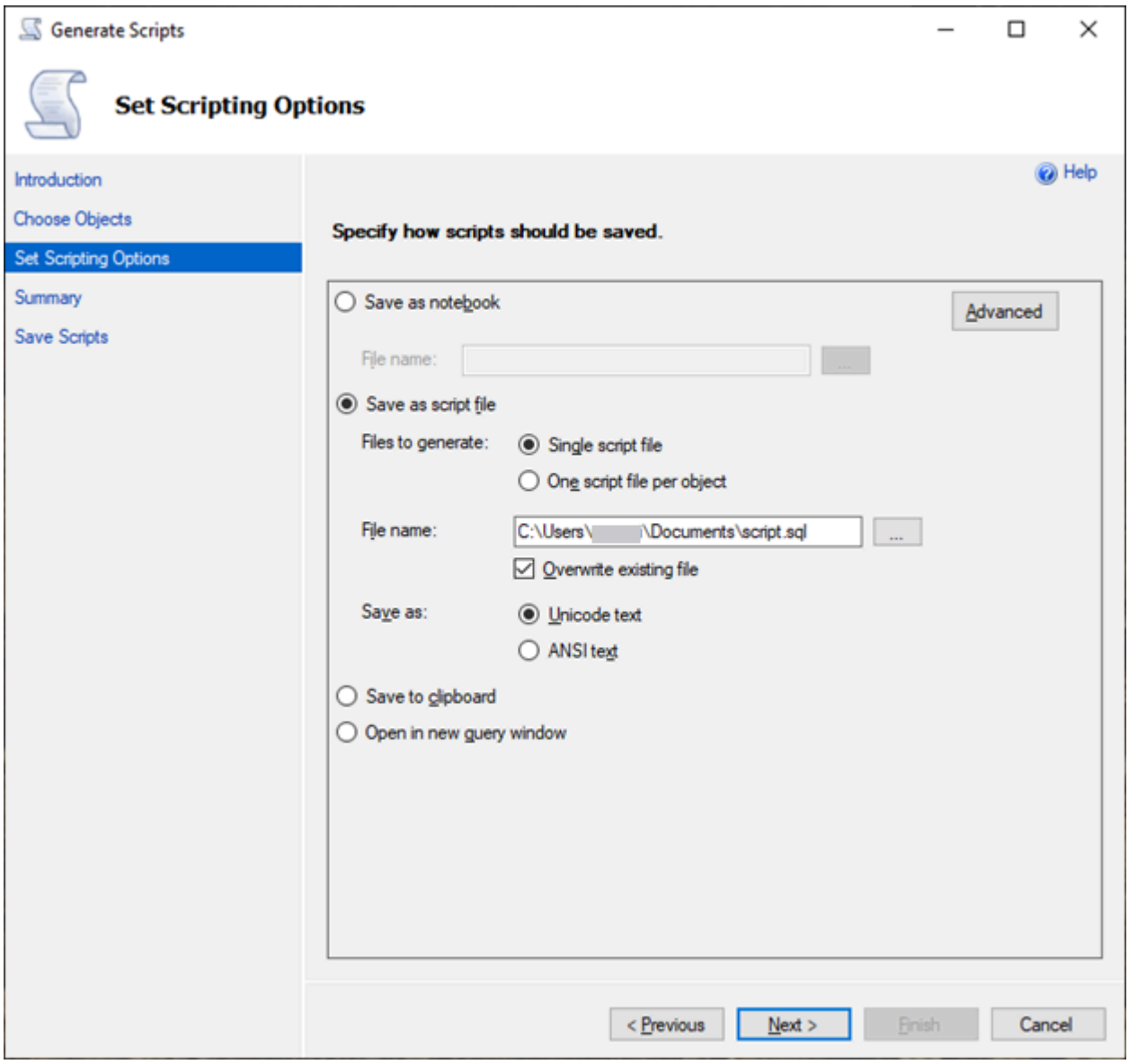
> **Note**
>
> Babelfish Compass は、オープンソースツールです。Babelfish Compass に関する問題は、AWS Support ではなく、GitHub を通じて報告してください。

SQL Server Management Studio (SSMS) でスクリプト生成ウィザードを使用して、Babelfish Compass または AWS Schema Conversion Tool CLI によって評価される SQL ファイルを生成できます。以下のステップによって評価を合理化することをお勧めします。

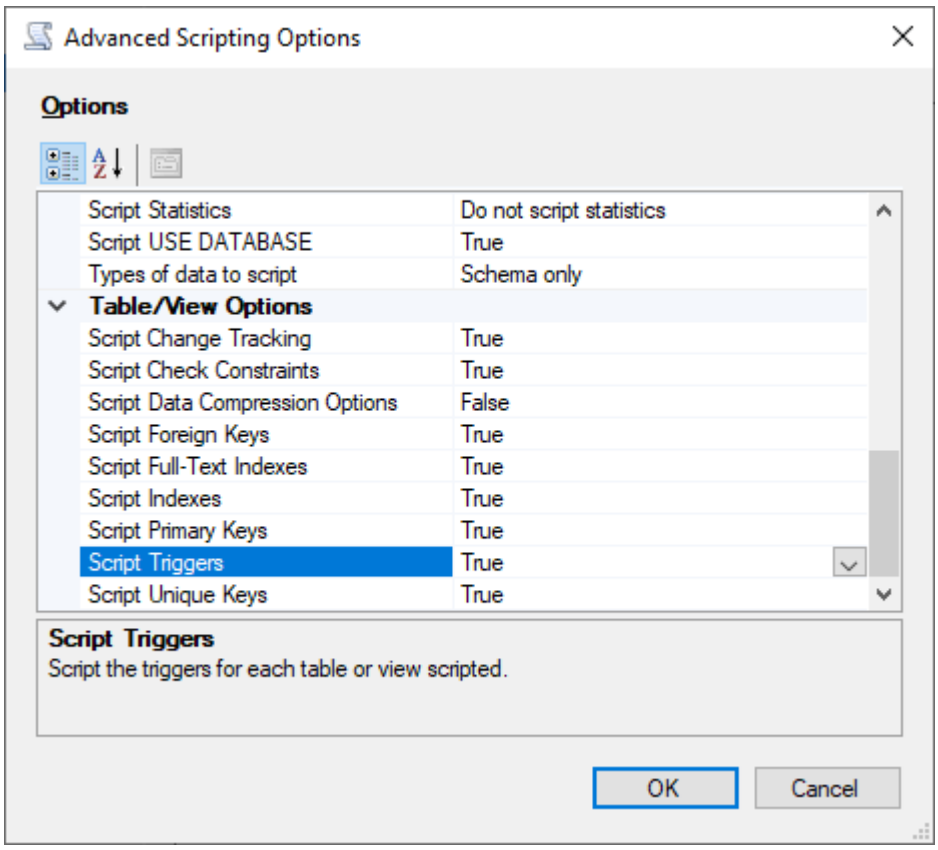
1. [Choose Objects] (オブジェクトを選択) ページで、[Script entire database and all database objects] (データベース全体とすべてのデータベースオブジェクトをスクリプト化) を選択します。



2. [Set Scripting Options] (スクリプトオプションを設定) で、[Save as script file] (名前を付けてスクリプトファイルを保存) を [Single script file] (単一のスクリプトファイル) として選択します。



3. [Advanced] (アドバンスト) を選択すると、デフォルトのスクリプトオプションが変更され、通常は完全な評価について false に設定される機能を識別できます。
   - スクリプトの変更追跡を True に
   - スクリプトの全文索引を True に
   - スクリプトのトリガーを True に
   - スクリプトのログインを True に
   - スクリプトオーナーを True に
   - スクリプトのオブジェクトレベルのアクセス許可を True に
   - スクリプトの照合順序を True に



4. ウィザードの残りの手順を実行して、ファイルを生成します。

SQL Server から Babelfish に移行するためのインポート/エクスポートツール

SQL Server から Babelfish に移行するための主要なツールとして、AWS DMS を使用することをお勧めします。ただし、Babelfish は、SQL Server ツールを使用してデータを移行する他の方法もいくつかサポートしています。これには次のものが含まれます。

- Babelfish のすべてのバージョンについて、SQL Server Integration Services (SSIS)。詳細については、「SSIS と Babelfish を使用して SQL Server から Aurora PostgreSQL に移行する」を参照してください。
- Babelfish バージョン 2.1.0 以降では、SSMS Import/Export ウィザードを使用します。このツールは SSMS から使用できますが、スタンドアロンツールとしても使用できます。詳細については、Microsoft ドキュメントの「SQL Server インポートおよびエクスポートウィザードへようこそ」を参照してください。
- Microsoft bulk data copy program (bcp) ユーティリティを使用すると、Microsoft SQL Server インスタンスから指定した形式のデータファイルにデータをコピーできます。トラブルシューティングの詳細については、Microsoft ドキュメントの「bcp ユーティリティ」を参照してくださ

い。Babelfish は BCP クライアントを使用したデータ移行をサポートするようになり、bcp ユーティリティは -E フラグ (ID 列用) と -b フラグ (バッチ挿入用) をサポートするようになりました。-C、-T、-G、-K、-R、-V、-h など、特定の bcp オプションはサポートされません。

SQL Server Management Studio (SSMS) を使用して Babelfish に移行する

特定のオブジェクトタイプごとに別々のファイルを生成することをお勧めします。最初に DDL ステートメントのセットごとに SSMS のスクリプト生成ウィザードを使用してから、オブジェクトをグループとして変更し、評価中に見つかった問題を修正できます。

AWS DMS または他のデータ移行方法を使用してデータを移行するには、次の手順を実行します。これらの作成スクリプトタイプを最初に実行すると、Aurora PostgreSQL の Babelfish テーブルに、より適切かつ迅速にデータをロードできます。

1. CREATE SCHEMA ステートメントを実行します。
2. CREATE TYPE ステートメントを実行して、ユーザー定義のデータ型を作成します。
3. プライマリキーまたは一意制約を使用して基本の CREATE TABLE ステートメントを実行します。

推奨されるインポート/エクスポートツールを使用して、データのロードを実行します。次の手順で修正したスクリプトを実行して、残りのデータベースオブジェクトを追加します。制約、トリガー、およびインデックスについて、これらのスクリプトを実行するには、create table ステートメントが必要です。スクリプトが生成されたら、create table ステートメントを削除します。

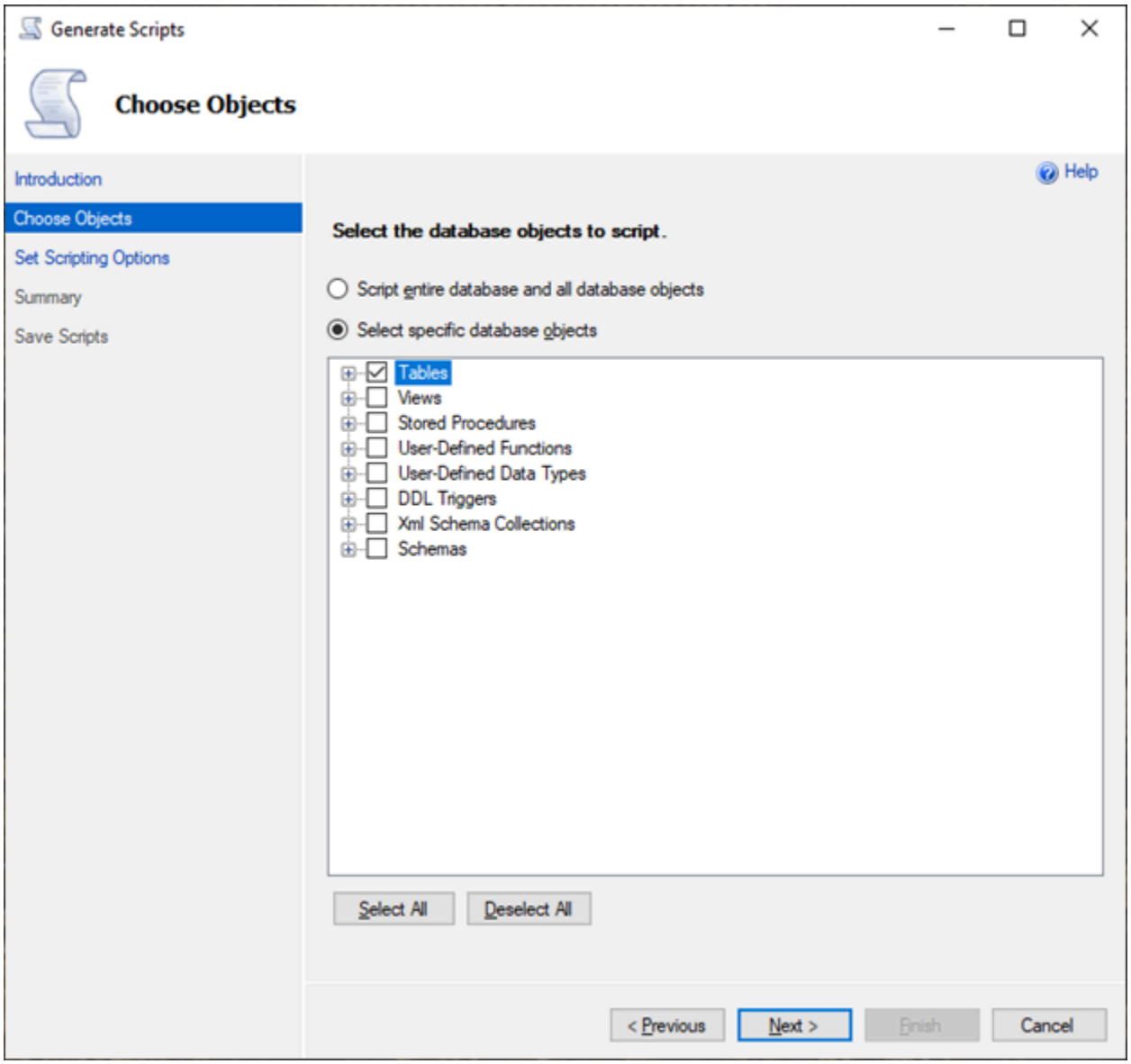
1. チェック制約、外部キー制約、デフォルト制約について、ALTER TABLE ステートメントを実行します。
2. CREATE TRIGGER ステートメントを実行します。
3. CREATE INDEX ステートメントを実行します。
4. CREATE VIEW ステートメントを実行します。
5. CREATE STORED PROCEDURE ステートメントを実行します。

各オブジェクトタイプのスクリプトを生成するには

SSMS のスクリプト生成ウィザードを使用して、基本的な create table ステートメントを作成するには、次の手順に従います。同じ手順に従って、さまざまなオブジェクトタイプのスクリプトを生成します。

1. 既存の SQL Server インスタンスに接続します。

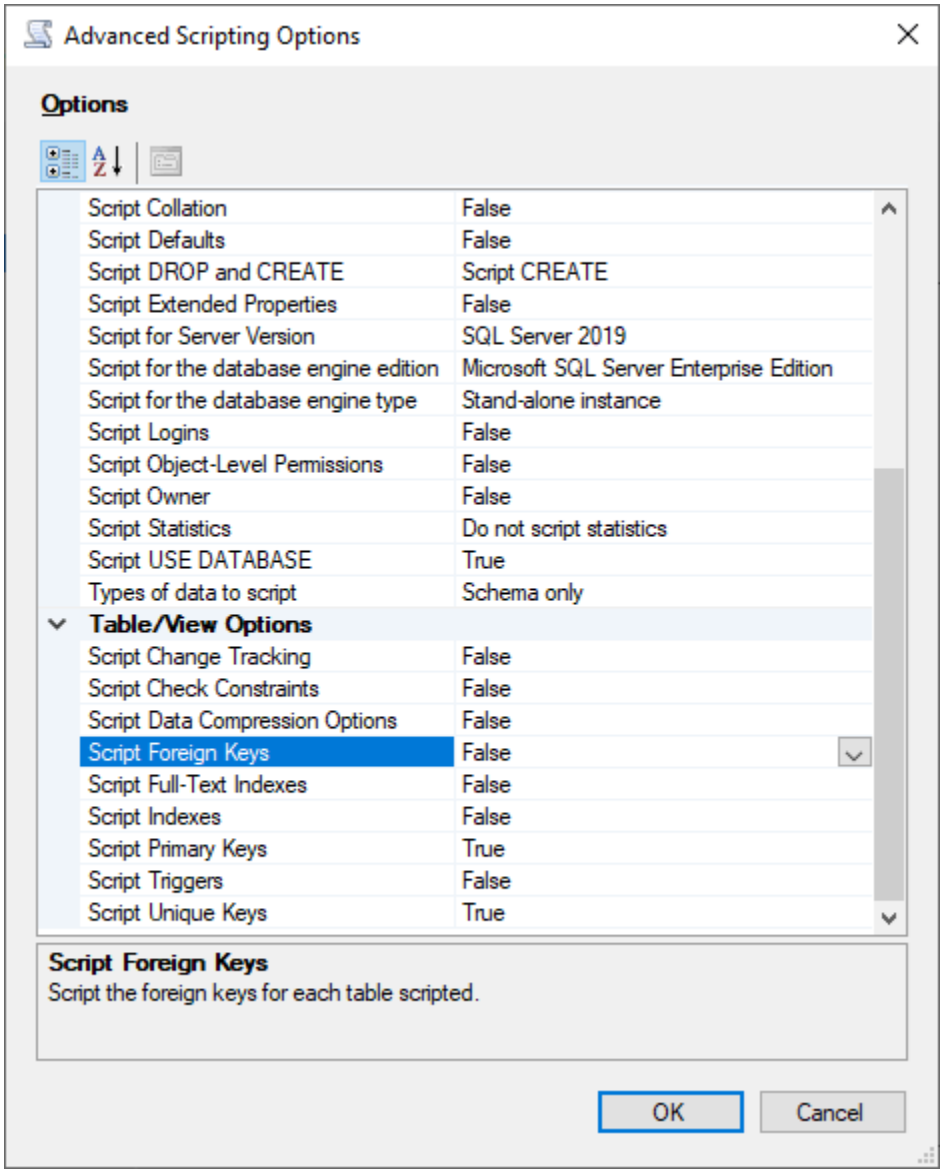
2. 右クリックでデータベース名のメニューを開きます。
3. [Tasks] (タスク) を選択し、[Generate Scripts...] (スクリプトを生成...) を選択します。
4. [オブジェクトを選択] ペインで、[特定のデータベースオブジェクトを選択する] を選択します。[Tables] (テーブル) を選択して、すべてのテーブルを選択します。次へを選択して続行します。



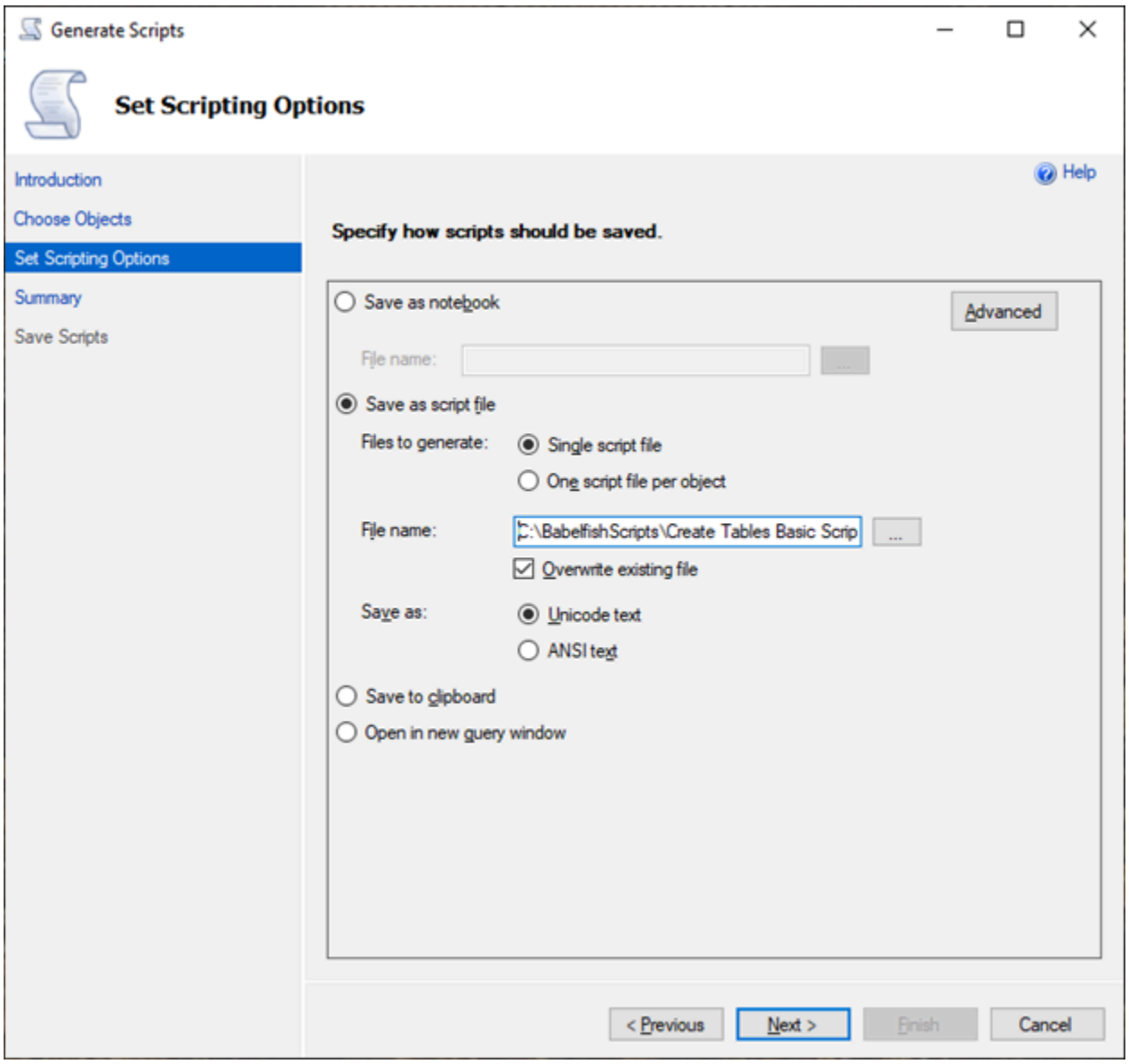
5. [スクリプトオプションの設定] を選択し、[アドバンスト] をクリックして、[オプション] 設定を開きます。基本的な create table ステートメントを生成するには、次のデフォルト値を変更します。

   - スクリプトのデフォルトを False に。

- スクリプトの拡張プロパティを False に。Babelfish は拡張プロパティをサポートしていません。
- スクリプトチェック制約を False に。スクリプトの外部キーを False に。



6. [OK] をクリックします。
7. [Set Scripting Options] (スクリプトオプションの設定) ページで、[Save as script file] (スクリプトファイルを名前を付けて保存) を選択し、[Single script file] (単一スクリプトファイル) オプションを選択します。[File name] (ファイル名) を入力します。



8. [Next] (次へ) を選択すると、[Summary wizard] (ウィザードの概要) ページが表示されます。
9. [Next] (次へ) を選択して、スクリプトの生成を開始します。

   ウィザードで他のオブジェクトタイプのスクリプトを引き続き生成できます。ファイルを保存した後、[Finish] (完了) を選択する代わりに、[Previous] (前へ) ボタンを 3 回選択して、[Choose Objects] (オブジェクトを選択) ページに戻ります。次に、ウィザードの手順を繰り返して、他のオブジェクトタイプのスクリプトを生成します。

Babelfish for Aurora PostgreSQL を使用したデータベース認証

Babelfish for Aurora PostgreSQL は、データベースユーザーを認証するいくつかの方法をサポートしています。パスワード認証は、すべての Babelfish DB クラスターでデフォルトで使用できます。同じ DB クラスターに Kerberos 認証を追加することもできます。

トピック

Babelfish を使用したパスワード認証

Babelfish for Aurora PostgreSQL はパスワード認証をサポートしています。パスワードは暗号化された形式でディスクに保存されます。Aurora PostgreSQL クラスターの認証の詳細については、Amazon Aurora PostgreSQL でのセキュリティ を参照してください。

Babelfish に接続するたびに認証情報を求められる場合があります。Aurora PostgreSQL に移行、または Aurora PostgreSQL で作成されたユーザーは、SQL Server ポートと PostgreSQL ポートの両方で同じ認証情報を使用できます。Babelfish はパスワードポリシーを強制しませんが、以下を実行することをお勧めします。

- 8 文字以上の複雑なパスワードが必要。
- パスワードの有効期限ポリシーを適用する。

データベースユーザーの完全なリストを確認するには、`SELECT * FROM pg_user;` コマンドを使用します。

Babelfish を使用した Kerberos 認証

Babelfish for Aurora PostgreSQL 15.2 バージョンでは、Kerberos を使用した DB クラスターへの認証をサポートしています。この方法により、Babelfish データベースに接続する際に、Microsoft Windows 認証を使用してユーザーを認証できます。そのためには、まず、Kerberos 認証に AWS Directory Service for Microsoft Active Directory を使用するように DB クラスターを設定する必要があります。詳細については、AWS Directory Service 管理ガイドの「AWS Directory Service とは」を参照してください。

Kerberos 認証の設定

Babelfish for Aurora PostgreSQL DB クラスターは 2 つの異なるポートを使用して接続できますが、Kerberos 認証の設定は 1 回限りのプロセスです。そのため、まず DB クラスターに Kerberos 認証を設定する必要があります。詳細については、「Kerberos 認証のセットアップ」を参照してください。設定が完了したら、Kerberos を使用して PostgreSQL クライアントに接続できることを確認します。詳細については、「Kerberos 認証に接続する」を参照してください。

Babelfish でのログインとユーザーのプロビジョニング

テーブル形式データストリーム (TDS) ポートから作成された Windows ログインは、TDS ポートまたは PostgreSQL ポートのどちらでも使用できます。まず、認証に Kerberos を使用できるログインは、T-SQL ユーザーやアプリケーションが Babelfish データベースの接続に使用する前に TDS ポートからプロビジョニングする必要があります。Windows ログインを作成する場合、管理者は DNS ドメイン名または NetBIOS ドメイン名のいずれかを使用してログインを提供できます。通常、NetBIOS ドメインは DNS ドメイン名のサブドメインです。例えば、DNS ドメイン名が `CORP.EXAMPLE.COM` の場合、NetBIOS ドメイン名は通常 `CORP` かもしれません。ログインに NetBIOS ドメイン名形式を指定する場合は、DNS ドメイン名へのマッピングが存在する必要があります。

NetBIOS ドメイン名と DNS ドメイン名マッピングの管理

NetBIOS ドメイン名と DNS ドメイン名の間のマッピングを管理するため、Babelfish はマッピングを追加、削除、切り捨てるためのシステムストアドプロシージャを提供しています。これらのプロシージャを実行できるのは、`sysadmin` ロールを持つユーザーだけです。

NetBIOS と DNS ドメイン名の間のマッピングを作成するには、Babelfish が提供するシステムストアドプロシージャ `babelfish_add_domain_mapping_entry` を使用します。どちらの引数も NULL ではなく、有効な値を持つ必要があります。

Example

```
EXEC babelfish_add_domain_mapping_entry 'netbios_domain_name',
 'fully_qualified_domain_name'
```

次の例は、NetBIOS 名の CORP と DNS ドメイン名 CORP.EXAMPLE.COM の間のマッピングを作成する方法を示しています。

Example

```
EXEC babelfish_add_domain_mapping_entry 'corp', 'corp.example.com'
```

既存のマッピングエントリを削除するには、システムストアドプロシージャ babelfish_remove_domain_mapping_entry を使用します。

Example

```
EXEC babelfish_remove_domain_mapping_entry 'netbios_domain_name'
```

次の例は、NetBIOS 名が CORP という名前のマッピングを削除する方法を示しています。

Example

```
EXEC babelfish_remove_domain_mapping_entry 'corp'
```

既存のマッピングエントリをすべて削除するには、システムストアドプロシージャ babelfish_truncate_domain_mapping_table を使用します。

Example

```
EXEC babelfish_truncate_domain_mapping_table
```

NetBIOS と DNS ドメイン名の間のマッピングをすべて表示するには、次のクエリを使用します。

Example

```
SELECT netbios_domain_name, fq_domain_name FROM babelfish_domain_mapping;
```

ログインの管理

ログインの作成

正しい権限を持つログインを使用して、TDS エンドポイント経由で DB に接続します。ログインに対するデータベースユーザーが作成されていない場合、そのログインはゲストユーザーにマッピングされます。ゲストユーザーが有効ではない場合、ログイン試行は失敗します。

次のクエリを使用して Windows ログインを作成します。FROM WINDOWS オプションでは、Active Directory を使用した認証が可能になります。

```
CREATE LOGIN login_name FROM WINDOWS [WITH DEFAULT_DATABASE=database]
```

Example

次の例では、db1 のデフォルトデータベースを使用して Active Directory ユーザー [corp\test1] のログインを作成しています。

```
CREATE LOGIN [corp\test1] FROM WINDOWS WITH DEFAULT_DATABASE=db1
```

この例では、NetBIOS ドメイン CORP と DNS ドメイン名 CORP.EXAMPLE.COM の間にマッピングがあることを前提としています。マッピングがない場合は、DNS ドメイン名 [CORP.EXAMPLE.COM\ test1] を指定する必要があります。

> **Note**
>
> Active Directory ユーザーに基づくログインは、21 文字未満の名前に制限されています。

ログインの削除

ログインを削除するには、次の例に示すように、他のログインと同じ構文を使用します。

```
DROP LOGIN [DNS domain name\login]
```

ログインの変更

ログインを変更するには、次の例のように、他のログインと同じ構文を使用します。

```
ALTER LOGIN [DNS domain name\login] { ENABLE|DISABLE|WITH DEFAULT_DATABASE=[master] }
```

ALTER LOGIN コマンドは、Windows ログインについて、次のような限定的なオプションをサポートしています。

- DISABLE — ログインを無効にします。無効になったログインを認証に使用することはできません。
- ENABLE — 無効になったログインを有効にします。
- DEFAULT_DATABASE — ログインのデフォルトデータベースを変更します。

> **Note**
>
> パスワード管理はすべて AWS Directory Service を介して行われるため、データベース管理者は、ALTER LOGIN コマンドでは Windows ログインのパスワードを変更または設定できません。

Kerberos 認証を使用して Babelfish for Aurora PostgreSQL に接続する

通常、Kerberos を使用して認証するデータベースユーザーは、クライアントマシンから認証します。これらのマシンは、Active Directory ドメインのメンバーです。クライアントアプリケーションから Windows 認証を使用して、TDS ポート上の Babelfish for Aurora PostgreSQL サーバーにアクセスします。

Kerberos 認証を使用して PostgreSQL ポートの Babelfish for Aurora PostgreSQL に接続する

TDS ポートから作成されたログインは、TDS ポートまたは PostgreSQL ポートのどちらでも使用できます。ただし、PostgreSQL は、デフォルトでユーザー名の大文字と小文字を区別して比較します。Aurora PostgreSQL によって 大文字と小文字を区別せずに Kerberos ユーザー名を解釈するためには、カスタム Babelfish クラスターパラメータグループで `krb_caseins_users` パラメータを `true` に設定する必要があります。このパラメータはデフォルトで `false` に設定されています。詳細については、「大文字と小文字を区別しないユーザー名を設定する」を参照してください。また、PostgreSQL クライアントアプリケーションから、ログインユーザー名を <login@DNS ドメイン名> の形式で指定する必要があります。<DNS ドメイン名\ログイン> の形式は使用できません。

頻繁に発生するエラー

オンプレミスの Microsoft Active Directory と AWS Managed Microsoft AD との間にフォレスト信頼関係を確立できます。詳細については、「信頼関係を作成する」を参照してください。次に、ホストエンドポイントで Amazon ドメイン `rds.amazonaws.com` を使用する代わりに、専用のドメイン固

有のエンドポイントを使用して接続する必要があります。正しいドメイン固有のエンドポイントを使用しない場合、次のエラーが発生する場合があります。

```
Error: “Authentication method "NTLMSSP" not supported (Microsoft SQL Server, Error:
 514)"
```

このエラーは、TDS クライアントが指定されたエンドポイント URL のサービスチケットをキャッシュできない場合に発生します。詳細については、「Kerberos 認証に接続する」を参照してください。

Babelfish DB クラスターへの接続

Babelfish に接続するには、Babelfish を実行している Aurora PostgreSQL クラスターのエンドポイントに接続します。クライアントは、TDS バージョン 7.1 ~ 7.4 以降に準拠した次のいずれかのクライアントドライバーを使用できます。

- Open Database Connectivity (ODBC)
- OLE DB ドライバー/ASOLEDBSQL
- Java データベース接続 (JDBC) バージョン 8.2.2 (mssql-jdbc-8.2.2) 以上
- SQL Server 用マイクロソフト SqlClient データプロバイダー
- SQL Server 用 .NET データプロバイダ
- SQL Server ネイティブクライアント 11.0 (非推奨)
- OLE DB プロバイダ / SQLOLEDB (非推奨)

Babelfish では、以下を実行します。

- デフォルトではポート 1433 の、TDS ポート上の SQL Server ツール、アプリケーション、および構文。
- PostgreSQL ポート (デフォルトではポート 5432) 上の PostgreSQL のツール、アプリケーション、および構文。

Aurora PostgreSQL への接続全般の詳細については、「Amazon Aurora PostgreSQL DB クラスターへの接続」を参照してください。

> **Note**
>
> SQL Server OLEDB プロバイダーを使用してメタデータにアクセスするサードパーティーのデベロッパーツールはサポートされていません。これらのツールには、SQL Server JDBC、ODBC、または SQL Native クライアント接続を使用することをお勧めします。

トピック

- ライターエンドポイントとポート番号の検索
- C# または JDBC クライアントから Babelfish への接続の作成
- SQL Server クライアントを使用した DB クラスターへの接続

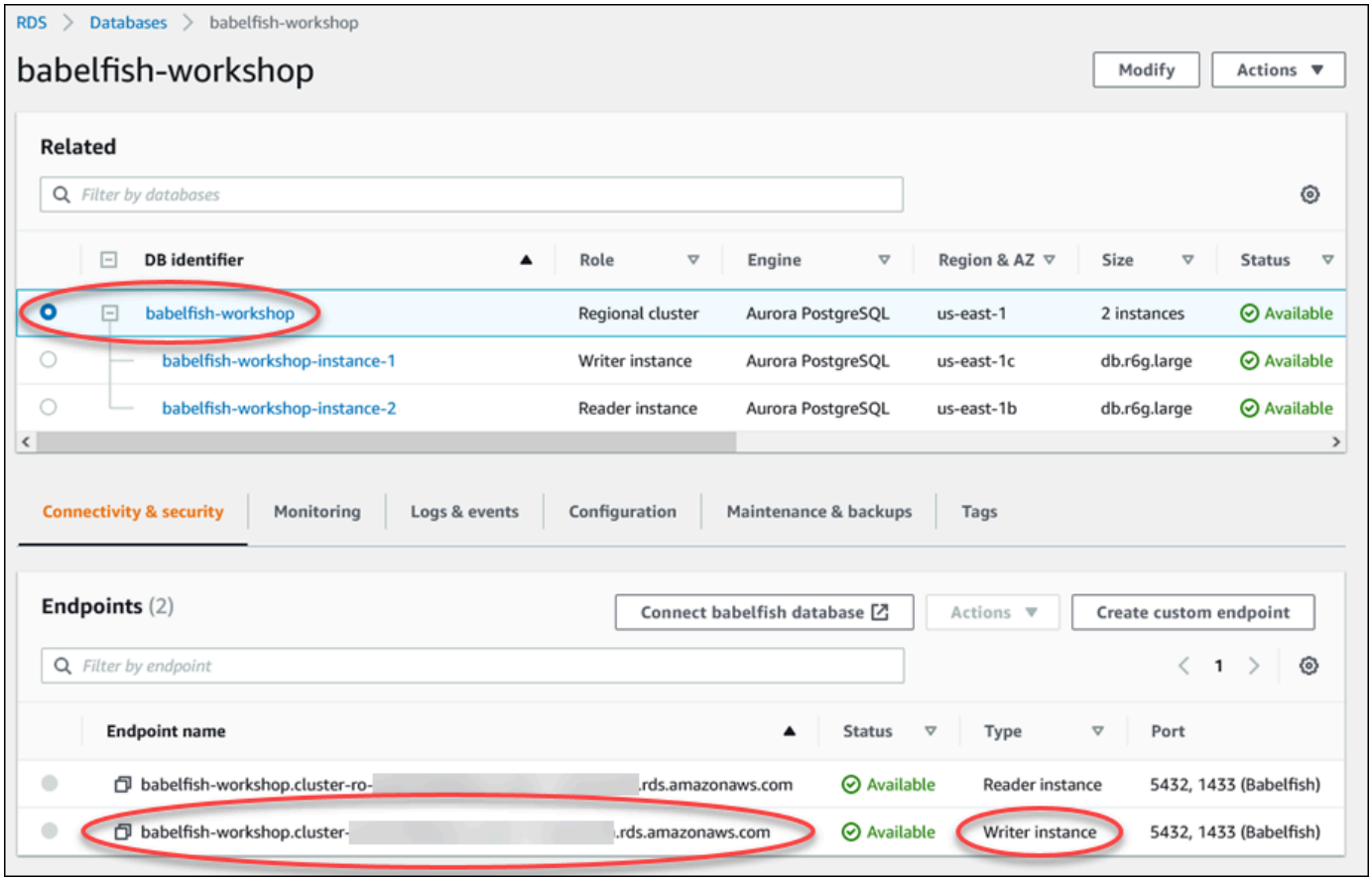
- PostgreSQL クライアントを使用した DB クラスターへの接続

ライターエンドポイントとポート番号の検索

Babelfish DB クラスターに接続するには、DB クラスターのライター (プライマリ) インスタンスに関連付けられたエンドポイントを使用します。インスタンスのステータスは [使用可能] である必要があります。Babelfish for Aurora PostgreSQL DB クラスターを作成した後、インスタンスが利用可能になるまでに最大 20 分かかる場合があります。

データベースエンドポイントを検索する方法

1. Babelfish のコンソールを開きます。
2. ナビゲーションペインから [データベース] を選択します。
3. リストされているクラスターから Babelfish for Aurora PostgreSQL DB クラスターを選択して、詳細を表示します。
4. [接続性とセキュリティ] タブで、使用可能なクラスターのエンドポイントを確認します。ライターインスタンスのクラスターエンドポイントは、書き込みまたは読み取りオペレーションを実行するすべてのアプリケーションの接続文字列で使用します。



DB クラスターの詳細については、「Amazon Aurora DB クラスターの作成」を参照してください。

C# または JDBC クライアントから Babelfish への接続の作成

以下に、C# クラスと JDBC クラスを使用して、Babelfish for Aurora PostgreSQL に接続する例をいくつか示します。

Example : C# コードを使用した DB クラスターへの接続

```
string dataSource = 'babelfishServer_11_24';

//Create connection
connectionString = @"Data Source=" + dataSource
    +";Initial Catalog=your-DB-name"
    +";User ID=user-id;Password=password";

SqlConnection cnn = new SqlConnection(connectionString);
cnn.Open();
```

Example : 一般 JDBC API クラスとインターフェースを使用した DB クラスターへの接続

```
String dbServer =
   "database-babelfish.cluster-123abc456def.us-east-1-rds.amazonaws.com";
String connectionUrl = "jdbc:sqlserver://" + dbServer + ":1433;" +
    "databaseName=your-DB-name;user=user-id;password=password";

// Load the SQL Server JDBC driver and establish the connection.
System.out.print("Connecting Babelfish Server ... ");
Connection cnn = DriverManager.getConnection(connectionUrl);
```

Example : SQL Server 固有の JDBC クラスおよびインターフェイスを使用した DB クラスターへの接続

```
// Create datasource.
SQLServerDataSource ds = new SQLServerDataSource();
ds.setUser("user-id");
ds.setPassword("password");
String babelfishServer =
   "database-babelfish.cluster-123abc456def.us-east-1-rds.amazonaws.com";

ds.setServerName(babelfishServer);
ds.setPortNumber(1433);
ds.setDatabaseName("your-DB-name");
```

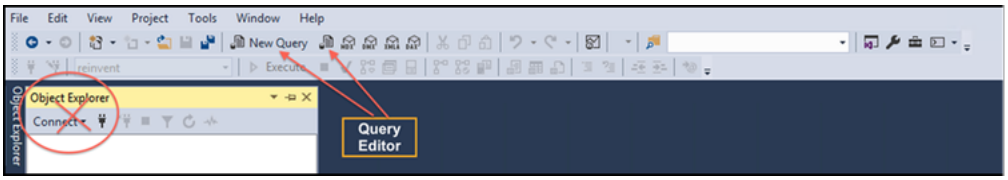
```
Connection con = ds.getConnection();
```

SQL Server クライアントを使用した DB クラスターへの接続

SQL Server クライアントを使用して、TDS ポート上の Babelfish に接続できます。Babelfish 2.1.0 以降のリリースでは、SSMS Object Explorer または SSMS Query Editor を使用して Babelfish クラスターに接続できます。

制約事項

- Babelfish 2.1.0 以前のバージョンで、PARSE を使用して、SQL 構文が正常に機能しないことを確認します。クエリを実行せずに構文をチェックするのではなく、PARSE コマンドはクエリを実行しますが、結果は表示されません。SMSS <Ctrl><F5> キーの組み合わせを使用して構文をチェックしても、正常には動作しません。つまり、Babelfish は出力なしで予期せずにクエリを実行します。
- Babelfishは MARS (複数のアクティブな結果セット) をサポートしていません。Babelfish への接続に使用するどのクライアントアプリケーションも MARS を使用するように設定されていないことを確認してください。
- Babelfish 1.3.0 以前のバージョンでは、SSMS では Query Editor のみがサポートされています。SSMS を Babelfish で使用するには、Object Explorer ではなく、必ず SSMS で Query Editor 接続ダイアログを開いてください。[Object Explorer] ダイアログボックスが開いた場合は、ダイアログボックスをキャンセルして、Query Editor を開き直します。次の図は、Babelfish 1.3.0 以前のバージョンに接続するときに選択するメニューオプションを示しています。



SQL Server と Babelfish の相互運用性と動作の違いについては、「Babelfish for Aurora PostgreSQL と SQL Server の違い」を参照してください。

sqlcmd を使用した DB クラスターへの接続

バージョン 19.1 以前の SQL Server `sqlcmd` コマンドラインクライアントを使用して、Babelfish をサポートしている Aurora PostgreSQL DB クラスターに接続して操作できます。SSMS バージョン 19.2 は Babelfish クラスターへの接続をサポートしていません。接続するには、以下のコマンドを使用します。

```
sqlcmd -S endpoint,port -U login-id -P password -d your-DB-name
```

オプションは以下のとおりです:

- -S はDB クラスターのエンドポイントおよび (オプション) TDS ポートです。
- -U はユーザーのログイン名です。
- -P はユーザーに関連付いたパスワードです。
- -d は Babelfish データベースの名前です。

接続後、SQL Server で使用するのと同じコマンドが多く使用できます。例については、「Babelfish システムカタログからの情報の入手」を参照してください。

SSMS を使用した DB クラスターへの接続

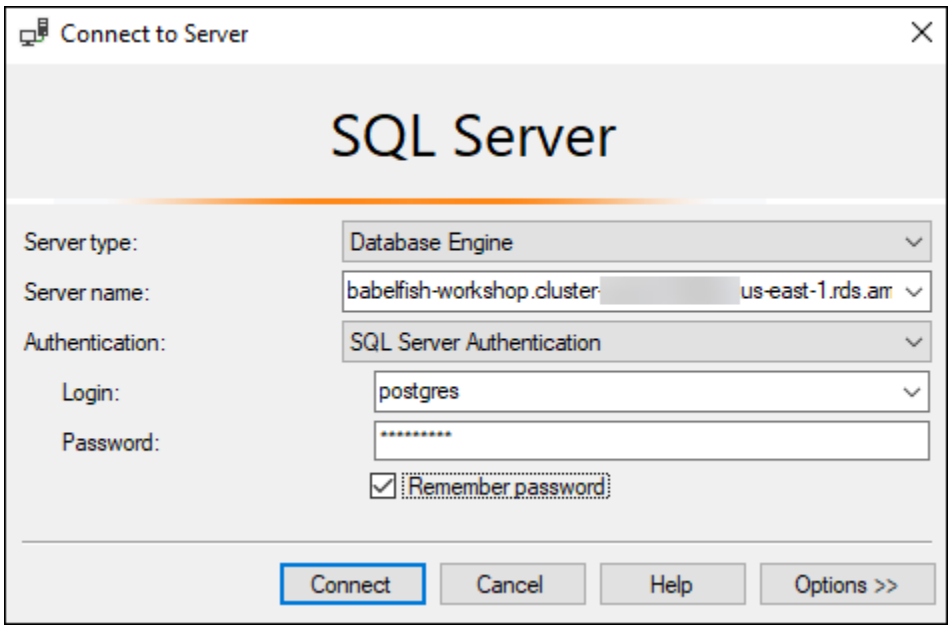
Microsoft SQL Server Management Studio (SSMS) を使用して Babelfish を実行している Aurora PostgreSQL DB クラスターに接続できます。SSMS には、「SQL Server データベースを Babelfish for Aurora PostgreSQL に移行する」で説明した SQL Server のインポートとエクスポートウィザードなど、さまざまなツールが含まれています。SSMS の詳細については、Microsoft ドキュメントの「SQL Server Management Studio (SSMS) のダウンロード」を参照してください。

SSMS で Babelfish データベースに接続する方法

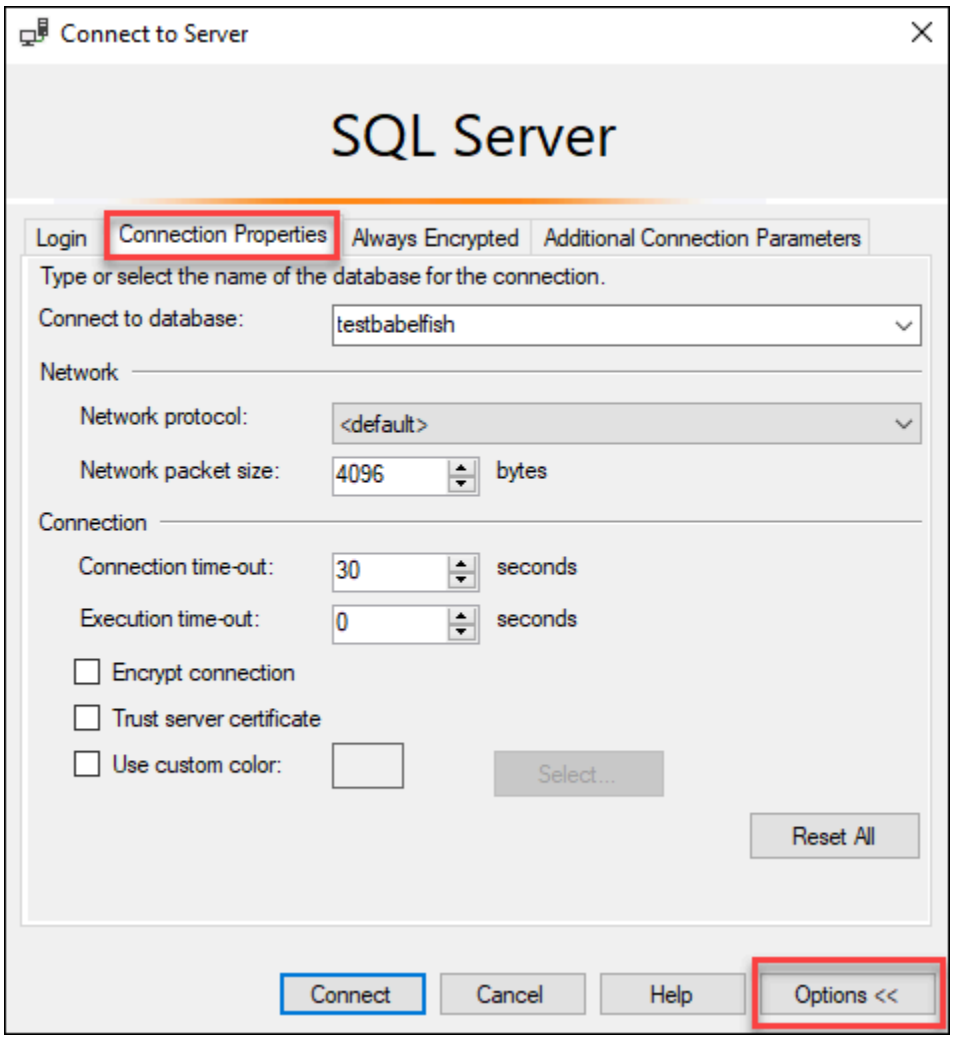
1. SSMS をスタートします。
2. [サーバーに接続] ダイアログボックスを開きます。接続を続行するには、次のいずれかの操作を行います。
   - [新しいクエリ] を選択します。
   - クエリエディタが開いている場合、[クエリ]、[接続]、[接続] を選択します。
3. 必要に応じて、データベースに次の情報を入力します:
   a. [サーバーの種類] で、[データベース エンジン] を選択します。
   b. [サーバー名] に DNS 名を入力します。サーバー名は、次の例のような見た目になっている必要があります。

      ```
      cluster-name.cluster-555555555555.aws-region.rds.amazonaws.com,1433
      ```

   c. [認証] で、[SQL Server 認証] を選択します。
   d. ログインには、データベース作成時に選択したユーザー名を入力します。
   e. パスワードには、データベース作成時に選択したパスワードを入力します。



4. (オプション) Options を選択した後、接続プロパティタブを選択します。



5. (オプション)データベースに接続するでは、接続したい移行された SQL Server データベースの名前を指定し、接続を選択します。

   SSMS が接続文字列を適用できないことを示すメッセージが表示されたら、[OK] を選択します。

   Babelfish に接続する際に問題が発生する場合は、「接続障害」を参照してください。

   SQL Server の接続問題の詳細については、Amazon RDS ユーザーガイドの「SQL Server DB インスタンスへの接続のトラブルシューティング」を参照してください。

PostgreSQL クライアントを使用した DB クラスターへの接続

PostgreSQL クライアントを使用して、PostgreSQL ポートの Babelfish に接続できます。

psql を使用したクラスターへの接続

PostgreSQL クライアントは、PostgreSQL のウェブサイトからダウンロードできます。オペレーティングシステムのバージョンに対応した手順に従い、psql をインストールします。

Babelfish をサポートしている Aurora PostgreSQL DB クラスターは、`psql` コマンドラインクライアントでクエリすることができます。接続する際は、 PostgreSQL ポートを使用します (デフォルトはポート 5432)。通常、デフォルトから変更しない限り、ポート番号を指定する必要はありません。`psql` クライアントから Babelfish に接続する場合は、次のコマンドを使用します。

```
psql -h bfish-db.cluster-123456789012.aws-region.rds.amazonaws.com
-p 5432 -U postgres -d babelfish_db
```

パラメータは次のとおりです。

- `-h` - アクセス先の DB クラスター (クラスターエンドポイント) のホスト名。
- `-p` - DB インスタンスへの接続に使用される PostgreSQL ポート番号。
- `-d` — 接続先のデータベース。デフォルトは `babelfish_db` です。
- `-U` - アクセス先のデータベースユーザーアカウント。(この例では、デフォルトのマスターユーザー名を示します) 。

psql クライアントで SQL コマンドを実行する場合、コマンドはセミコロンで終了します。例えば、次の SQL コマンドでは pg_tables system view をクエリして、データベース内の各テーブルに関する情報を返します。

```
SELECT * FROM pg_tables;
```

psql クライアントには、一連の組み込みメタコマンドもあります。metacommand (メタコマンド) はフォーマットを調整するショートカットで、またメタデータを使いやすい形式で返すショートカットを利用可能にします。例えば、 次のメタコマンドは以前の SQL コマンドに似た情報を返します:

```
\d
```

メタコマンドはセミコロン (;) で終了する必要がありません。

psql クライアントを終了するには、次のように入力します\q。

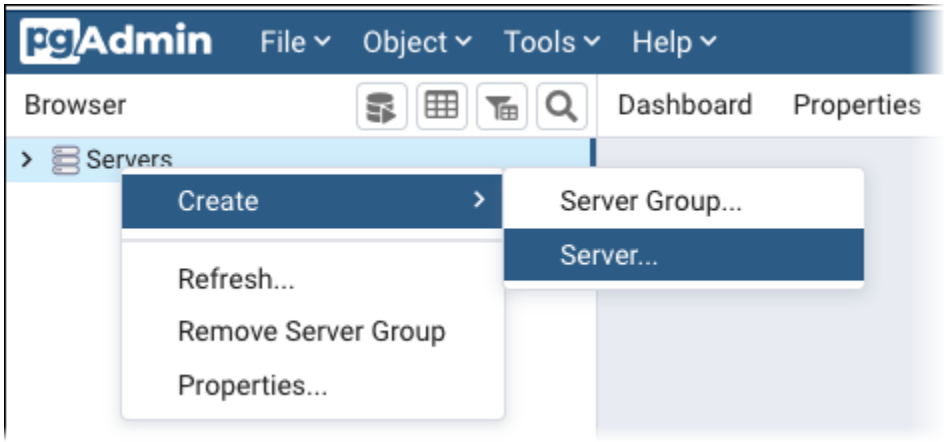
psql クライアントを使用して Aurora PostgreSQL クラスターをクエリする方法の詳細については、PostgreSQL ドキュメントを参照してください。

pgAdmin を使用した DB クラスターへの接続

pgAdmin クライアントを使用して、ネイティブの PostgreSQL ダイアレクトでデータにアクセスできます。

pgAdmin クライアントを使用してクラスターに接続する方法

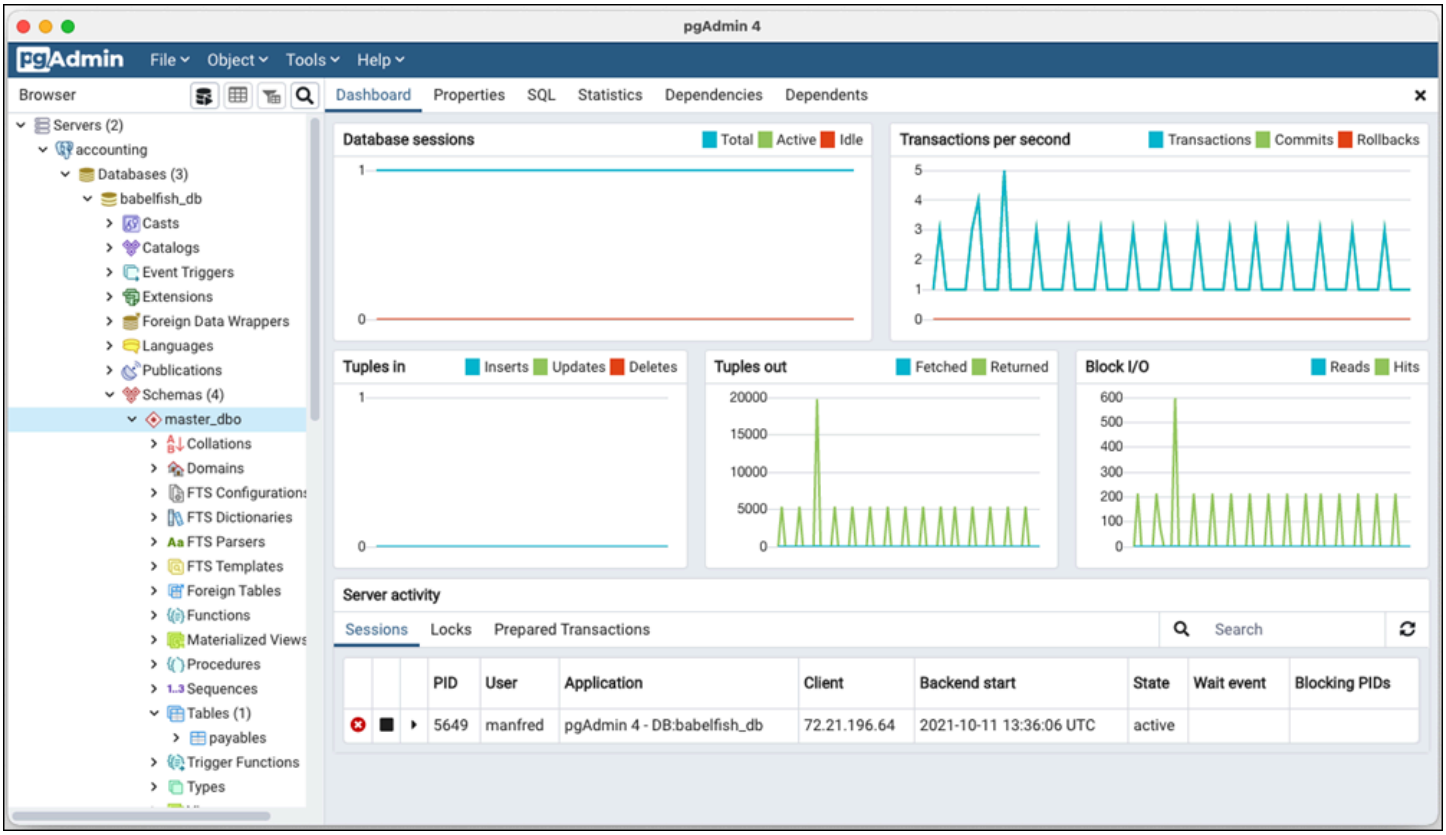
1. pgAdmin ウェブサイトから pgAdmin クライアントをダウンロードしてインストールします。
2. クライアントを開き、pgAdmin で認証します。
3. サーバーのコンテキスト (右クリック) メニューを開き、「作成」、「サーバー」の順に選択します。



4. Create - Server (サーバー - 作成) ダイアログボックスに情報を入力します。

   接続タブで、ホストに Aurora PostgreSQL クラスターアドレスを、ポートに PostgreSQL のポート番号 (デフォルトでは 5432) を追加します。認証の詳細を入力し、保存 を選択します。

接続後、Pgadmin 機能を使用して PostgreSQL ポートで Aurora PostgreSQL クラスターをモニタリングおよび管理できます。



詳細については、「pgAdmin」Web ページを参照してください。

Babelfish での作業

以下に、Babelfish と SQL Server での作業、および Babelfish データベースと PostgreSQL データベース間の違いなど、Babelfish の使用情報を確認できます。

トピック

Babelfish システムカタログからの情報の入手

SQL Server で使用するのと同じシステムビューを多数クエリすることにより、Babelfish クラスターに格納されているデータベースオブジェクトに関する情報を取得できます。Babelfish の新しいリリースごとに、より多くのシステムビューがサポートされます。現在使用可能なビューのリストについては、SQL Server system catalog views テーブルを参照してください。

これらのシステムビューは、システムカタログ (`sys.schemas`) からの情報を提供します。Babelfish の場合、これらのビューには SQL Server と PostgreSQL の両方のシステムスキーマが含まれます。Babelfish でシステムカタログ情報を照会するには、次の例に示すように、TDS ポートまたは PostgreSQL ポートを使用できます。

- **`sqlcmd`** または別の SQL Server クライアントを使用して、T-SQL ポートをクエリします。

```
1> SELECT * FROM sys.schemas
2> GO
```

このクエリは、次に示すように SQL Server および Aurora PostgreSQL システムスキーマを返します。

```
name
-----------------------------------------------------------
```

```
demographic_dbo
public
sys
master_dbo
tempdb_dbo
...
```

- **psql**または**pgAdmin** を使用して PostgreSQL ポートをクエリします。この例では psql リストスキーマメタコマンド (\dn):を使用します

  ```
  babelfish_db=> \dn
  ```

  クエリは、T-SQL ポート上で sqlcmd によって返された結果と同じ結果セットを返します。

  ```
          List of schemas
               Name
------------------------------

 demographic_dbo

 public
 sys
 master_dbo
 tempdb_dbo
...
  ```

Babelfish で利用可能な SQL Server システムカタログ

次の表は、Babelfish に現在実装されている SQL Server ビューです。SQL Server のシステムカタログの詳細については、Microsoft ドキュメントの「システムカタログビュー (Transact-SQL)」を参照してください。

ビュー名	説明または Babelfish 制限事項 (ある場合)
sys.all_columns	すべてのテーブルとビューのすべての列
sys.all_objects	すべてのスキーマ内のすべてのオブジェクト
sys.all_sql_modules	sys.sql_modules と sys.system_sql_modules の統合

ビュー名	説明または Babelfish 制限事項 (ある場合)
sys.all_views	すべてのスキーマのすべてのビュー
sys.columns	ユーザー定義のテーブルおよびビューのすべての列
sys.configurations	Babelfish のサポートは、単一の読み取り専用構成に制限されています。
sys.data_spaces	各データスペースの行が含まれます。これには、ファイルグループ、パーティションスキーム、または FILESTREAM データファイルグループを指定できます。
sys.database_files	データベース自体に格納されているデータベースのファイルごとに 1 つの行を含むデータベースごとのビュー。
sys.database_mirroring	詳細については、Microsoft Transact-SQLのドキュメントの「sys.database_mirroring」を参照してください。
sys.database_principals	詳細については、Microsoft Transact-SQLのドキュメントの「sys.database_principals」を参照してください。
sys.database_role_members	詳細については、Microsoft Transact-SQLのドキュメントの「sys.database_role_members」を参照してください。
sys.databases	すべてのスキーマ内のすべてのデータベース
sys.dm_exec_connections	詳細については、Microsoft Transact-SQLのドキュメントの「sys.dm_exec_connections」を参照してください。

ビュー名	説明または Babelfish 制限事項 (ある場合)
sys.dm_exec_sessions	詳細については、Microsoft Transact-SQLのドキュメントの「sys.dm_exec_sessions」を参照してください。
sys.dm_hadr_database_replica_states	詳細については、Microsoft Transact-SQLのドキュメントの「sys.dm_hadr_database_replica_states」を参照してください。
sys.dm_os_host_info	詳細については、Microsoft Transact-SQLのドキュメントの「sys.dm_os_host_info」を参照してください。
sys.endpoints	詳細については、Microsoft Transact-SQLのドキュメントの「sys.endpoints」を参照してください。
sys.indexes	詳細については、Microsoft Transact-SQLのドキュメントの「sys.indexes」を参照してください。
sys.languages	詳細については、Microsoft Transact-SQLのドキュメントの「sys.languages」を参照してください。
sys.schemas	すべてのスキーマ
sys.server_principals	すべてのログインとロール
sys.sql_modules	詳細については、Microsoft Transact-SQLのドキュメントの「sys.sql_modules」を参照してください。
sys.sysconfigures	Babelfish のサポートは、単一の読み取り専用構成に制限されています。
sys.syscurconfigs	Babelfish のサポートは、単一の読み取り専用構成に制限されています。

ビュー名	説明または Babelfish 制限事項 (ある場合)
sys.sysprocesses	詳細については、Microsoft Transact-SQLのドキュメントの「sys.sysprocesses」を参照してください。
sys.system_sql_modules	詳細については、Microsoft Transact-SQLのドキュメントの「sys.system_sql_modules」を参照してください。
sys.table_types	詳細については、Microsoft Transact-SQLのドキュメントの「sys.table_types」を参照してください。
sys.tables	スキーマ内のすべてのテーブル
sys.xml_schema_collections	詳細については、Microsoft Transact-SQLのドキュメントの「sys.xml_schema_collections」を参照してください。

PostgreSQL は、SQL Server オブジェクトカタログビューに似たシステムカタログを実装しています。システムカタログの詳細なリストについては、PostgreSQL ドキュメントの「システムカタログ」を参照してください。

Babelfish がサポートする DDL エクスポート

Babelfish 2.4.0 および 3.1.0 バージョン以降、Babelfish はさまざまなツールを使用する DDL エクスポートをサポートしています。例えば、SQL Server Management Studio (SSMS) からこの機能を使用して、Babelfish for Aurora PostgreSQL データベース内のさまざまなオブジェクトのデータ定義スクリプトを生成できます。その後、このスクリプトで生成された DDL コマンドを使用して、同じオブジェクトを別の Babelfish for Aurora PostgreSQL または SQL Server データベースに作成できます。

Babelfish は、指定されたバージョンで、以下のオブジェクトの DDL エクスポートをサポートしています。

オブジェクトのリスト	2.4.0	3.1.0
ユーザーテーブル	はい	はい
プライマリキー	はい	はい
外部キー	はい	はい
一意の制約	はい	はい
インデックス	はい	はい
検査制約	はい	はい
ビュー	はい	はい
ストアドプロシージャ	はい	はい
ユーザー定義関数	はい	はい
テーブル値関数	はい	はい
トリガー	はい	はい
ユーザー定義のデータ型	いいえ	いいえ
ユーザー定義のテーブルタイプ	いいえ	いいえ
[ユーザー]	いいえ	いいえ
ログイン	いいえ	いいえ
シーケンス	いいえ	いいえ
ロール	いいえ	いいえ

エクスポートされた DDL による制限事項

- エクスポートされた DDL でオブジェクトの再作成前にエスケープハッチを使用する – Babelfish では、エクスポートされた DDL スクリプトのすべてのコマンドをサポートしているわけではありません。Babelfish の DDL コマンドからオブジェクトを再作成するときに発生するエラーを回避する

ため、エスケープハッチを使用します。エスケープハッチの詳細については、「エスケープハッチ処理時の Babelfish のエラー処理の管理」を参照してください。

- 明示的な COLLATE 句を含む CHECK 制約を含むオブジェクト — SQL Server データベースから生成されたこれらのオブジェクトを含むスクリプトの照合順序は、Babelfish データベースとは異なりますが同等です。例えば、sql_latin1_general_cp1_cs_as、sql_latin1_general_cp1251_cs_as、latin1_general_cs_as などの一部の照合は、Windows の照合に最も近い latin1_general_cs_as として生成されます。

Babelfish for Aurora PostgreSQL と SQL Server の違い

Babelfish は進化を続ける Aurora PostgreSQL の機能であり、Aurora PostgreSQL 13.4 での最初の提供以降、リリースごとに新しい機能が追加されています。TDS ポートを使用して、T-SQL 言語経由で、Postgre SQL の上に T-SQL セマンティクスを提供するように設計されています。バージョンごとに Babelfish でサポートされている機能 テーブルに示すように、Babelfish のそれぞれの新しいバージョンでは、T-SQL の機能や動作にさらに適した機能と関数が追加されています。Babelfish を使用する際に最良の結果を得るには、SQL Server でサポートされている T-SQL と最新バージョンの Babelfish の間に現存する違いを理解することをお勧めします。詳細については、「Babelfish での T-SQL の違い」を参照してください。

Babelfish と SQL Server でサポートされている T-SQL の違いに加えて、Aurora PostgreSQL DB クラスターのコンテキストでの Babelfish と PostgreSQL の間の相互運用性の問題も考慮する必要がある可能性があります。前述のとおり、Babelfish は、TDS ポートを使用して、T-SQL 言語経由で、Postgre SQL の上で T-SQL セマンティクスをサポートしています。同時に、PostgreSQL SQL ステートメントを使用して標準の PostgreSQL ポートから Babelfish データベースにアクセスすることもできます。PostgreSQL と Babelfish の両方の機能を本番デプロイで使用することを検討している場合、スキーマ名、識別子、アクセス権限、トランザクションセマンティクス、複数の結果セット、デフォルトの照合などの間の潜在的な相互運用性の問題を認識する必要があります。簡単に言うと、PostgreSQL ステートメントまたは PostgreSQL アクセスが Babelfish のコンテキストで発生すると、PostgreSQL と Babelfish の間で干渉が発生する可能性があり、Babelfish の新しいバージョンがリリースされたときに構文、セマンティクス、および互換性に影響を与える可能性があります。すべての考慮事項に関する詳細な情報とガイダンスについては、Babelfish for PostgreSQL のドキュメントにある「Guidance on Babelfish Interoperability」(Babelfish の相互運用性に関するガイダンス)を参照してください。

> **Note**
>
> PostgreSQL のネイティブ機能と Babelfish の機能の両方を同じアプリケーションコンテキストで使用する前に、Babelfish for PostgreSQL のドキュメントの「Guidance on Babelfish Interoperability」(Babelfish の相互運用性に関するガイダンス) で説明されている問題を検討することを強くお勧めします。これらの相互運用性の問題 (Aurora PostgreSQL と Babelfish) は、Babelfish と同じアプリケーションコンテキストで PostgreSQL データベースインスタンスを使用する予定の場合にのみ関係します。

トピック

Babelfish ダンプと復元

バージョン 4.0.0 および 3.4.0 以降、Babelfish ユーザーはダンプユーティリティと復元ユーティリティを使用してデータベースをバックアップおよび復元できるようになりました。詳細については、「Babelfish ダンプと復元」を参照してください。この機能は PostgreSQL ダンプユーティリティと復元ユーティリティ上に構築されています。詳細については、「pg_dump」と「pg_restore」を参照してください。Babelfish でこの機能を効果的に使用するには、Babelfish 用に特別にカスタマイズされた PostgreSQL ベースのツールを使用する必要があります。Babelfish のバックアップおよび復元機能は、SQL Server のバックアップおよび復元機能とは大きく異なります。これらの違いの詳細については、「ダンプと復元の機能の違い: Babelfish と SQL Server」を参照してください。Babelfish for Aurora PostgreSQL には、Amazon Aurora PostgreSQL DB クラスターをバックアップおよび復元するための追加機能が用意されています。詳細については、「Amazon Aurora DB クラスターのバックアップと復元」を参照してください。

Babelfish での T-SQL の違い

以下に、Babelfish の現在のリリースでサポートされている T-SQL 機能の表と SQL Server 用との動作の違いに関する注意事項を示します。

さまざまなバージョンのサポートの詳細については、「バージョンごとに Babelfish でサポートされている機能」を参照してください。現在サポートされていない機能の詳細については、「Babelfish でサポートされていない機能」を参照してください。

Babelfish は、Aurora PostgreSQL 互換エディションで使用可能です。Babelfish のリリースの詳細については、「Aurora PostgreSQL リリースノート」を参照してください。

機能または構文	動作または違いの説明
\ (行継続文字)	文字および 16 進数の文字列の行継続文字 (改行の前のバックスラッシュ) は現在サポートされていません。文字列の場合、バックスラッシュ改行は文字列内の文字として解釈されます。16 進数の文字列の場合、バックスラッシュ改行は構文エラーになります。
@@バージョン	`@@version` によって返される値の形式は SQL Server によって返される値とは若干異なります。`@@version` のフォーマットによっては、コードが正しく動作しないことがあります。
集計関数	集計関数は部分的にサポートされています (AVG、COUNT、COUNT_BIG、GROUPING、MAX、MIN、STRING_AGG、SUM がサポートされています)。サポートされていない集計関数のリストについては、サポートされていない関数 を参照してください。
ALTER TABLE	単一の列または制約の追加または削除のみをサポートします。
ALTER TABLE..ALTER COLUMN	NULL と NOT NULL は現在指定できません。カラムのヌル可能性を変更するには、PostgreSQL ステートメント ALTER TABLE..{SET|DROP} NOT NULL を使用してください。
列エイリアスのない空白の列名	`sqlcmd` と `psql` ユーティリティは、空白の名前を持つ列を異なる方法で処理します: • SQL Server `sqlcmd` は空の列名を返します。 • PostgreSQL `psql` は生成されたカラム名を返します。
CHECKSUM 関数	Babelfish と SQL Server は CHECKSUM 関数で異なるハッシュアルゴリズムを使用しています。その結果、Babelfish の CHECKSUM 関数によって生成されるハッシュ値は、SQL

機能または構文	動作または違いの説明
	Server の CHECKSUM 関数によって生成されるハッシュ値と異なる場合があります。
列のデフォルト	列のデフォルトを作成する場合、制約名は無視されます。列のデフォルトを削除するには、次の構文を使用します: `ALTER TABLE...ALTER COLUMN..DROP DEFAULT...`
制約	PostgreSQL は、個々の制約のオンとオフをサポートしていません。ステートメントは無視され、警告が表示されます。
DESC (降順) 列で作成された制約	制約は ASC (昇順) 列で作成されます。
IGNORE_DUP_KEY による制約	制約は、このプロパティなしで作成されます。
サーバーロールの作成、変更、削除	ALTER SERVER ROLE は、`sysadmin` のみでサポートされています。その他の構文はサポートされていません。 Babelfish の T-SQL ユーザーは、ログイン (サーバープリンシパル)、データベース、およびデータベースユーザー (データベースプリンシパル) の概念について、SQL Server と類似の経験をします。
CREATE、ALTER LOGIN 句は、限定された構文でサポートされています	ログイン作成..。パスワード句、...DEFAULT_DATABASE 句、および ...DEFAULT_LANGUAGE 句がサポートされています。ログイン変更..。PASSWORD 句はサポートされていますが、ALTER LOGIN..。OLD_PASSWORD 句はサポートされていません。パスワードを変更できるのは、sysadmin メンバーのログインのみです。
大文字と小文字を区別する照合順序を作成	CREATE DATABASE ステートメントでは、大文字と小文字を区別する照合はサポートされていません。

機能または構文	動作または違いの説明
CREATE DATABASE キーワードと句	COLLATE および CONTAINMENT=NONE 以外のオプションはサポートされていません。COLLATE 句は受け入れられ、常に `babelfishpg_tsql.server_collation_name` の値に設定されます。
CREATE SCHEMA... サポート句	CREATE SCHEMA コマンドを使用して、空のスキーマを作成できます。追加のコマンドを使用して、スキーマオブジェクトを作成します。
データベース ID の値は Babelfish で異なります	マスターデータベースと tempdb データベースは、データベース ID 1 および 2 にはなりません。
TO_CHAR 日付型フォーマット関数は、次の制限付きでサポートされています	1 文字の午前/午後はサポートされていません。 SQL サーバーでは、「yyy」形式は、1000 年を超える年では 4 桁を返しますが、それ以外の場合は 3 桁しか返しません。 「g」および「R」形式はサポートされていません 「Vi-VN」ロケールの翻訳は少し異なります。
63 文字を超える識別子	PostgreSQL は、最大 63 文字の識別子をサポートしています。Babelfish は 63 文字を超える識別子を、元の名前のハッシュを含んだ名前に変換します。例えば、「AB(ABC1234567890123456789012345678901234567890123456789012345678901234567890」として作成された表は、「ABC12345678901234567890123456789012345678901234567890123456789012345678901234567890」に変換される可能性があります。

機能または構文	動作または違いの説明
IDENTITY 列のサポート	IDENTITY 列は tinyint、smallint、int、bigint、numeric、 および decimal のデータ型でサポートされています。 SQL Server は、IDENTITY 列で numeric と decimal のデータ型の精度を 38 か所までサポートしています。 PostgreSQL Server は、IDENTITY 列で numeric と decimal のデータ型の精度を 19 か所までサポートしています。
IGNORE_DUP_KEY のインデックス	IGNORE_DUP_KEY を含むインデックスを作成する構文は、このプロパティを省略したかのようにインデックスを作成します。
32 列を超えるインデックス	インデックスには 32 個を超える列を含めることはできません。含まれるインデックス列は PostgreSQL では最大までカウントされますが、SQL Server ではカウントされません。
インデックス (クラスター)	クラスター化したインデックスは、NONCLUSTERED が指定されているかのように作成されます。
インデックス句	次の句は無視されます: FILLFACTOR、ALLOW_PAGE_LOCKS、ALLOW_ROW_LOCKS、PAD_INDEX、STATISTICS_NORECOMPUTE、OPTIMIZE_FOR_SEQUENTIAL_KEY、SORT_IN_TEMPDB、DROP_EXISTING、ONLINE、COMPRESSION_DELAY、MAXDOP、および DATA_COMPRESSION
JSON サポート	名前と値のペアの順序は保証されていません。ただし、配列タイプは影響を受けません。
LOGIN オブジェクト	PASSWORD、DEFAULT_DATABASE、ENABLE、DISABLE を除く LOGIN オブジェクトのすべてのオプションがサポートされています。

機能または構文	動作または違いの説明
NEWSEQUENTIALID 関数	NEWID として実装されたシーケンシャル動作は保証されません。NEWSEQUENTIALID を呼び出す際、PostgreSQL は新しい GUID 値を生成します。
OUTPUT 句は、以下の制限付きでサポートされています。	同じ DML クエリでは、OUTPUT と OUTPUT INTO はサポートされていません。OUTPUT 句内の UPDATE または DELETE オペレーションのターゲット以外のテーブルへの参照はサポートされていません。OUTPUT... DELETED *、INSERTED * は同じクエリではサポートされていません。
プロシージャまたは関数パラメータの制限	Babelfish は、プロシージャまたは関数に対して最大 100 個のパラメータをサポートしています。
ROWGUIDCOL	現在のところ、この句は無視されます。参照するクエリ$GUIDGOL構文エラーを引き起こします。
SEQUENCE オブジェクトサポート	SEQUENCE オブジェクトは、tinyint、smallint、int、bigint、numeric、および decimal のデータ型でサポートされています。 Aurora PostgreSQL は、SEQUENCE 内の数値および小数点のデータ型の精度を 19 桁までサポートしています。
サーバーレベルのロール	sysadmin サーバーレベルのロールがサポートされています。その他のサーバーレベルのロール (sysadmin 以外) はサポートされていません。
db_owner 以外のデータベースレベルのロール	db_owner データベースレベルのロールとユーザー定義のデータベースレベルのロールがサポートされています。その他のデータベースレベルのロール (db_owner 以外) はサポートされていません。
SQL キーワード SPARSE	キーワード SPARSE は受け入れられ、無視されます。
SQL キーワード句 ON filegroup	現在のところ、この句は無視されます。

機能または構文	動作または違いの説明
インデックスと制約の SQL キーワード CLUSTERED と NONCLUSTERED	Babelfish は CLUSTERED と NONCLUSTERED キーワードを受け入れ、無視します。
sysdatabases.cmptlevel	sysdatabases.cmptlevel は常に 120 に設定されます。
tempdb は再起動時に再初期化されません	tempdb で作成された永続オブジェクト (テーブルやプロシージャなど) は、データベースを再起動しても削除されません。
TEXTIMAGE_ON ファイルグループ	Babelfish は TEXTIMAGE_ON *filegroup* 句を無視します。
時間精度	Babelfish は 少数秒 で 6 桁の精度をサポートします。この動作では、悪影響は予想されません。
トランザクション分離レベル	READUNCOMMITTED は READCOMMITTED と同じように扱われます
仮想計算列 (非永続的)	仮想計算列は永続として作成されます。
WITHOUT SCHEMABINDING 句	この句は、関数、プロシージャ、トリガー、またはビューでサポートされていません。オブジェクトは作成されますが、WITH SCHEMABINDING が指定されているかのようになります。

Babelfish でのトランザクション分離レベル

Babelfish は、トランザクション分離レベル READ UNCOMMITTED、READ COMMITTED、SNAPSHOT をサポートしています。Babelfish 3.4 バージョン以降、追加の分離レベル REPEATABLE READ と SERIALIZABLE がサポートされています。Babelfish のすべての分離レベルは、PostgreSQL の対応する分離レベルの動作でサポートされています。SQL Server と Babelfish は、トランザクション分離レベル (同時アクセスのブロック、トランザクションによって保持されるロック、エラー処理など) を実装するための基盤となるさまざまなメカニズムを使用します。また、同時アクセスがワークロードによってどのように機能するかには微妙な違いがいくつかあります。この PostgreSQL の動作の詳細については、「トランザクション分離」を参照してください。

トピック

- トランザクション分離レベルの概要。
- トランザクション分離レベルのセットアップ
- トランザクション分離レベルの有効化または無効化
- Babelfish と SQL Server の分離レベルの違い

トランザクション分離レベルの概要。

元の SQL Server トランザクション分離レベルは、データのコピーが 1 つしか存在せず、クエリがデータにアクセスする前に行などのリソースをロックする必要がある悲観的ロックに基づいています。後に、Read Committed 分離レベルのバリエーションが導入されました。これにより、行バージョンを使用することで、ノンブロッキングアクセスを使用するリーダーとライター間の同時実行性が向上します。さらに、スナップショットと呼ばれる新しい分離レベルも利用できるようになりました。これも行バージョンを使用して、トランザクションの終了まで保持される読み取りデータの共有ロックを回避することで、REPEATABLE READ 分離レベルよりも優れた同時実行性を実現します。

SQL Server とは異なり、Babelfish のすべてのトランザクション分離レベルは楽観的ロック (MVCC) に基づいています。各トランザクションは、基になるデータの現在の状態に関係なく、ステートメントの先頭 (READ COMMITTED) またはトランザクションの先頭 (REPEATABLE READ、SERIALIZABLE) にデータのスナップショットを表示します。したがって、Babelfish での同時トランザクションの実行動作は SQL Server とは異なる場合があります。

例えば、分離レベル SERIALIZABLE のトランザクションのうち、最初に SQL Server でブロックされ、後に成功したトランザクションがあるとします。同じ行を読み取るまたは更新する同時トランザクションとのシリアル化の競合により、Babelfish で失敗する可能性があります。また、複数の同時トランザクションを実行すると、SQL Server と比較して Babelfish の最終結果が異なる場合もあり

ます。分離レベルを使用するアプリケーションは、同時実行シナリオについて徹底的にテストする必要があります。

SQL Server の分離レベル	Babelfish 分離レベル	PostgreSQL 分離レベル	コメント
READ UNCOMMITTED	READ UNCOMMITTED	READ UNCOMMITTED	Read Uncommitted は、Babelfish/PostgreSQL で Read Committed と同じです
READ COMMITTED	READ COMMITTED	READ COMMITTED	SQL Server Read Committed は悲観的ロックベースで、Babelfish Read Committed はスナップショット (MVCC) ベースです。
READ COMMITTED SNAPSHOT	READ COMMITTED	READ COMMITTED	どちらもスナップショット (MVCC) ベースですが、まったく同じではありません。
SNAPSHOT	SNAPSHOT	REPEATABLE READ	まったく同じです。
REPEATABLE READ	REPEATABLE READ	REPEATABLE READ	SQL Server Repeatable Read は悲観的ロックベースで、Babelfish Repeatable Read はスナップショット (MVCC) ベースです。
SERIALIZABLE	SERIALIZABLE	SERIALIZABLE	SQL Server Serializable は悲観的分離

SQL Server の分離レベル	Babelfish 分離レベル	PostgreSQL 分離レベル	コメント
			であり、Babelfish Serializable はスナップショット (MVCC) ベースです。

> **Note**
>
> テーブルヒントは現在サポートされておらず、その動作は Babelfish の定義済みエスケープハッチ `escape_hatch_table_hints` を使用して制御されます。

トランザクション分離レベルのセットアップ

次のコマンドを使用して、トランザクション分離レベルを設定します。

Example

```
SET TRANSACTION ISOLATION LEVEL { READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ |
 SNAPSHOT | SERIALIZABLE }
```

トランザクション分離レベルの有効化または無効化

Babelfish では、トランザクション分離レベル REPEATABLE READ と SERIALIZABLE はデフォルトで無効になっているため、`sp_babelfish_configure` を使用して `babelfishpg_tsql.isolation_level_serializable` または `babelfishpg_tsql.isolation_level_repeatable_read` エスケープハッチを `pg_isolation` に設定して明示的に有効にする必要があります。詳細については、「エスケープハッチ処理時の Babelfish のエラー処理の管理」を参照してください。

以下は、それぞれのエスケープハッチを設定して、現在のセッションで REPEATABLE READ と SERIALIZABLE の使用を有効または無効にする例です。オプションで、現在のセッションとそれ以降のすべての新しいセッションのエスケープハッチを設定する `server` パラメータを含めます。

現在のセッションでのみ SET TRANSACTION ISOLATION LEVEL REPEATABLE READ の使用を有効にする場合。

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_repeatable_read', 'pg_isolation'
```

現在のセッションおよびすべての後続の新しいセッションで SET TRANSACTION ISOLATION LEVEL REPEATABLE READ の使用を有効にする場合。

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_repeatable_read', 'pg_isolation',
 'server'
```

現在のセッションおよび後続の新しいセッションで SET TRANSACTION ISOLATION LEVEL REPEATABLE READ の使用を無効にする場合。

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_repeatable_read', 'off', 'server'
```

現在のセッションでのみ SET TRANSACTION ISOLATION LEVEL SERIALIZABLE の使用を有効にする場合。

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_serializable', 'pg_isolation'
```

現在のセッションおよびすべての後続の新しいセッションで SET TRANSACTION ISOLATION LEVEL SERIALIZABLE の使用を有効にする場合。

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_serializable', 'pg_isolation', 'server'
```

現在のセッションおよび後続の新しいセッションで SET TRANSACTION ISOLATION LEVEL SERIALIZABLE の使用を無効にする場合。

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_serializable', 'off', 'server'
```

Babelfish と SQL Server の分離レベルの違い

以下は、SQL Server と Babelfish が ANSI 分離レベルを実装する方法に関するニュアンスの例です。

> **Note**
>
> - Babelfish では、分離レベル Repeatable Read とスナップショットは同じです。
> - Babelfish では、Read Uncommitted と Read Committed は同じです。

次の例は、以下で説明するすべての例においてベーステーブルを作成する方法を示しています。

```
CREATE TABLE employee (
    id sys.INT NOT NULL PRIMARY KEY,
    name sys.VARCHAR(255)NOT NULL,
    age sys.INT NOT NULL
);
INSERT INTO employee (id, name, age) VALUES (1, 'A', 10);
INSERT INTO employee (id, name, age) VALUES (2, 'B', 20);
INSERT INTO employee (id, name, age) VALUES (3, 'C', 30);
```

トピック

- BABELFISH READ UNCOMMITTED と SQL SERVER READ UNCOMMITTED 分離レベルの比較
- BABELFISH READ COMMITTED と SQL SERVER READ COMMITTED 分離レベルの比較
- BABELFISH READ COMMITTED と SQL SERVER READ COMMITTED スナップショット分離レベルの比較
- BABELFISH REPEATABLE READ と SQL SERVER REPEATABLE READ 分離レベルの比較

- BABELFISH SERIALIZABLE と SQL SERVER SERIALIZABLE 分離レベルの比較

BABELFISH READ UNCOMMITTED と SQL SERVER READ UNCOMMITTED 分離レベルの比較

SQL SERVER の DIRTY READS

トランザクション 1	トランザクション 2	SQL Server Read Uncommitted	Babelfish Read Uncommitted
BEGIN TRANSACTION	BEGIN TRANSACTION		
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;	SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;		
	UPDATE employee SET age=0;	更新に成功。	更新に成功。
	INSERT INTO employee VALUES (4, 'D', 40);	挿入に成功。	挿入に成功。
SELECT * FROM employee;		トランザクション 1 では、トランザクション 2 からのコミットされていない変更が表示されます。	Babelfish での Read Committed と同じです。トランザクション 2 からのコミットされていない変更は、トランザクション 1 には表示されません。
	COMMIT		
SELECT * FROM employee;		トランザクション 2 によってコミットされた変更が表示されます。	トランザクション 2 によってコミットされた変更が表示されます。

BABELFISH READ COMMITTED と SQL SERVER READ COMMITTED 分離レベルの比較

READ - WRITE BLOCKING

トランザクション 1	トランザクション 2	SQL Server Read Committed	Babelfish Read Committed
BEGIN TRANSACTION	BEGIN TRANSACTION		
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;	SET TRANSACTION ISOLATION LEVEL READ COMMITTED;		
SELECT * FROM employee;			
	UPDATE employee SET age=100 WHERE id = 1;	更新に成功。	更新に成功。
UPDATE employee SET age = 0 WHERE age IN (SELECT MAX(age) FROM employee);		トランザクション 2 がコミットされるまでステップがブロックされます。	トランザクション 2 の変更はまだ表示されていません。Updates row with id=3.
	COMMIT	トランザクション 2 は正常にコミットされます。トランザクション 1 はブロック解除され、トランザクション 2 からの更新が表示されます。	トランザクション 2 は正常にコミットされます。
SELECT * FROM employee;		トランザクション 1 は id = 1 で行を更新します。	トランザクション 1 は id = 3 で行を更新します。

BABELFISH READ COMMITTED と SQL SERVER READ COMMITTED スナップショット分離レベルの比較

新しく挿入された行のブロック動作

トランザクション 1	トランザクション 2	SQL Server Read Committed スナップショット	Babelfish Read Committed
BEGIN TRANSACTION	BEGIN TRANSACTION		
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;	SET TRANSACTION ISOLATION LEVEL READ COMMITTED;		
INSERT INTO employee VALUES (4, 'D', 40);			
	UPDATE employee SET age = 99;	トランザクション 1 がコミットされるまでステップがブロックされます。挿入された行はトランザクション 1 によってロックされます。	3 行を更新しました。新しく挿入された行はまだ表示されません。
COMMIT		コミットは成功しました。トランザクション 2 のブロックが解除されました。	コミットは成功しました。
	SELECT * FROM employee;	All 4 rows have age=99.	id = 4 の行は、更新クエリ中にトランザクション 2 に表示されなかったため、保持時間の値は 40 です。

トランザクション 1	トランザクション 2	SQL Server Read Committed スナップショット	Babelfish Read Committed
			その他の行は age=99 に更新されます。

BABELFISH REPEATABLE READ と SQL SERVER REPEATABLE READ 分離レベルの比較

読み取り/書き込みブロック動作

トランザクション 1	トランザクション 2	SQL Server Repeatable Read	Babelfish Repeatable Read
BEGIN TRANSACTION	BEGIN TRANSACTION		
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;	SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;		
SELECT * FROM employee;			
UPDATE employee SET name='A_TXN1' WHERE id=1;			
	SELECT * FROM employee WHERE id != 1;		
	SELECT * FROM employee;	トランザクション 2 はトランザクション 1 がコミットされるまでブロックされます。	トランザクション 2 は正常に進みます。
COMMIT			

トランザクション 1	トランザクション 2	SQL Server Repeatable Read	Babelfish Repeatable Read
	SELECT * FROM employee;	トランザクション 1 からの更新が表示されます。	トランザクション 1 からの更新が表示されません。
COMMIT			
	SELECT * FROM employee;	トランザクション 1 からの更新が表示されます。	トランザクション 1 からの更新が表示されます。

書き込み/書き込みブロック動作

トランザクション 1	トランザクション 2	SQL Server Repeatable Read	Babelfish Repeatable Read
BEGIN TRANSACTION	BEGIN TRANSACTION		
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;	SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;		
UPDATE employee SET name='A_TXN1' WHERE id=1;			
	UPDATE employee SET name='A_TXN2' WHERE id=1;	トランザクション 2 がブロックされました。	トランザクション 2 がブロックされました。
COMMIT		コミットが成功し、トランザクション 2 のブロックが解除されました。	コミットが成功し、トランザクション 2 がエラーで失敗し、同時更新によりアク

トランザクション 1	トランザクション 2	SQL Server Repeatable Read	Babelfish Repeatable Read
			セスをシリアル化できませんでした。
	COMMIT	コミットは成功しました。	トランザクション 2 はすでに中止されています。
	SELECT * FROM employee;	id=1 の行に name='A_TX2' があります。	id=1 の行に name='A_TX1' があります。

PHANTOM READ

トランザクション 1	トランザクション 2	SQL Server Repeatable Read	Babelfish Repeatable Read
BEGIN TRANSACTION	BEGIN TRANSACTION		
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;	SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;		
SELECT * FROM employee;			
	INSERT INTO employee VALUES (4, 'NewRowName', 20);	トランザクション 2 はブロッキングなしで進行します。	トランザクション 2 はブロッキングなしで進行します。
	SELECT * FROM employee;	新しく挿入された行が表示されます。	新しく挿入された行が表示されます。
	COMMIT		

トランザクション 1	トランザクション 2	SQL Server Repeatable Read	Babelfish Repeatable Read
SELECT * FROM employee;		トランザクション 2 によって挿入された新しい行が表示されます。	トランザクション 2 によって挿入された新しい行が表示されません。
COMMIT			
SELECT * FROM employee;		新しく挿入された行が表示されます。	新しく挿入された行が表示されます。

さまざまな最終結果

トランザクション 1	トランザクション 2	SQL Server Repeatable Read	Babelfish Repeatable Read
BEGIN TRANSACTION	BEGIN TRANSACTION		
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;	SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;		
UPDATE employee SET age = 100 WHERE age IN (SELECT MIN(age) FROM employee);		トランザクション 1 は id 1 で行を更新します。	トランザクション 1 は id 1 で行を更新します。
	UPDATE employee SET age = 0 WHERE age IN (SELECT MAX(age) FROM employee);	SELECT ステートメントはトランザクション 1 の UPDATE クエリによってロックされた行の読み取りを試みるため、ト	読み取りがブロックされないため、トランザクション 2 はブロックなしで進行します。SELECT ステートメントが実行され、トランザク

トランザクション 1	トランザクション 2	SQL Server Repeatable Read	Babelfish Repeatable Read
		ランザクション 2 はブロックされます。	ション 1 の変更がまだ表示されないため、id = 3 の行が最終的に更新されます。
	SELECT * FROM employee;	このステップは、トランザクション 1 がコミットされた後に実行されます。id = 1 の行は、前のステップでトランザクション 2 によって更新され、ここに表示されます。	id = 3 の行はトランザクション 2 によって更新されます。
COMMIT		トランザクション 2 のブロックが解除されました。	コミットは成功しました。
	COMMIT		
SELECT * FROM employee;		どちらのトランザクションも、id = 1 の行で更新されます。	トランザクション 1 と 2 によって異なる行が更新されます。

BABELFISH SERIALIZABLE と SQL SERVER SERIALIZABLE 分離レベルの比較

SQL SERVER の範囲ロック

トランザクション 1	トランザクション 2	SQL Server Serializable	Babelfish Serializable
BEGIN TRANSACTION	BEGIN TRANSACTION		

トランザクション 1	トランザクション 2	SQL Server Serializable	Babelfish Serializable
SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;	SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;		
SELECT * FROM employee;			
	INSERT INTO employee VALUES (4, 'D', 35);	トランザクション 2 はトランザクション 1 がコミットされるまでブロックされます。	トランザクション 2 はブロッキングなしで進行します。
	SELECT * FROM employee;		
COMMIT		トランザクション 1 は正常にコミットされます。トランザクション 2 のブロックが解除されました。	トランザクション 1 は正常にコミットされます。
	COMMIT		
SELECT * FROM employee;		新しく挿入された行が表示されます。	新しく挿入された行が表示されます。

さまざまな最終結果

トランザクション 1	トランザクション 2	SQL Server Serializable	Babelfish Serializable
BEGIN TRANSACTION	BEGIN TRANSACTION		

トランザクション 1	トランザクション 2	SQL Server Serializable	Babelfish Serializable
SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;	SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;		
	INSERT INTO employee VALUES (4, 'D', 40);		
UPDATE employee SET age =99 WHERE id = 4;		トランザクション 1 はトランザクション 2 がコミットされるまでブロックされます。	トランザクション 1 はブロッキングなしで進行します。
	COMMIT	トランザクション 2 は正常にコミットされます。トランザクション 1 のブロックが解除されました。	トランザクション 2 は正常にコミットされます。
COMMIT			
SELECT * FROM employee;		新しく挿入された行は、保持時間の値 = 99 で表示されます。	新しく挿入された行は、保持時間の値 = 40 で表示されます。

一意の制約でのテーブルへの挿入

トランザクション 1	トランザクション 2	SQL Server Serializable	Babelfish Serializable
BEGIN TRANSACTION	BEGIN TRANSACTION		

トランザクション 1	トランザクション 2	SQL Server Serializable	Babelfish Serializable
SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;	SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;		
	INSERT INTO employee VALUES (4, 'D', 40);		
INSERT INTO employee VALUES ((SELECT MAX(id)+1 FROM employee), 'E', 50);		トランザクション 1 はトランザクション 2 がコミットされるまでブロックされます。	トランザクション 1 はトランザクション 2 がコミットされるまでブロックされます。
	COMMIT	トランザクション 2 は正常にコミットされます。トランザクション 1 のブロックが解除されました。	トランザクション 2 は正常にコミットされます。エラー重複キー値で中止されたトランザクション 1 は、一意の制約に違反します。
COMMIT		トランザクション 1 は正常にコミットされます。	トランザクション 1 のコミットは失敗し、トランザクション間の読み取り/書き込み依存関係によりアクセスをシリアル化できませんでした。
SELECT * FROM employee;		行 (5, 'E', 50) が挿入されます。	4 行のみ存在します。

Babelfish では、分離レベルでシリアル化可能な同時トランザクションは、これらのトランザクションの実行がそれらのトランザクションの可能なすべてのシリアル (一度に 1 つ) 実行と一致しない場合、シリアル化異常エラーで失敗します。

シリアル化異常

トランザクション 1	トランザクション 2	SQL Server Serializable	Babelfish Serializable
BEGIN TRANSACTION	BEGIN TRANSACTION		
SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;	SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;		
SELECT * FROM employee;			
UPDATE employee SET age=5 WHERE age=10;			
	SELECT * FROM employee;	トランザクション 2 はトランザクション 1 がコミットされるまでブロックされます。	トランザクション 2 はブロッキングなしで進行します。
	UPDATE employee SET age=35 WHERE age=30;		
COMMIT		トランザクション 1 は正常にコミットされます。	トランザクション 1 は最初にコミットされ、正常にコミットできます。

トランザクション 1	トランザクション 2	SQL Server Serializable	Babelfish Serializable
	COMMIT	トランザクション 2 は正常にコミットされます。	トランザクション 2 のコミットはシリアル化エラーで失敗し、トランザクション全体がロールバックされました。トランザクション 2 を再試行します。
SELECT * FROM employee;		両方のトランザクションからの変更が表示されます。	トランザクション 2 はロールバックされました。トランザクション 1 の変更のみが表示されます。

Babelfish では、すべての同時トランザクションが分離レベル SERIALIZABLE で実行されている場合にのみ、シリアル化異常が発生する可能性があります。例えば、上記の例で、代わりにトランザクション 2 を分離レベル REPEATABLE READ に設定するとします。

トランザクション 1	トランザクション 2	SQL Server 分離レベル	Babelfish 分離レベル
BEGIN TRANSACTION	BEGIN TRANSACTION		
SET TRANSACTION ISOLATION LEVEL SERILAIZABLE;	SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;		
SELECT * FROM employee;			

トランザクション 1	トランザクション 2	SQL Server 分離レベル	Babelfish 分離レベル
UPDATE employee SET age=5 WHERE age=10;			
	SELECT * FROM employee;	トランザクション 2 はトランザクション 1 がコミットされるまでブロックされます。	トランザクション 2 はブロッキングなしで進行します。
	UPDATE employee SET age=35 WHERE age=30;		
COMMIT		トランザクション 1 は正常にコミットされます。	トランザクション 1 は正常にコミットされます。
	COMMIT	トランザクション 2 は正常にコミットされます。	トランザクション 2 は正常にコミットされます。
SELECT * FROM employee;		両方のトランザクションからの変更が表示されます。	両方のトランザクションからの変更が表示されます。

実装が制限されている機能

Babelfish の新しいバージョンでは、T-SQL の機能や動作にさらに適した機能が追加されています。ただし、現在の実装ではサポートされていない機能や違いがいくつかあります。以下に、Babelfish と T-SQL の機能の違いに関する情報と、回避策や使用上の注意事項を示します。

Babelfish のバージョン 1.2.0 時点では、以下の機能で実装が制限されています。

- SQL Server カタログ (システムビュー) — カタログ `sys.sysconfigures`、`sys.syscurconfigs`、`sys.configurations` では、単一の読み取り専用設定のみをサポートします。現在、`sp_configure` はサポートされていません。Babelfish によって実装される SQL Server の他のビューの詳細については、「Babelfish システムカタログからの情報の入手」を参照してください。
- 許可の付与— GRANT…TO PUBLIC はサポートされていますが、GRANT..TO PUBLIC WITH GRANT OPTION は現在サポートされていません。
- SQL Server 所有権チェーンと許可メカニズムの制限 — Babelfish では、SQL Server の所有権チェーンはビューに対して機能しますが、ストアドプロシージャでは機能しません。つまり、プロシージャには、呼び出しプロシージャと同じ所有者が所有する他のオブジェクトへの明示的なアクセスを付与する必要があります。SQL Server では、プロシージャに対する呼び出し元の EXECUTE 許可を付与すれば、同じ所有者が所有する他のオブジェクトを呼び出すのに十分です。Babelfish では、呼び出し元にプロシージャによってアクセスされるオブジェクトに対する許可も付与する必要があります。
- 非修飾 (スキーマ名なし) オブジェクト参照の解決 — SQL オブジェクト (プロシージャ、ビュー、関数、トリガー) が、スキーマ名で修飾せずにオブジェクトを参照する場合、SQL Server は、参照が発生する SQL オブジェクトのスキーマ名を使用してオブジェクトのスキーマ名を解決します。現在、Babelfish は、プロシージャを実行するデータベースユーザーのデフォルトスキーマを使用して、これを異なる方法で解決しています。
- デフォルトのスキーマの変更、セッション、および接続 — ユーザーがデフォルトのスキーマを `ALTER USER...WITH DEFAULT SCHEMA` で変更した場合、その変更はそのセッションですぐに有効になります。ただし、同じユーザーに属する現在接続されている他のセッションでは、次のようにタイミングが異なります。
  - SQL Server の場合: 変更は、このユーザーの他のすべての接続ですぐに有効になります。
  - Babelfish の場合: この変更は、新しい接続に対してのみ有効になります。
- ROWVERSION および TIMESTAMP データ型の実装とエスケープハッチ設定 — Babelfish で ROWVERSION と TIMESTAMP データ型がサポートされるようになりました。Babelfish で ROWVERSION または TIMESTAMP を使用するには、エスケープハッチ `babelfishpg_tsql.escape_hatch_rowversion` の設定をデフォルト (strict) から `ignore` に変更する必要があります。ROWVERSION および TIMESTAMP データ型の Babelfish の実装は、意味的に SQL Server とほぼ同じですが、次の例外があります。
  - 組み込みの @@DBTS 関数は、SQL Server と同様に動作しますが、わずかに違います。BabelBabelfish は、`SELECT @@DBTS` の最後に使用された値を返すのではなく、基礎となる PostgreSQL データベースエンジンとそのマルチバージョン同時実行制御 (MVCC) の実装により、新しいタイムスタンプを生成します。

- SQL Server では、挿入または更新されたすべての行に一意の ROWVERSION/TIMESTAMP 値が付与されます。Babelfish では、同じステートメントによって更新される挿入されたすべての行に、同じ ROWVERSION/TIMESTAMP 値が割り当てられます。

  例えば、UPDATE ステートメントまたは INSERT-SELECT ステートメントが複数の行に影響する場合、SQL Server では、影響を受けるすべての行の ROWVERSION/TIMESTAMP 列の値が異なります。Babelfish (PostgreSQL) では、行の値は同じです。
- SQL Server では、SELECT-INTO を使用して新しいテーブルを作成するときに、作成される ROWVERSION/TIMESTAMP 列に明示的な値 (NULL など) をキャストできます。Babelfish で同じことをすると、Babelfish によって、実際の ROWVERSION/TIMESTAMP 値が新しいテーブルの各行に割り当てられます。

ROWVERSION/TIMESTAMP のデータ型のこれらのわずかな違いは、Babelfish で実行されているアプリケーションに悪影響を及ぼさないはずです。

スキーマの作成、所有権、およびアクセス許可 — 非 DBO ユーザーによって所有されるスキーマ内のオブジェクトを (`CREATE SCHEMA schema_name AUTHORIZATION user_name` を使用して) 作成およびアクセスするためのアクセス許可は、次の表に示すように、SQL Server および Babelfish の非 DBO ユーザーでは異なります。

スキーマを所有しているデータベースユーザー (非 DBO) は、次のことができます。	SQL Server	Babelfish
DBO による追加の付与なしでスキーマにオブジェクトを作成しますか?	いいえ	はい
DBO によってスキーマに作成されたオブジェクトに追加の付与なしでアクセスしますか?	はい	いいえ

Babelfish クエリパフォーマンスの向上

クエリヒントと PostgreSQL オプティマイザーを使用すると、Babelfish のクエリ処理を高速化できます。

トピック

- 説明プランを使用して Babelfish クエリのパフォーマンスを向上させる
- T-SQL クエリヒントを使用して Babelfish クエリのパフォーマンスを向上させる

sp_babelfish_volatility プロシージャを使用してクエリのパフォーマンスを向上させることもできます。詳しくは、「sp_babelfish_volatility」を参照してください。

説明プランを使用して Babelfish クエリのパフォーマンスを向上させる

バージョン 2.1.0 以降、Babelfish には、PostgreSQL オプティマイザを透過的に使用して、TDS ポート上で T-SQL クエリの推定および実際のクエリプランを生成する 2 つの関数が含まれています。これらの関数は、SQL Server データベースで SET STATISTICS PROFILE または SET SHOWPLAN_ALL を使用して、実行速度が遅いクエリを識別して改善するのと同様です。

> **Note**
>
> 関数、制御フロー、およびカーソルからのクエリプランの取得は、現在サポートされていません。

この表では、SQL Server、Babelfish、および PostgreSQL のクエリプランの説明関数を比較できます。

SQL Server	Babelfish	PostgreSQL
SHOWPLAN_ALL	BABELFISH_SHOWPLAN_ALL	EXPLAIN
STATISTICS PROFILE	BABELFISH_STATISTICS PROFILE	EXPLAIN ANALYZE
SQL Server オプティマイザを使用する	PostgreSQL オプティマイザを使用する	PostgreSQL オプティマイザを使用する
SQL Server の入力および出力形式	SQL Server の入力および PostgreSQL 出力形式	PostgreSQL の入力および出力形式
セッションに設定	セッションに設定	特定のステートメントに適用する

SQL Server	Babelfish	PostgreSQL
以下をサポートしています。 • SELECT • INSERT • UPDATE • DELETE • CURSOR • CREATE • EXECUTE • 制御フロー（CASE、WHILE-BREAK-CONTINUE、WAITFOR、BEGIN-END、IF-ELSEなど）を含む EXEC と関数	以下をサポートしています。 • SELECT • INSERT • UPDATE • DELETE • CREATE • EXECUTE • EXEC • RAISEERROR • THROW • PRINT • 使用	以下をサポートしています。 • SELECT • INSERT • UPDATE • DELETE • CURSOR • CREATE • EXECUTE

Babelfish 関数を次のように使用します。

- BABELFISH_SHOWPLAN_ALL [ON|OFF] — ON に設定すると、推定されたクエリ実行プランが生成されます。この関数は PostgreSQL `EXPLAIN` コマンドの動作を実装します。このコマンドを使用して、特定のクエリの説明プランを取得します。
- SET BABELFISH_STATISTICS PROFILE [ON|OFF] — 実際のクエリ実行プランの場合は ON に設定します。この関数は PostgreSQL の `EXPLAIN ANALYZE` コマンドの動作を実装します。

PostgreSQL `EXPLAIN` および `EXPLAIN ANALYZE` の詳細については、PostgreSQL ドキュメントの EXPLAIN を参照してください。

> **Note**
>
> バージョン 2.2.0 以降、`escape_hatch_showplan_all` パラメータを ignore に設定して、`SHOWPLAN_ALL` と `STATISTICS PROFILE` SET コマンドの SQL Server 構文で BABELFISH_ プレフィックスが使用されないようにできます。

例えば、次のコマンドシーケンスは、クエリプランをオンにし、クエリを実行せずに SELECT ステートメントの推定クエリ実行プランを返します。この例では、sqlcmd コマンドラインツールを使って SQL Server のサンプル northwind データベースを使用し、TDS ポートをクエリします。

```
1> SET BABELFISH_SHOWPLAN_ALL ON
2> GO
1> SELECT t.territoryid, e.employeeid FROM
2> dbo.employeeterritories e, dbo.territories t
3> WHERE e.territoryid=e.territoryid ORDER BY t.territoryid;
4> GO

QUERY PLAN


------------------------------------------------------------------------------


Query Text: SELECT t.territoryid, e.employeeid FROM
dbo.employeeterritories e, dbo.territories t
WHERE e.territoryid=e.territoryid ORDER BY t.territoryid
Sort  (cost=6231.74..6399.22 rows=66992 width=10)
  Sort Key: t.territoryid NULLS FIRST
  ->  Nested Loop  (cost=0.00..861.76 rows=66992 width=10)
        ->  Seq Scan on employeeterritories e  (cost=0.00..22.70 rows=1264 width=4)
              Filter: ((territoryid)::"varchar" IS NOT NULL)
        ->  Materialize  (cost=0.00..1.79 rows=53 width=6)
              ->  Seq Scan on territories t  (cost=0.00..1.53 rows=53 width=6)
```

クエリの確認と調整が完了したら、次に示すように関数をオフにします。

```
1> SET BABELFISH_SHOWPLAN_ALL OFF
```

BABELFISH_STATISTICS PROFILE を ON に設定すると、実行された各クエリは通常の結果セットと、それに続く実際のクエリ実行プランを示す追加の結果セットを返します。Babelfish は、SELECT ステートメントを呼び出すときに最速の結果セットを提供するクエリプランを生成します。

```
1> SET BABELFISH_STATISTICS PROFILE ON
1>
2> GO
1> SELECT e.employeeid, t.territoryid FROM
```

```
2> dbo.employeeterritories e, dbo.territories t
3> WHERE t.territoryid=e.territoryid ORDER BY t.territoryid;
4> GO
```

結果セットとクエリプランが返されます (この例に表示されているのはクエリプランのみです)。

```
QUERY PLAN


------------------------------------------------------------------------------
Query Text: SELECT e.employeeid, t.territoryid FROM
dbo.employeeterritories e, dbo.territories t
WHERE t.territoryid=e.territoryid ORDER BY t.territoryid
Sort  (cost=42.44..43.28 rows=337 width=10)
  Sort Key: t.territoryid NULLS FIRST


  ->  Hash Join  (cost=2.19..28.29 rows=337 width=10)
       Hash Cond: ((e.territoryid)::"varchar" = (t.territoryid)::"varchar")
        ->  Seq Scan on employeeterritories e  (cost=0.00..22.70 rows=1270 width=36)
        ->  Hash  (cost=1.53..1.53 rows=53 width=6)
             ->  Seq Scan on territories t  (cost=0.00..1.53 rows=53 width=6)
```

クエリと PostgreSQL オプティマイザによって返される結果を分析する方法の詳細については、「explain.depesz.com」を参照してください。PostgreSQL EXPLAIN および EXPLAIN ANALYZE の詳細については、PostgreSQL ドキュメントの「EXPLAIN」を参照してください。

Babelfish 説明オプションを制御するパラメータ

次の表に示すパラメータを使用して、クエリプランに表示される情報のタイプを制御できます。

パラメータ	説明
babelfishpg_tsql.explain_buffers	オプティマイザのバッファ使用状況情報をオン (またはオフ) にするブール値。(デフォルト: off) (許容値: off、on)
babelfishpg_tsql.explain_costs	オプティマイザの推定起動および総コスト情報をオン (またはオフ) にするブール値。(デフォルト: on) (許容値: off、on)

パラメータ	説明
babelfishpg_tsql.explain_format	EXPLAIN プランの出力形式を指定します。(デフォルト: text) (許容値: text, xml, json, yaml)
babelfishpg_tsql.explain_settings	構成パラメータに関する情報を EXPLAIN プランの出力に含めることをオン (またはオフ) するブール値。(デフォルト: off) (許容値: off、on)
babelfishpg_tsql.explain_summary	クエリプランの後の合計時間などのサマリー情報をオン (またはオフ) にするブール値。(デフォルト: on) (許容値: off、on)
babelfishpg_tsql.explain_timing	出力の各ノードでの実際の起動時間と滞在時間をオン (またはオフ) にするブール値。(デフォルト: on) (許容値: off、on)
babelfishpg_tsql.explain_verbose	説明プランの最も詳細なバージョンをオン (またはオフ) にするブール値。(デフォルト: off) (許容値: off、on)
babelfishpg_tsql.explain_wal	説明プランの一部として WAL レコード情報の生成をオン (またはオフ) にするブール値。(デフォルト: off) (許容値: off、on)

PostgreSQL クライアントまたは SQL Server クライアントを使用して、システム上の BabelFish 関連パラメータの値を確認できます。次のコマンドを実行して、現在のパラメータ値を取得します。

```
1> execute sp_babelfish_configure '%explain%';
2> GO
```

次の出力では、この特定の Babelfish DB クラスターのすべての設定がデフォルト値になっていることがわかります。この例ではすべての出力が表示されているわけではありません。

```
            name                    setting                       short_desc
---------------------------------- --------
 ------------------------------------------------------------
babelfishpg_tsql.explain_buffers   off      Include information on buffer usage
```

```
babelfishpg_tsql.explain_costs     on       Include information on estimated startup
 and total cost
babelfishpg_tsql.explain_format    text     Specify the output format, which can be
 TEXT, XML, JSON, or YAML
babelfishpg_tsql.explain_settings  off      Include information on configuration
 parameters
babelfishpg_tsql.explain_summary   on       Include summary information (e.g.,totaled
 timing information) after the query plan
babelfishpg_tsql.explain_timing    on       Include actual startup time and time spent
 in each node in the output
babelfishpg_tsql.explain_verbose   off      Display additional information regarding
 the plan
babelfishpg_tsql.explain_wal       off      Include information on WAL record
 generation

(8 rows affected)
```

次の例に示すように、sp_babelfish_configure を使用して、これらのパラメータの設定を変更できます。

```
1> execute sp_babelfish_configure 'explain_verbose', 'on';
2> GO
```

クラスター全体のレベルで設定を永続化するには、次の例に示されているように、 キーワード server を含めます。

```
1> execute sp_babelfish_configure 'explain_verbose', 'on', 'server';
2> GO
```

T-SQL クエリヒントを使用して Babelfish クエリのパフォーマンスを向上させる

バージョン 2.3.0 以降では、Babelfish は pg_hint_plan を使用したクエリヒントの使用をサポートしています。Aurora PostgreSQL では、pg_hint_plan はデフォルトでインストールされます。PostgreSQL 拡張機能 pg_hint_plan の詳細については、「https://github.com/ossc-db/pg_hint_plan」を参照してください。Aurora PostgreSQL でサポートされているこの拡張機能の詳細については、「Aurora PostgreSQL リリースノート」の「Amazon Aurora PostgreSQL の拡張機能バージョン」を参照してください。

クエリオプティマイザは、SQL ステートメントの最適な実行プランを見つけるように設計されています。プランを選択する際、クエリオプティマイザはエンジンのコストモデルと列およびテーブル統計の両方を考慮します。ただし、提案されるプランはデータセットのニーズを満たさない場合が

あります。したがって、クエリヒントは、パフォーマンスの問題に対処して、実行プランを改善します。query hint は SQL 標準に追加された構文であり、データベースエンジンにクエリの実行方法を指示します。例えば、ヒントは、シーケンシャルスキャンを実行して、クエリオプティマイザが選択したプランをオーバーライドするようにエンジンに指示する場合があります。

Babelfish で T-SQL クエリヒントを有効にする

現在、Babelfish はデフォルトですべての T-SQL ヒントを無視します。T-SQL ヒントを適用するには、enable_pg_hint 値を ON にして、コマンド sp_babelfish_configure を実行します。

```
EXECUTE sp_babelfish_configure 'enable_pg_hint', 'on' [, 'server']
```

クラスタ全体のレベルで設定を永続化するには、server キーワードを含めます。現在のセッションについてのみ設定する場合は、server を使用しません。

enable_pg_hint がオンになると、Babelfish は以下の T-SQL ヒントを適用します。

- INDEX ヒント
- JOIN ヒント
- FORCE ORDER ヒント
- MAXDOP ヒント

例えば、次のコマンドシーケンスは pg_hint_plan をオンにします。

```
1> CREATE TABLE t1 (a1 INT PRIMARY KEY, b1 INT);
2> CREATE TABLE t2 (a2 INT PRIMARY KEY, b2 INT);
3> GO
1> EXECUTE sp_babelfish_configure 'enable_pg_hint', 'on';
2> GO
1> SET BABELFISH_SHOWPLAN_ALL ON;
2> GO
1> SELECT * FROM t1 JOIN t2 ON t1.a1 = t2.a2; --NO HINTS (HASH JOIN)
2> GO
```

SELECT ステートメントにはヒントは適用されません。ヒントのないクエリプランが返されます。

```
QUERY PLAN
```

```
-------------------------------------------------------------------------------
Query Text: SELECT * FROM t1 JOIN t2 ON t1.a1 = t2.a2
Hash Join (cost=60.85..99.39 rows=2260 width=16)
 Hash Cond: (t1.a1 = t2.a2)
 -> Seq Scan on t1 (cost=0.00..32.60 rows=2260 width=8)
 -> Hash (cost=32.60..32.60 rows=2260 width=8)
 -> Seq Scan on t2 (cost=0.00..32.60 rows=2260 width=8)
```

```
1> SELECT * FROM t1 INNER MERGE JOIN t2 ON t1.a1 = t2.a2;
2> GO
```

SELECT ステートメントにクエリヒントが適用されます。次の出力は、マージ結合を含むクエリプランが返されることを示しています。

```
QUERY PLAN


-------------------------------------------------------------------------------
Query Text: SELECT/*+ MergeJoin(t1 t2) Leading(t1 t2)*/ * FROM t1 INNER JOIN t2 ON
 t1.a1 = t2.a2
Merge Join (cost=0.31..190.01 rows=2260 width=16)
 Merge Cond: (t1.a1 = t2.a2)
 -> Index Scan using t1_pkey on t1 (cost=0.15..78.06 rows=2260 width=8)
 -> Index Scan using t2_pkey on t2 (cost=0.15..78.06 rows=2260 width=8)
```

```
1> SET BABELFISH_SHOWPLAN_ALL OFF;
2> GO
```

制限事項

クエリヒントを使用する場合は、以下の制限事項を考慮してください。

- enable_pg_hint がオンになる前にクエリプランがキャッシュされた場合、ヒントは同じセッションに適用されません。新しいセッションに適用されます。
- スキーマ名が明示的に指定された場合、ヒントは適用できません。回避策としてテーブルエイリアスを使用できます。
- クエリヒントをビューやサブクエリに適用することはできません。

- ヒントは、JOIN を含む UPDATE/DELETE ステートメントでは機能しません。
- 存在しないインデックスまたはテーブルについてのインデックスヒントは無視されます。
- FORCE ORDER ヒントは、HASH JOIN と非 ANSI JOIN では機能しません。

Babelfish で Aurora PostgreSQL 拡張機能を使用する

Aurora PostgreSQL には、他の AWS のサービスとの連携のための拡張機能が用意されています。これらは、データのインポートやエクスポートに DB クラスターで Amazon S3 を使用するなど、さまざまなユースケースをサポートするオプションの拡張機能です。

- Amazon S3 バケットから Babelfish for Aurora PostgreSQL DB クラスターにデータをインポートするには、aws_s3 Aurora PostgreSQL 拡張機能をセットアップします。この拡張機能を使用すると、Aurora PostgreSQL DB クラスターから Amazon S3 バケットにデータをエクスポートすることもできます。
- AWS Lambda はサーバーのプロビジョニングや管理をする必要がなく、コードを実行できるコンピューティングサービスです。DB インスタンスのイベント通知を処理するときなどに Lambda 関数を使用できます。Lambda の詳細については、AWS Lambda デベロッパーガイドの「AWS Lambda とは」を参照してください。Babelfish DB クラスターから Lambda 関数を呼び出すには、aws_lambda Aurora PostgreSQL 拡張機能をセットアップします。

Babelfish クラスターにこれらの拡張機能をセットアップするには、まず内部 Babelfish ユーザーに拡張機能をロードするアクセス許可を付与する必要があります。アクセス許可を付与した後、Aurora PostgreSQL 拡張機能をロードできます。

Babelfish DB クラスターで Aurora PostgreSQL 拡張機能を有効にする

aws_s3 または aws_lambda 拡張機能をロードする前に、Babelfish DB クラスターに必要な権限を付与します。

次の手順では、psql PostgreSQL コマンドラインツールを使用して DB クラスターに接続します。詳細については、「psql を使用したクラスターへの接続」を参照してください。pgAdmin を使用することもできます。詳細については、「pgAdmin を使用した DB クラスターへの接続」を参照してください。

この手順では、aws_s3 と aws_lambda の両方を順次ロードします。これらの拡張機能のうちの 1 つだけを使用する場合は、両方をロードする必要はありません。どちらにも aws_commons 拡張機能が必要で、これは出力に示すようにデフォルトでロードされます。

Aurora PostgreSQL 拡張機能の権限で Babelfish DB クラスターをセットアップするには

1. Babelfish DB クラスターに接続します。Babelfish DB クラスターの作成時に指定した「マスター」ユーザー (-U) の名前を使用します。デフォルト (postgres) を例で示します。

   Linux、macOS、Unix の場合:

   ```
   psql -h your-Babelfish.cluster.444455556666-us-east-1.rds.amazonaws.com \
   -U postgres \
   -d babelfish_db \
   -p 5432
   ```

   Windows の場合:

   ```
   psql -h your-Babelfish.cluster.444455556666-us-east-1.rds.amazonaws.com ^
   -U postgres ^
   -d babelfish_db ^
   -p 5432
   ```

   コマンドは、ユーザー名 (-U) のパスワードを入力するプロンプトを表示します。

   ```
   Password:
   ```

   DB クラスターのユーザー名 (-U) のパスワードを入力します。接続が成功した場合は、次のような出力が表示されます。

   ```
   psql (13.4)
   SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
    compression: off)
   Type "help" for help.

   postgres=>
   ```

2. 内部 Babelfish ユーザーに拡張機能を作成およびロードする権限を付与します。

   ```
   babelfish_db=> GRANT rds_superuser TO master_dbo;
   GRANT ROLE
   ```

3. aws_s3 拡張機能を作成してロードします。aws_commons 拡張機能が必要で、aws_s3 がインストールされるときに自動的にインストールされます。

```
babelfish_db=> create extension aws_s3 cascade;
NOTICE:  installing required extension "aws_commons"
CREATE EXTENSION
```

4. aws_lambda 拡張機能を作成してロードします。

```
babelfish_db=> create extension aws_lambda cascade;
CREATE EXTENSION
babelfish_db=>
```

Amazon S3 での Babelfish の使用

Babelfish DB クラスターで使用する Amazon S3 バケットをまだ持っていない場合は、作成できます。使用するすべての Amazon S3 バケットに対して、アクセスを提供します。

Amazon S3 バケットを使用してデータをインポートまたはエクスポートする前に、次の 1 回限りの手順を完了してください。

Amazon S3 バケットへの Babelfish DB インスタンスのアクセスを設定するには

1. 必要に応じて、Babelfish インスタンス用の Amazon S3 バケットを作成します。これを行うには、「Amazon Simple Storage Service ユーザーガイド」の「最初の S3 バケットを作成する」の手順に従います。
2. ファイルを Amazon S3 バケットにアップロードします。これを行うには、「Amazon Simple Storage Service ユーザーガイド」の「バケットにオブジェクトをアップロードする」の手順に従います。
3. 必要に応じてアクセス許可を次のように設定します。

   - Amazon S3 からデータをインポートするには、Babelfish DB クラスターにバケットへのアクセス許可が必要です。AWS Identity and Access Management (IAM) ロールを使用し、クラスターのそのロールに IAM ポリシーをアタッチすることをお勧めします。これを行うには、「IAM ロールを使用した Amazon S3 バケットへのアクセス」の手順を実行します。
   - Babelfish DB クラスターからデータをエクスポートするには、クラスターに Amazon S3 バケットへのアクセス権が付与されている必要があります。インポートの場合と同様に、IAM ロールとポリシーを使用することをお勧めします。これを行うには、「Amazon S3 バケットへのアクセスを設定する」の手順を実行します。

aws_s3 拡張機能がある Amazon S3 を Babelfish DB クラスターで使用できるようになりました。

Amazon S3 から Babelfish にデータをインポートし、Babelfish データを Amazon S3 にエクスポートするには

1. Babelfish DB クラスターで aws_s3 拡張機能を使用します。

   その場合、PostgreSQL のコンテキスト内に存在するテーブルを参照するようにしてください。つまり、[database].[schema].[tableA] という名前の Babelfish テーブルにインポートする場合、そのテーブルを aws_s3 関数の database_schema_tableA として参照してください。

   - データをインポートする aws_s3 関数の使用例については「Amazon S3 から Aurora PostgreSQL DB クラスターにデータをインポートする」を参照してください。
   - データをエクスポートする aws_s3 関数の使用例については「aws_s3.query_export_to_s3 関数を使用したクエリデータのエクスポート」を参照してください。
2. aws_s3 拡張機能および Amazon S3 使用時には、次の表に示すように、PostgreSQL のネーミングを使用して Babelfish テーブルを参照するようにしてください。

Babelfish テーブル	Aurora PostgreSQL テーブル
database.*schema*.*table*	*database*_*schema*_*table*

Aurora PostgreSQL での Amazon S3 の使用の詳細については、「Amazon S3 から Aurora PostgreSQL DB クラスターにデータをインポートする」および「Aurora PostgreSQL DB クラスターから Amazon S3 へのデータのエクスポート」を参照してください。

AWS Lambda での Babelfish の使用

aws_lambda 拡張機能が Babelfish DB クラスターにロードされても、Lambda 関数の呼び出しができるようにするには、この手順に従って DB クラスターへの Lambda アクセスを許可する必要があります。

Babelfish DB クラスターが Lambda で動作するようにアクセスを設定するには

この手順では、AWS CLI を使用して、IAM ポリシーとロールを作成し、これらを Babelfish DB クラスターに関連付けます。

1. Babelfish DB クラスターから Lambda へのアクセスを許可する IAM ポリシーを作成します。

```
aws iam create-policy  --policy-name rds-lambda-policy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Sid": "AllowAccessToExampleFunction",
        "Effect": "Allow",
        "Action": "lambda:InvokeFunction",
        "Resource": "arn:aws:lambda:aws-region:444455556666:function:my-function"
        }
    ]
}'
```

2. 実行時にポリシーが引き受けることができる IAM ロールを作成します。

```
aws iam create-role  --role-name rds-lambda-role --assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Principal": {
            "Service": "rds.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
        }
    ]
}'
```

3. ロールへのポリシーの付与

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::444455556666:policy/rds-lambda-policy \
    --role-name rds-lambda-role --region aws-region
```

4. ロールを Babelfish DB クラスターにアタッチします。

```
aws rds add-role-to-db-cluster \
       --db-cluster-identifier my-cluster-name \
       --feature-name Lambda \
       --role-arn  arn:aws:iam::444455556666:role/rds-lambda-role   \
       --region aws-region
```

これらのタスクの完了後、Lambda 関数を呼び出すことができます。AWS Lambda での Aurora PostgreSQL DB クラスターへの AWS Lambda の設定の詳細と例については、「ステップ 2: Aurora PostgreSQL DB クラスターおよび AWS Lambda のために IAM を設定する」を参照してください。

Babelfish DB クラスターから Lambda 関数を呼び出すには

AWS Lambda は、Java、Node.js、Python、Ruby、および他の言語で書かれた関数をサポートしています。関数が呼び出されたときにテキストを返す場合は、Babelfish DB クラスターから呼び出すことができます。次の例は、挨拶を返すプレースホルダー Python 関数です。

```
lambda_function.py
import json
def lambda_handler(event, context):
    #TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
```

現在、Babelfish は JSON をサポートしていません。関数が JSON を返す場合、ラッパーを使用して JSON を処理します。例えば、前に示した `lambda_function.py` は、Lambda に `my-function` として保存されています。

1. `psql` クライアント (または pgAdmin クライアント) を使用して Babelfish DB クラスターに接続します。詳細については、「psql を使用したクラスターへの接続」を参照してください。
2. ラッパーを作成します。この例では、PostgreSQL の手続き言語を SQL、PL/pgSQL に使用します。詳細については、「PL/pgSQL – SQL 手続き言語」を参照してください。

```
create or replace function master_dbo.lambda_wrapper()
returns text
language plpgsql
as
$$
declare
   r_status_code integer;
   r_payload text;
begin
   SELECT payload INTO r_payload
     FROM aws_lambda.invoke(  aws_commons.create_lambda_function_arn('my-function',
 'us-east-1')
                            ,'{"body": "Hello from Postgres!"}'::json );
   return r_payload ;
```

```
end;
$$;
```

この関数は、Babelfish TDS ポート (1433) または PostgreSQL ポート (5433) から実行できるようになりました。

a. PostgreSQL ポートからこの関数を起動する (呼び出す) には、次の操作を実行します。

```
SELECT * from aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-
function', 'us-east-1'), '{"body": "Hello from Postgres!"}'::json );
```

出力は次の例のようになります。

```
status_code |                          payload                          |
 executed_version | log_result
-------------+-----------------------------------------------------------
+------------------+------------
         200 | {"statusCode": 200, "body": "\"Hello from Lambda!\""} | $LATEST
         |
(1 row)
```

b. TDS ポートからこの関数を起動する (呼び出す) には、SQL Server `sqlcmd` コマンドラインクライアントを使用してポートに接続します。詳細については、「SQL Server クライアントを使用した DB クラスターへの接続」を参照してください。接続したら、以下を実行します。

```
1> select lambda_wrapper();
2> go
```

このコマンドにより、以下のような出力が返されます。

```
{"statusCode": 200, "body": "\"Hello from Lambda!\""}
```

Aurora PostgreSQL での Lambda の使用の詳細については、「Aurora PostgreSQL DB クラスターから AWS Lambda 関数を呼び出す」を参照してください。Lambda 関数の操作の詳細については、AWS Lambda デベロッパーガイドの「Lambda の開始方法」を参照してください。

Babelfish での pg_stat_statements の使用

Babelfish for Aurora PostgreSQL は 3.3.0 からの pg_stat_statements 拡張機能をサポートしています。詳細については、pg_stat_statements を参照してください。

Aurora PostgreSQL でサポートされているこの拡張機能の詳細については、「拡張機能バージョン」を参照してください。

pg_stat_statements 拡張機能の作成

pg_stat_statements を有効にするには、クエリ ID 計算を有効にする必要があります。compute_query_id がパラメータグループで on または auto に設定されている場合、これは自動的に実行されます。compute_query_id パラメータのデフォルト値は auto です。また、この機能をオンにするには、この拡張機能も作成する必要があります。T-SQL エンドポイントから拡張機能をインストールするには、次のコマンドを使用します。

```
1>EXEC sp_execute_postgresql 'CREATE EXTENSION pg_stat_statements WITH SCHEMA sys';
```

次のクエリを使用してクエリの統計にアクセスできます。

```
postgres=>select * from pg_stat_statements;
```

> **Note**
>
> インストール中に拡張機能のスキーマ名を指定しない場合、デフォルトではパブリックスキーマに作成されます。これにアクセスするには、次に示すように、スキーマ修飾子付きの角括弧を使用する必要があります。

```
postgres=>select * from [public].pg_stat_statements;
```

PSQL エンドポイントから拡張機能を作成することもできます。

拡張機能の認証

デフォルトでは、T-SQL データベース内で実行されたクエリの統計を認証なしで表示できます。

他のユーザーが作成したクエリ統計にアクセスするには、pg_read_all_stats PostgreSQL ロールが必要です。以下の手順に従って GRANT pg_read_all_stats コマンドを作成します。

1. T-SQL では、内部 PG ロール名を返す以下のクエリを使用します。

```
SELECT rolname FROM pg_roles WHERE oid = USER_ID();
```

2. rds_superuser 権限で次のコマンドを使用して、Babelfish for Aurora PostgreSQL データベースに接続します。

```
GRANT pg_read_all_stats TO <rolname_from_above_query>
```

例

T-SQL エンドポイントから：

```
1>SELECT rolname FROM pg_roles WHERE oid = USER_ID();
2>go
```

```


rolname
-------
master_dbo
(1 rows affected)
```

PSQL エンドポイントから：

```
babelfish_db=# grant pg_read_all_stats to master_dbo;
```

```
GRANT ROLE
```

pg_stat_statements ビューを使用してクエリ統計にアクセスできます。

```
1>create table t1(cola int);
2>go
1>insert into t1 values (1),(2),(3);
2>go
```

```
(3 rows affected)
```

```
1>select userid, dbid, queryid, query from pg_stat_statements;
2>go
```

```
userid dbid queryid             query
------ ---- -------             -----
37503 34582 6487973085327558478 select * from t1
37503 34582 6284378402749466286 SET QUOTED_IDENTIFIER OFF
37503 34582 2864302298511657420 insert into t1 values ($1),($2),($3)
10    34582 NULL                <insufficient privilege>
37503 34582 5615368793313871642 SET TEXTSIZE 4096
37503 34582 639400815330803392  create table t1(cola int)
(6 rows affected)
```

クエリの統計のリセット

pg_stat_statements_reset() を使用して、これまで pg_stat_statements によって収集された統計をリセットできます。詳細については、pg_stat_statements を参照してください。現在、PSQL エンドポイントでのみサポートされています。rds_superuser 権限で、次のコマンドを使用して Babelfish for Aurora PostgreSQL に接続します。

```
SELECT pg_stat_statements_reset();
```

制限事項

- 現在、pg_stat_statements() は T-SQL エンドポイントではサポートされていません。pg_stat_statements ビューは、統計を収集するのに推奨される方法です。
- 一部のクエリは、Aurora PostgreSQL エンジンによって実装された T-SQL パーサーによって書き直される可能性があります。pg_stat_statements ビューには、元のクエリではなく、書き直されたクエリが表示されます。

  例

```
select next value for [dbo].[newCounter];
```

上記のクエリは pg_stat_statements ビューで次のように書き直されます。

```
select nextval($1);
```

- ステートメントの実行フローに基づき、一部のクエリは pg_stat_statements によって追跡されず、ビューに表示されない場合があります。これには、`use dbname`、`goto`、`print`、`raise error`、`set`、`throw`、`declare cursor` のステートメントが含まれます。
- CREATE LOGIN ステートメントと ALTER LOGIN ステートメントでは、クエリとクエリ ID は表示されません。権限が不十分だと表示されます。
- `pg_stat_statements` ビューには常に以下の 2 つのエントリが含まれます。これらは `sqlcmd` クライアントによって内部的に実行されるからです。
  - SET QUOTED_IDENTIFIER OFF
  - SET TEXTSIZE 4,096

Babelfish での pgvector の使用

オープンソースの拡張機能である pgvector を使用すると、Postgres データベース内で同様のデータを直接検索できます。Babelfish は、バージョン 15.6 および 16.2 以降、この拡張機能をサポートするようになりました。詳細については、「pgvector Open source Documentation」を参照してください。

前提条件

pgvector 機能を有効にするには、次のいずれかの方法を使用して、拡張機能を sys スキーマにインストールします。

- sqlcmd クライアントで次のコマンドを実行します。

```
exec sys.sp_execute_postgresql 'CREATE EXTENSION vector WITH SCHEMA sys';
```

- psql クライアントで次のコマンドを実行して、`babelfish_db` に接続して実行します。

```
CREATE EXTENSION vector WITH SCHEMA sys;
```

> **Note**
>
> pgvector 拡張機能をインストールした後、ベクトルデータ型は、確立した新しいデータベース接続でのみ使用できます。既存の接続では、新しいデータ型は認識されません。

サポートされている機能

Babelfish は T-SQL 機能を拡張して、以下をサポートします。

- 保存

  Babelfish はベクトルデータ型互換構文をサポートするようになり、T-SQL の互換性が強化されました。pgvector を使用したデータの保存の詳細については、「Storing」を参照してください。
- クエリ

  Babelfish は T-SQL 式のサポートを拡張して、ベクトル類似度演算子をサポートしています。ただし、他のすべてのクエリでは、標準の T-SQL 構文が引き続き必要です。

  > **Note**
  >
  > T-SQL は配列型をサポートしておらず、データベースドライバーにはそれらを処理するインターフェイスがありません。この問題を避けるため、Babelfish はテキスト文字列 (varchar/nvarchar) を使用してベクトルデータを保存します。例えば、ベクトル値 [1,2,3] を要求すると、Babelfish は文字列「[1,2,3]」をレスポンスとして返します。この文字列は、必要に応じてアプリケーションレベルで解析および分割できます。

  pgvector を使用したデータのクエリの詳細については、「Querying」を参照してください。
- インデックス作成

  T-SQL `Create Index` は `USING INDEX_METHOD` 構文をサポートするようになりました。インデックスの作成時に、特定の列で使用する類似度検索演算子を定義できるようになりました。

  また、文法は必要な列 (column_name_list_with_order_for_vector 文法を参照) でのベクトル類似度オペレーションをサポートするように拡張されています。

```
CREATE [UNIQUE] [clustered] [COLUMNSTORE] INDEX <index_name> ON <table_name> [USING
 vector_index_method] (<column_name_list_with_order_for_vector>)
```

```
Where column_name_list_with_order_for_vector is:
    <column_name> [ASC | DESC] [VECTOR_COSINE_OPS | VECTOR_IP_OPS | VECTOR_L2_OPS]
 (COMMA simple_column_name [ASC | DESC] [VECTOR_COSINE_OPS | VECTOR_IP_OPS |
 VECTOR_L2_OPS])
```

pgvector を使用したデータのインデックス作成の詳細については、「Indexing」を参照してください。

- パフォーマンス
  - SET BABELFISH_STATISTICS PROFILE ON を使用して、T-SQL エンドポイントからクエリプランをデバッグします。
  - T-SQL でサポートされている set_config 関数を使用して max_parallel_workers_get_gather を増やします。
  - 近似検索に IVFFlat を使用します。詳細については、「IVFFlat」を参照してください。

  pgvector のパフォーマンスを向上させるには、「Performance」を参照してください。

制限事項

- Babelfish は、ハイブリッド検索の全文検索をサポートしていません。詳細については、「Hybrid Search」を参照してください。
- Babelfish は現在、インデックスの再作成機能をサポートしていません。ただし、PostgreSQL エンドポイントを使用してインデックスを再作成することはできます。詳細については、「Vacuuming」を参照してください。

Babelfish での Amazon Aurora 機械学習の使用

Amazon Aurora の機械学習と統合することで、Babelfish for Aurora PostgreSQL DB クラスターの機能を拡張できます。このシームレスな統合により、Amazon Comprehend、Amazon SageMaker、Amazon Bedrock などのさまざまな強力なサービスにアクセスできます。それぞれのサービスは、個別の機械学習ニーズに対応するようにカスタマイズされています。

Babelfish のユーザーは、Aurora の機械学習を使用する際に T-SQL 構文とセマンティクスに関する既存の知識を使用できます。Aurora PostgreSQL の AWS ドキュメントに記載されている手順に従ってください。詳細については、「Aurora PostgreSQL で Amazon Aurora 機械学習を使用する」を参照してください。

前提条件

- Babelfish for Aurora PostgreSQL DB クラスターを設定して Aurora 機械学習の使用を開始する前に、関連する要件と前提条件を確認してください。詳細については、「Aurora 機械学習 を Aurora PostgreSQL で使用するための要件」を参照してください。
- Postgres エンドポイントまたは sp_execute_postgresql ストアプロシージャを使用して、aws_ml 拡張機能をインストールします。

```
exec sys.sp_execute_postgresql 'Create Extension aws_ml'
```

> **Note**
>
> 現在、Babelfish は Babelfish での sp_execute_postgresql を使用したカスケードオペレーションをサポートしていません。aws_ml は aws_commons に依存するため、Postgres エンドポイントを使用して個別にインストールする必要があります。

```
create extension aws_common;
```

aws_ml 関数を使用した T-SQL 構文とセマンティクスの処理

次の例では、T-SQL 構文とセマンティクスが Amazon ML サービスにどのように適用されるかを説明します。

Example : aws_bedrock.invoke_model – Amazon Bedrock 関数を使用したシンプルなクエリ

```
aws_bedrock.invoke_model(
   model_id      varchar,
   content_type  text,
   accept_type   text,
   model_input   text)
Returns Varchar(MAX)
```

次の例は、invoke_model を使用して Bedrock の Anthropic Claude 2 モデルを呼び出す方法を示しています。

```
SELECT aws_bedrock.invoke_model (
```

```
    'anthropic.claude-v2', -- model_id
    'application/json', -- content_type
    'application/json', -- accept_type
    '{"prompt": "\n\nHuman:
    You are a helpful assistant that answers questions directly
    and only using the information provided in the context below.
    \nDescribe the answerin detail.\n\nContext: %s \n\nQuestion:
    %s \n\nAssistant:","max_tokens_to_sample":4096,"temperature"
    :0.5,"top_k":250,"top_p":0.5,"stop_sequences":[]}' -- model_input
);
```

Example : aws_comprehend.detect_sentiment – Amazon Comprehend 関数を使用したシンプルなクエリ

```
aws_comprehend.detect_sentiment(
   input_text varchar,
   language_code varchar,
   max_rows_per_batch int)
Returns table (sentiment varchar, confidence real)
```

次の例は、Amazon Comprehend サービスを呼び出す方法を示しています。

```
select sentiment from aws_comprehend.detect_sentiment('This is great', 'en');
```

Example : aws_sagemaker.invoke_endpoint – Amazon SageMaker 関数を使用したシンプルなクエリ

```
aws_sagemaker.invoke_endpoint(
  endpoint_name varchar,
  max_rows_per_batch int,
  VARIADIC model_input "any") -- Babelfish inherits PG's variadic parameter type
Rerurns Varchar(MAX)
```

model_input は VARIADIC としてマークされ、型が「any」であるため、ユーザーは任意の長さと任意のデータ型のリストをモデルへの入力として機能する関数に渡すことができます。次の例は、Amazon SageMaker サービスを呼び出す方法を示しています。

```
SELECT CAST (aws_sagemaker.invoke_endpoint(
    'sagemaker_model_endpoint_name',
    NULL,
    arg1, arg2 -- model inputs are separate arguments )
AS INT) -- cast the output to INT
```

Aurora PostgreSQL での Aurora 機械学習の詳細については、「Aurora PostgreSQL で Amazon Aurora 機械学習を使用する」を参照してください。

制限事項

- Babelfish では配列の作成を行えませんが、配列を表すデータは処理することができます。配列を返す aws_bedrock.invoke_model_get_embeddings のような関数を使用すると、結果は配列要素を含む文字列として返されます。

Babelfish は、リンクサーバーをサポートしています

Babelfish for Aurora PostgreSQL では、バージョン 3.1.0 の PostgreSQL tds_fdw 拡張を使用することにより、リンクサーバーをサポートしています。リンクサーバーを使用するには、tds_fdw 拡張機能をインストールする必要があります。tds_fdw 拡張機能の詳細については、「Amazon Aurora PostgreSQL でサポートされている外部データラッパーを使用する」を参照してください。

tds_fdw 拡張機能のインストール

次の方法を使用して tds_fdw 拡張機能をインストールできます。

PostgreSQL エンドポイントからの CREATE EXTENSION の使用

1. Babelfish データベースの PostgreSQL DB インスタンスに PostgreSQL ポートで接続します。rds_superuser ロールを持つアカウントを使用します。

   ```
   psql --host=your-DB-instance.aws-region.rds.amazonaws.com --port=5432 --
   username=test --dbname=babelfish_db --password
   ```

2. tds_fdw 拡張機能をインストールします。インストールプロセスは 1 回で完了します。DB クラスターの再起動時に、再度インストールする必要はありません。

   ```
   babelfish_db=> CREATE EXTENSION tds_fdw;
   ```

```
CREATE EXTENSION
```

TDS エンドポイントからの **sp_execute_postgresql** ストアドプロシージャの呼び出し

Babelfish は、バージョン 3.3.0 からの sp_execute_postgresql プロシージャの呼び出しによる tds_fdw 拡張機能のインストールをサポートしています。T-SQL エンドポイントから T-SQL ポートを終了せずに PostgreSQL ステートメントを実行できます。詳細については、「Babelfish for Aurora PostgreSQL プロシージャリファレンス」を参照してください。

1. Babelfish データベースの PostgreSQL DB インスタンスに T-SQL ポートで接続します。

```
sqlcmd -S your-DB-instance.aws-region.rds.amazonaws.com -U test -P password
```

2. tds_fdw 拡張機能をインストールします。

```
1>EXEC sp_execute_postgresql N'CREATE EXTENSION tds_fdw';
2>go
```

サポートされている機能

Babelfish では、リモート RDS for SQL Server または Babelfish for Aurora PostgreSQL エンドポイントをリンクサーバーとして追加することをサポートしています。また、他のリモート SQL Server インスタンスをリンクサーバーとして追加できます。次に、OPENQUERY( ) を使用して、これらのリンクサーバーからデータを取得します。Babelfish バージョン 3.2.0 以降、4 つの部分からなる名前もサポートされています。

リンクサーバーを使用するために、以下のストアドプロシージャとカタログビューをサポートしています。

ストアドプロシージャ

- sp_addlinkedserver — Babelfish は、@provstr パラメータをサポートしていません。
- sp_addlinkedsrvlogin
  - リモートデータソースに接続するには、リモートユーザー名とパスワードを明示的に指定する必要があります。ユーザーの自己認証情報では接続できません。Babelfish では、@useself = false のみをサポートしています。
  - Babelfish では、ローカルログインに固有のリモートサーバーへのアクセス設定がサポートされていないため、@locallogin パラメータをサポートしていません。

- sp_linkedservers
- sp_helplinkedsrvlogin
- sp_dropserver
- sp_droplinkedsrvlogin — Babelfish では、ローカルログインに固有のリモートサーバーへのアクセス設定がサポートされていないため、`@locallogin` パラメータをサポートしていません。
- sp_serveroption – Babelfish は以下のサーバーオプションをサポートしています。
  - クエリタイムアウト (Babelfish バージョン 3.2.0 以降)
  - 接続タイムアウト (Babelfish バージョン 3.3.0 以降)
- sp_testlinkedserver (Babelfish バージョン 3.3.0 以降)
- sp_enum_oledb_providers (Babelfish バージョン 3.3.0 以降)

カタログビュー

- sys.servers
- sys.linked_logins

接続に転送中の暗号化を使用する

ソース Babelfish for Aurora PostgreSQL サーバーからターゲットリモートサーバーへの接続には、リモートサーバーのデータベース設定に応じて、転送中の暗号化 (TLS/SSL) を使用します。リモートサーバーが暗号化用に設定されていない場合、リモートデータベースへの要求を行う Babelfish サーバーは、暗号化されていない状態に戻ります。

接続を暗号化するには

- ターゲットリンクサーバーが RDS for SQL Server インスタンスの場合、ターゲット SQL Server インスタンスに `rds.force_ssl = on` を設定します。RDS for SQL Server での SSL/TLS 設定の詳細については、「Microsoft SQL Server DB インスタンスでの SSL の使用」を参照してください。
- ターゲットリンクサーバーが Babelfish for Aurora PostgreSQL クラスターの場合は、ターゲットサーバーに `babelfishpg_tsql.tds_ssl_encrypt = on` および `ssl = on` を設定します。SSL/TLS の使用の詳細については、「Babelfish SSL 設定とクライアント接続」を参照してください。

Babelfish を SQL Server からのリンクサーバーとして追加する

Babelfish for Aurora PostgreSQL を SQL Server からのリンクサーバーとして追加できます。SQL Server データベースで、ODBC 用の Microsoft OLE DB プロバイダー MSDASQL を使用して、Babelfish をリンクサーバーとして追加できます。

MSDASQL プロバイダーを使用して SQL Server からのリンクサーバーとして Babelfish を設定するには、次の 2 つの方法があります。

- ODBC 接続文字列をプロバイダー文字列として指定します。
- リンクサーバーを追加する際に、ODBC データソースのシステム DSN を指定します。

制限事項

- OPENQUERY () は SELECT に対してのみ機能し、DML では機能しません。
- 4 つの部分からなるオブジェクト名は読み取り専用で、リモートテーブルの変更には使用できません。UPDATE は、FROM 句内のリモートテーブルを変更せずに参照できます。
- Babelfish リンクサーバーに対するストアドプロシージャの実行はサポートされていません。
- Babelfish のメジャーバージョンアップグレードは、OPENQUERY( ) に依存しているオブジェクトや、4 つの部分からなる名前で参照されるオブジェクトがある場合、機能しない可能性があります。メジャーバージョンアップグレードの前に、OPENQUERY( ) または 4 つの部分からなる名前を参照するオブジェクトがすべて削除されていることを確認する必要があります。
- データ型 `nvarchar(max)`、`varchar(max)`、`varbinary(max)`、`binary(max)`、および `time` は、リモートの Babelfish サーバーに対して期待どおりに機能しません。CAST 関数を使用して、これらをサポートされているデータ型に変換することをお勧めします。

例

次の例では、Babelfish for Aurora PostgreSQL インスタンスがクラウド内の RDS for SQL Server のインスタンスに接続しています。

```
EXEC master.dbo.sp_addlinkedserver @server=N'rds_sqlserver', @srvproduct=N'',
 @provider=N'SQLNCLI', @datasrc=N'myserver.CB2XKFSFFMY7.US-WEST-2.RDS.AMAZONAWS.COM';
EXEC master.dbo.sp_addlinkedsrvlogin
 @rmtsrvname=N'rds_sqlserver',@useself=N'False',@locallogin=NULL,@rmtuser=N'username',@rmtpassw
```

リンクサーバーが配置されたら、T-SQL OPENQUERY () または標準の 4 部構成の名前付けを使用して、リモートサーバー上のテーブル、ビュー、またはその他のサポートされているオブジェクトを参照できます。

```
SELECT * FROM OPENQUERY(rds_sqlserver, 'SELECT * FROM TestDB.dbo.t1');
SELECT * FROM rds_sqlserver.TestDB.dbo.t1;
```

リンクサーバーとそれに関連するすべてのログインを削除するには:

```
EXEC master.dbo.sp_dropserver @server=N'rds_sqlserver', @droplogins=N'droplogins';
```

トラブルシューティング

ソースサーバーとリモートサーバーの両方に同じセキュリティグループを使用して、互いに通信できるようにします。セキュリティグループは TDS ポート (デフォルトでは 1433) のインバウンドトラフィックのみを許可し、セキュリティグループの送信元 IP をセキュリティグループ ID 自体として設定できます。同じセキュリティグループの別のインスタンスからインスタンスに接続するためのルールを設定する方法の詳細については、「同じセキュリティグループのインスタンスからインスタンスに接続するためのルール」を参照してください。

アクセスが正しく設定されていない場合、リモートサーバーをクエリしようとすると、次の例のようなエラーメッセージが表示されます。

```
TDS client library error: DB #: 20009, DB Msg: Unable to connect: server is unavailable
 or does not exist (mssql2019.aws-region.rds.amazonaws.com), OS #: 110, OS Msg:
 Connection timed out, Level: 9
```

Babelfish での全文検索の使用

バージョン 4.0.0 以降、Babelfish では全文検索 (FTS) の制限付きサポートを提供しています。FTS は、テキストが多いデータの効率的な検索とインデックス作成を可能にするリレーショナルデータベースの強力な機能です。これにより、ユーザーは複雑なテキスト検索を実行し、関連する結果をすばやく取得できます。FTS は、コンテンツ管理システム、e コマースプラットフォーム、ドキュメントアーカイブなど、大量のテキストデータを処理するアプリケーションにとって特に重要です。

Babelfish の全文検索でサポートされている機能について

Babelfish は、以下の全文検索機能をサポートしています。

- CONTAINS 句:
  - CONTAINS 句の基本サポート。

```
CONTAINS (
     {
        column_name
     }
     , '<contains_search_condition>'
)
```

> **Note**
>
> 現在、英語のみがサポートされています。

  - `simple_term` 検索文字列の包括的な処理と翻訳。
- `FULLTEXT INDEX` 句:
  - `CREATE FULLTEXT INDEX ON table_name(column_name [...n]) KEY INDEX index_name` ステートメントのみをサポートします。
  - 完全な `DROP FULLTEXT INDEX` ステートメントをサポートします。

> **Note**
>
> 全文インデックスのインデックスを再作成するには、全文インデックスを削除し、同じ列に新しいインデックスを作成する必要があります。

- 検索条件の特殊文字:
  - Babelfish は、検索文字列内の特殊文字の 1 回限りの出現を効果的に処理します。

> **Note**
>
> Babelfish は検索文字列内の特殊文字を識別するようになりましたが、取得される結果は T-SQL で取得されるものとは異なる場合があることに注意してください。

- column_name のテーブルエイリアス:

- テーブルエイリアスがサポートされているため、ユーザーは全文検索でより簡潔で読みやすい SQL クエリを作成できます。

Babelfish での全文検索に関する制限

- 現在、CONTAINS 句に対して Babelfish では以下のオプションはサポートされていません。
  - 英語以外の特殊文字や言語はサポートされていません。サポートされていない文字と言語に関する一般的なエラーメッセージが表示されます

    ```
    Full-text search conditions with special characters or languages other than English
     are not currently supported in Babelfish
    ```

  - column_list のような複数の列
  - PROPERTY 属性
  - prefix_term、generation_term、generic_proximity_term、custom_proximity_term、および weighted_term
  - ブール演算子はサポートされていません。使用すると、次のエラーメッセージが表示されます。

    ```
    boolean operators not supported
    ```

  - ドットを含む識別子名はサポートされていません。
- 現在、CREATE FULLTEXT INDEX 句に対して Babelfish では以下のオプションはサポートされていません。
  - [TYPE COLUMN type_column_name]
  - [LANGUAGE language_term]
  - [STATISTICAL_SEMANTICS]
  - カタログファイルグループオプション
  - オプションあり
- フルテキストカタログの作成はサポートされていません。フルテキストインデックスを作成する場合、フルテキストカタログは必要ありません。
- CREATE FULLTEXT INDEX は、ドットを含む識別子名をサポートしていません。
- Babelfish は現在、検索文字列で連続する特殊文字をサポートしていません。連続した特殊文字を使用すると、次のメッセージが表示されます。

```
Consecutive special characters in the full-text search condition are not currently
 supported in Babelfish
```

Babelfish は地理空間データ型をサポート

バージョン 3.5.0 および 4.1.0 以降、Babelfish では次の 2 つの空間データ型がサポートされています。

- ジオメトリデータ型 – このデータ型は、平面データまたはユークリッド (フラットアース) データを保存することを目的としています。
- 地理データ型 – このデータ型は、GPS の緯度や経度の座標など、楕円データまたはラウンドアースデータを保存することを目的としています。

これらのデータ型により、空間データの保存と操作が可能になりますが、いくつかの制限があります。

Babelfish の地理空間データ型について

- 地理空間データ型は、ビュー、プロシージャ、テーブルなどのさまざまなデータベースオブジェクトでサポートされています。

- 2-D ポイントデータ型をサポートし、位置データを緯度、経度、有効な空間リファレンスシステム識別子 (SRID) で定義されたポイントとして保存します。
- JDBC、ODBC、DOTNET、PYTHON などのドライバーを介して Babelfish に接続するアプリケーションは、この地理空間機能を利用できます。

Babelfish でサポートされているジオメトリデータ型関数

- STGeomFromText (***geometry_tagged_text***, SRID) – Well-Known Text (WKT) 表現を使用してジオメトリインスタンスを作成します。
- STPointFromText (***point_tagged_text***, SRID) – WKT 表現を使用してポイントインスタンスを作成します。
- Point (X、Y、SRID) – x および y 座標の浮動小数点値を使用してポイントインスタンスを作成します。
- <geometry_instance>.STAsText ( ) – ジオメトリインスタンスから WKT 表現を抽出します。

- <geometry_instance>.STDistance (other_geometry) – 2 つのジオメトリインスタンス間の距離を計算します。
- <geometry_instance>.STX – ジオメトリインスタンスの X 座標 (経度) を抽出します。
- <geometry_instance>.STY – ジオメトリインスタンスの Y 座標 (緯度) を抽出します。

Babelfish でサポートされている地理データ型関数

- STGeomFromText (***geography_tagged_text***, SRID) – WKT 表現を使用して地理インスタンスを作成します。
- STPointFromText (***point_tagged_text***, SRID) – WKT 表現を使用してポイントインスタンスを作成します。
- Point (Lat, Long, SRID) – 緯度と経度の浮動小数点値を使用してポイントインスタンスを作成します。
- <geography_instance>.STAsText ( ) – 地理インスタンスから WKT 表現を抽出します。
- <geography_instance>.STDistance (other_geography) – 2 つの地理インスタンス間の距離を計算します。
- <geography_instance>.Lat – 地理インスタンスの緯度値を抽出します。
- <geography_instance>.Long – 地理インスタンスの経度値を抽出します。

Babelfish での地理空間データ型の制限

- 現在、Babelfish は地理空間データ型のポイントインスタンスの Z-M フラグなどのより高度な機能をサポートしていません。
- ポイントインスタンス以外のジオメトリ型は現在サポートされていません。
  - LineString
  - CircularString
  - CompoundCurve
  - Polygon
  - CurvePolygon
  - MultiPoint
  - MultiLineString
  - MultiPolygon
  - GeometryCollection

- 現在、空間インデックス作成は地理空間データ型ではサポートされていません。
- 現在、これらのデータ型では、記載されている関数のみがサポートされています。詳細については、Babelfish でサポートされているジオメトリデータ型関数およびBabelfish でサポートされている地理データ型関数を参照してください。
- 地理データの STDistance 関数の出力は、T-SQL と比較して精度に若干のばらつきが含まれる場合があります。これは、基盤となる PostGIS 実装に起因するものです。詳細については、「ST_Distance」を参照してください。
- 最適なパフォーマンスを得るには、Babelfish で追加の抽象化レイヤーを作成せずに、組み込みの地理空間データ型を使用します。

> **Tip**
>
> カスタムデータ型を作成することはできますが、地理空間データの上に作成することはお勧めしません。これは複雑性が増し、サポートの制限により予期しない動作が発生する可能性があるためです。

- Babelfish では、地理空間関数名がキーワードとして使用され、意図した方法で使用された場合にのみ空間オペレーションを実行します。

> **Tip**
>
> Babelfish でユーザー定義の関数とプロシージャを作成する際は、組み込みの地理空間関数と同じ名前を使用しないでください。同じ名前の既存のデータベースオブジェクトがある場合は、 `sp_rename`を使用して名前を変更します。

Babelfish のトラブルシューティング

以下に、いくつかの Babelfish DB クラスターの問題のトラブルシューティングのヒントと回避策を示します。

トピック

接続障害

Babelfish と Aurora DB クラスターの接続障害の一般的な原因は、以下を含みます:

- セキュリティグループはアクセスを許可していません — Babelfish への接続に問題がある場合は、デフォルトの Amazon EC2 セキュリティグループに IP アドレスが追加されていることを確認してください。https://checkip.amazonaws.com/を使用して IP アドレスを特定し、TDS ポートと PostgreSQL ポートのインバウンドルールに追加します。詳細については、Amazon EC2 ユーザーガイドの「セキュリティグループへのルールの追加」を参照してください。
- SSL 設定の不一致 — Aurora PostgreSQL で `rds.force_ssl` パラメータがオン (1 に設定) の場合、クライアントは SSL 経由で Babelfish に接続する必要があります。クライアントが正しく設定されていない場合は、次のようなエラーメッセージが表示されます。

  ```
  Cannot connect to your-Babelfish-DB-cluster, 1433
  ----------------------
  ADDITIONAL INFORMATION:
  no pg_hba_conf entry for host "256.256.256.256", user "your-user-name",
  "database babelfish_db", SSL off (Microsoft SQL Server, Error: 33557097)
  ...
  ```

  このエラーは、ローカルクライアントと Babelfish DB クラスターの間で SSL 設定の問題が発生する可能性があり、クラスターがクライアントに SSL を使用する必要があることを示します (`rds.force_ssl` パラメータが 1 に設定されます)。SSL 設定の詳細については、Amazon RDS ユーザーガイドの「PostgreSQL DB インスタンスで SSL を使用する」を参照してください。

  Babelfish に接続するために SQL Server Management Studio (SSMS) を使用していてこのエラーが発生する場合は、[Connection Properties] (接続プロパティ) ペインで [Encrypt connection] (接続の暗号化) と [Trust server certificate] (サーバー証明書を信頼する) 接続オプションを選択して、もう一度試すことができます。これらの設定は、SSMS の SSL 接続要件を処理します。

Aurora 接続不具合のトラブルシューティングの詳細については、「Amazon RDS DB インスタンスに接続できない」を参照してください。

Babelfish をオフにする

Babelfish が不要になったら、Babelfish 機能をオフにすることができます。

いくつかの考慮事項に注意してください。

- 場合によっては、Aurora PostgreSQL への移行が完了する前に Babelfish をオフにしてしまうことがあります。この場合にもし DDL が SQL Server のデータ型に依存している、またはコードで T-SQL 機能を使用していると、コードは失敗します。
- Babelfish インスタンスをプロビジョニングした後、Babelfish 拡張機能をオフにした場合、同じクラスターで同じデータベースを再度プロビジョンすることはできません。

Babelfish をオフにするにはパラメータグループを変更し、`rds.babelfish_status` を `OFF` に設定します。`rds.babelfish_status` を`datatypeonly` に設定すると、Babelfish をオフにした状態で SQL Server のデータ型を引き続き使用できます。

パラメータグループで Babelfish をオフにすると、そのパラメータグループを使用するすべてのクラスターで Babelfish の機能が失われます。

Aurora パラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。Babelfish 独自のパラメータの詳細については、「Babelfish の DB クラスターパラメータグループ設定」を参照してください。

Babelfish バージョンのアップデート

Babelfish は Aurora PostgreSQL バージョン 13.4 以降のリリースで利用できるオプションです。Babelfish のアップデートは、Aurora PostgreSQL データベースエンジンの特定の新しいリリースで利用可能になります。詳細については、「Aurora PostgreSQL のリリースノート」を参照してください。

> **Note**
>
> Aurora PostgreSQL 13 バージョンで実行している Babelfish DB クラスターを Aurora PostgreSQL 14.3、14.4、14.5 にアップグレードすることはできません。また、Babelfish は 13.x から 15.x への直接のアップグレードをサポートしていません。最初に 13.x DB クラスターを 14.6 以降のバージョンにアップグレードしてから、15.x バージョンにアップグレードする必要があります。

Babelfish の各リリースでサポートされる機能のリストについては、「バージョンごとに Babelfish でサポートされている機能」を参照してください。

サポートされていない機能のリストについては、「Babelfish でサポートされていない機能」を参照してください。

以下の例のように、db-engine-versions AWS CLI コマンドを使用して、AWS リージョン で Babelfish をサポートしている Aurora PostgreSQL バージョンのリストを取得できます。

Linux、macOS、Unix の場合:

```
$ aws rds describe-db-engine-versions --region us-east-1 \
    --engine aurora-postgresql \
    --query '*[]|[?SupportsBabelfish==`true`].[EngineVersion]' \
    --output text
13.4
13.5
13.6
13.7
13.8
14.3
14.4
14.5
14.6
```

```
14.7
14.8
14.9
14.10
15.2
15.3
15.4
15.5
16.1
```

詳細については、「AWS CLI コマンドリファレンス」の describe-db-engine-versions を参照してください。

以下のトピックでは、Aurora PostgreSQL DB クラスターで実行されている Babelfish のバージョンを確認する方法と、新しいバージョンにアップグレードする方法について説明します。

目次

Babelfish のバージョンの確認

Babelfish をクエリして、Babelfish のバージョン、Aurora PostgreSQL バージョン、および互換性のある Microsoft SQL Server のバージョンに関する詳細を調べることができます。TDS ポートまたは PostgreSQL ポートを使用できます。

- To use the PostgreSQL port to query for version information

TDS ポートを使用してバージョン情報を照会するには

1. sqlcmd または ssms を使用して、Babelfish DB クラスターのエンドポイントに接続します。

```
sqlcmd -S bfish_db.cluster-123456789012.aws-region.rds.amazonaws.com,1433 -U
    login-id -P password -d db_name
```

2. Babelfish のバージョンを特定するには、以下のクエリを実行します。

```
1> SELECT CAST(serverproperty('babelfishversion') AS VARCHAR)
2> GO
```

クエリによって以下のような結果が返されます:

```
serverproperty
------------------------------
3.4.0

(1 rows affected)
```

3. Aurora PostgreSQL DB クラスターのバージョンを確認するには、以下のクエリを実行します:

```
1> SELECT aurora_version() AS aurora_version
2> GO
```

クエリによって以下のような結果が返されます:

```
aurora_version


------------------------------------------------
15.5.0

(1 rows affected)
```

4. 互換性のある Microsoft SQL Server のバージョンを特定するには、次のクエリを実行します。

```
1> SELECT @@VERSION AS version
```

```
2> GO
```

クエリによって以下のような結果が返されます:

```
Babelfish for Aurora PostgreSQL with SQL Server Compatibility - 12.0.2000.8
Dec 7 2023 09:43:06
Copyright (c) Amazon Web Services
PostgreSQL 15.5 on x86_64-pc-linux-gnu (Babelfish 3.4.0)

(1 rows affected)
```

Babelfish と Microsoft SQL Server のマイナーな違いを 1 つ示す例として、次のクエリを実行できます。Babelfish ではクエリが 1 を返し、Microsoft SQL Server ではクエリが NULL を返します。

```
SELECT CAST(serverproperty('babelfish') AS VARCHAR) AS runs_on_babelfish
```

次の手順に示すように、PostgreSQL ポートを使用してバージョン情報を取得することもできます。

PostgreSQL ポートを使用してバージョン情報を照会するには

1. psql または pgAdmin を使用して、Babelfish DB クラスターのエンドポイントに接続します。

```
psql host=bfish_db.cluster-123456789012.aws-region.rds.amazonaws.com
     port=5432 dbname=babelfish_db user=sa
```

2. 読みやすい出力のためには、psql の拡張機能 (\x) をオンにします。

```
babelfish_db=> \x
babelfish_db=> SELECT
babelfish_db=> aurora_version() AS aurora_version,
babelfish_db=> version() AS postgresql_version,
babelfish_db=> sys.version() AS Babelfish_compatibility,
babelfish_db=> sys.SERVERPROPERTY('BabelfishVersion') AS Babelfish_Version;
```

クエリでは、以下のような出力が返されます。

```

-[ RECORD 1 ]-----------
+-------------------------------------------------------------------------------------
aurora_version          | 15.5.0
```

```
postgresql_version      | PostgreSQL 15.5 on x86_64-pc-linux-gnu, compiled by
 x86_64-pc-linux-gnu-gcc (GCC) 9.5.0, 64-bit
babelfish_compatibility | Babelfish for Aurora Postgres with SQL Server
 Compatibility - 12.0.2000.8                      +
                         | Dec 7 2023 09:43:06
                                             +
                         | Copyright (c) Amazon Web Services
                                              +
                         | PostgreSQL 15.5 on x86_64-pc-linux-gnu (Babelfish 3.4.0)
babelfish_version       | 3.4.0
```

Babelfish クラスターを新しいメジャーバージョンにアップグレードする

Babelfish の新しいバージョンは、バージョン 13.4 以降の Aurora PostgreSQL データベースエンジンのいくつかの新しいリリースで使用可能になります。Babelfish の新しいリリースごとに、独自のバージョン番号があります。Aurora の PostgreSQL と同様に、Babelfish は、バージョンについて *major.minor.patch* という命名方式を使用します。例えば、Babelfish の最初のリリースである Babelfish バージョン 1.0.0 は、Aurora PostgreSQL 13.4.0 の一部として使用可能になりました。

Babelfish は個別のインストールプロセスを必要としません。Babelfish for Aurora PostgreSQL DB クラスターの作成 で説明されているように、[Turn on Babelfish] (Babelfish を有効にする) は、Aurora PostgreSQL DB クラスターを作成するときに選択するオプションです。

同様に、サポートしている Aurora DB クラスターとは独立して Babelfish をアップグレードすることはできません。既存の Babelfish for Aurora PostgreSQL DB クラスターを新しいバージョンの Babelfish にアップグレードするには、Aurora PostgreSQL DB クラスターを、使用したい Babelfish のバージョンをサポートする新しいバージョンにアップグレードします。アップグレードの手順は、Babelfish デプロイをサポートしている Aurora PostgreSQL のバージョンによって次のように異なります。

メジャーバージョンのアップグレード

Aurora PostgreSQL 15.2 バージョンにアップグレードする前に、以下の Aurora PostgreSQL バージョンを、Aurora PostgreSQL 14.6 以降のバージョンにアップグレードする必要があります。

- Aurora PostgreSQL 13.8 以上のすべてのバージョン
- Aurora PostgreSQL 13.7.1 以上のすべてのマイナーバージョン
- Aurora PostgreSQL 13.6.4 以上のすべてのマイナーバージョン

Aurora PostgreSQL 14.6 以降のバージョンを、Aurora PostgreSQL 15.2 以降にアップグレードできます。

Aurora PostgreSQL DB クラスターを新しいメジャーバージョンにアップグレードするには、いくつかの準備作業が必要です。詳細については、「メジャーバージョンのアップグレードを実施する方法」を参照してください。Babelfish for Aurora PostgreSQL DB クラスターを正常にアップグレードするには、新しい Aurora PostgreSQL バージョン用のカスタム DB クラスターパラメータグループを作成する必要があります。この新しいパラメータグループには、アップグレードするクラスターと同じ Babelfish パラメータ設定が含まれている必要があります。詳細およびメジャーバージョンアップグレードのソースとターゲットの表については、「Babelfish を新しいメジャーバージョンにアップグレードする」を参照してください。

マイナーバージョンのアップグレードとパッチ

マイナーバージョンとパッチでは、アップグレード用に新しい DB クラスターパラメータグループを作成する必要はありません。マイナーバージョンとパッチは、自動適用か手動適用かにかかわらず、マイナーバージョンアップグレードプロセスを使用できます。詳細およびバージョンのソースとターゲットの表については、「Babelfish を新しいマイナーバージョンにアップグレードする」を参照してください。

> **Note**
>
> メジャーアップグレードまたはマイナーアップグレードを実行する前に、保留中のすべてのメンテナンスタスクを Babelfish for Aurora PostgreSQL クラスターに適用してください。

トピック

- Babelfish を新しいマイナーバージョンにアップグレードする
- Babelfish を新しいメジャーバージョンにアップグレードする

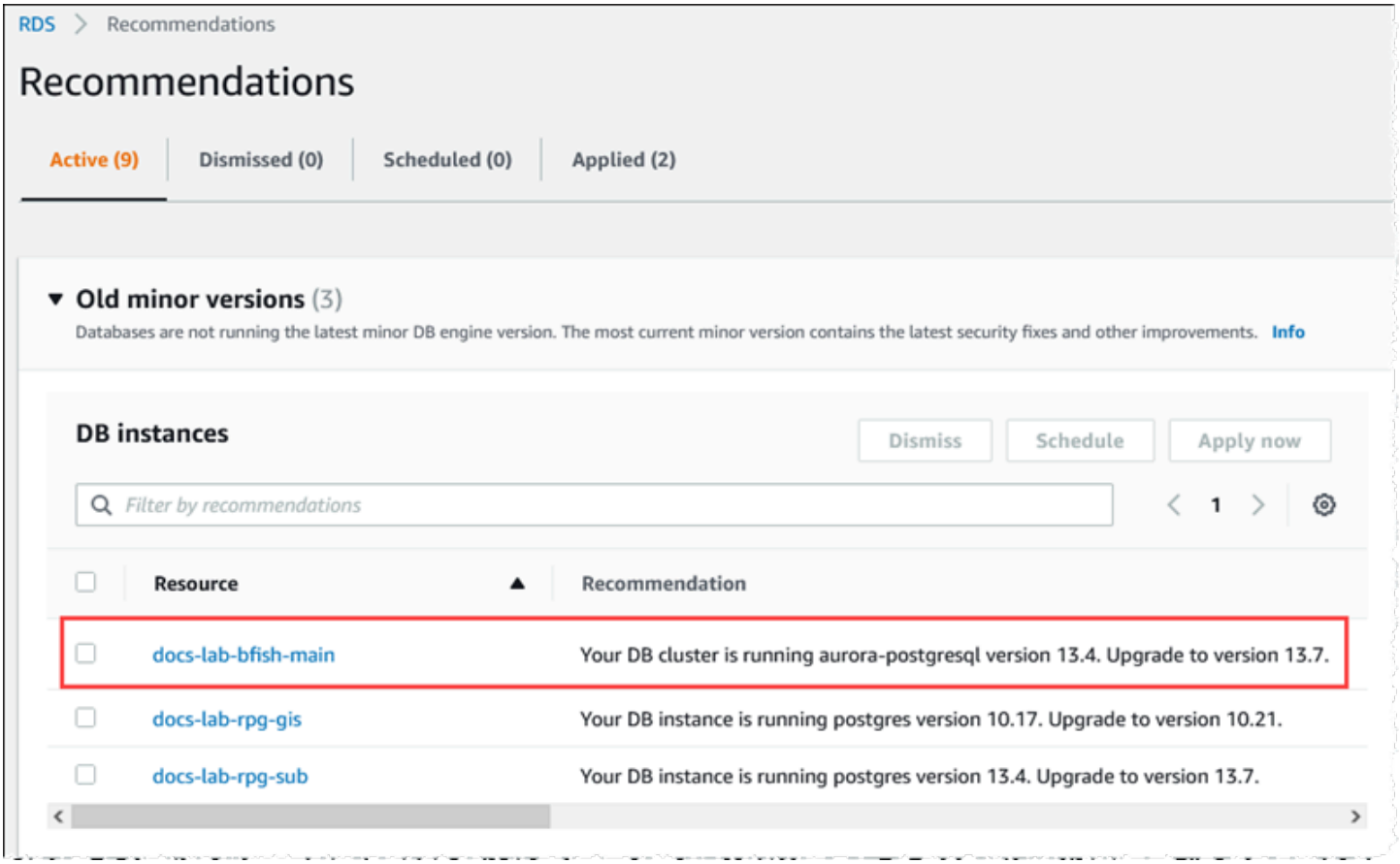
Babelfish を新しいマイナーバージョンにアップグレードする

新しいマイナーバージョンには、下位互換性のある変更のみが含まれます。パッチバージョンには、リリース後にマイナーバージョンに追加された重要な修正が含まれています。例えば、Aurora PostgreSQL 13.4 の最初のリリースのバージョンラベルは Aurora PostgreSQL 13.4.0 でした。Aurora PostgreSQL 13.4.1、13.4.2、13.4.4 など、そのマイナーバージョン用のいくつかのパッチがこれまでにリリースされています。Aurora PostgreSQL の各バージョンで利用できるパッ

チは、そのバージョンの Aurora PostgreSQL リリースノートの上部にある [Patch releases] (パッチリリース) リストで確認できます。詳細については、「Aurora PostgreSQL リリースノート」の「PostgreSQL 14.3」を参照してください。

Aurora PostgreSQL DB クラスターに [Auto minor version upgrade] (自動マイナーバージョンアップグレード) オプションが設定されている場合、Babelfish for Aurora PostgreSQL DB クラスターは、クラスターのメンテナンスウィンドウ中に自動的にアップグレードされます。自動マイナーバージョンアップグレード (AMVU) とその使用方法については、「Aurora DB クラスターのマイナーバージョン自動アップグレード」を参照してください。クラスターが AMVU を使用していない場合、メンテナンスタスクに応答するか、新しいバージョンを使用するようにクラスターを変更することによって、Babelfish for Aurora PostgreSQL DB クラスターを新しいマイナーバージョンに手動でアップグレードできます。

インストールする Aurora PostgreSQL バージョンを選択し、AWS Management Console で既存の Aurora PostgreSQL DB クラスターを表示すると、バージョンには *major.minor* の桁のみが表示されます。例えば、Aurora PostgreSQL 13.4 を搭載した既存の Babelfish for Aurora PostgreSQL DB クラスターのコンソールにある次の画像では、クラスターを Aurora PostgreSQL の新しいマイナーリリースであるバージョン 13.7 にアップグレードすることを推奨しています。



###レベルを含む完全なバージョンの詳細を取得するには、`aurora_version` Aurora PostgreSQL 関数を使用して、Aurora PostgreSQL DB クラスターにクエリできます。詳細については、「Aurora PostgreSQL 関数のリファレンス」の「aurora_version」を参照してください。To use the PostgreSQL port to query for version information プロシージャでの関数の使用例については、「Babelfish のバージョンの確認」を参照してください。

次の表は、Aurora PostgreSQL と Babelfish のバージョン、およびマイナーバージョンアップグレードプロセスをサポートできる利用可能なターゲットバージョンを示しています。

現在のソースバージョン		最新のアップグレードターゲット		その他の利用可能なアップグレードバージョン			
Aurora PostgreSQL	Babelfish	Aurora PostgreSQL	Babelfish	Babelfish オプションを伴う Aurora PostgreSQL バージョン			
15.4	3.3.0	15.5	3.4.0				
15.3.2	3.2.1	15.5	3.4.0	15.4			
15.2.4	3.1.3	15.5	3.4.0	15.4	15.3		
14.9.1	2.6.0	14.10	2.7.0				
14.8.2	2.5.1	14.10	2.7.0	14.9.1			
14.7.4	2.4.3	14.10	2.7.0	14.9.1	14.8.2		
14.6.4	2.3.3	14.10	2.7.0	14.9.1	14.8.2	14.7.4	
14.5.3	2.2.3	14.10	2.7.0	14.9.1	14.8.2	14.7.4	14.6.4
14.3.1	2.1.1	14.6	2.3.0				
14.3.0	2.1.0	14.6	2.3.0	14.3.1			
13.8	1.4.0	13.9	1.5				
13.7.1	1.3.1	13.9	1.5	13.8			
13.7.0	1.3.0	13.9	1.5	13.7.1			

現在のソースバージョン		最新のアップグレードターゲット		その他の利用可能なアップグレードバージョン			
13.6.4	1.2.4	13.9	1.5	13.7			
13.6.3	1.2.1	13.9	1.5	13.7	13.6.4		
13.6.2	1.2.1	13.9	1.5	13.7	13.6.4		
13.6.1	1.2.0	13.9	1.5	13.7	13.6.4		
13.6.0	1.2.0	13.9	1.5	13.7	13.6.4		
13.5	1.1.0	13.9	1.5	13.7	13.6		
13.4	1.0.0	13.9	1.5	13.7	13.6	13.5	

Babelfish を新しいメジャーバージョンにアップグレードする

メジャーバージョンアップグレードの場合、まず、Babelfish for Aurora PostgreSQL DB クラスターをメジャーバージョンアップグレードをサポートするバージョンにアップグレードする必要があります。これを実現するには、DB クラスターにパッチアップデートまたはマイナーバージョンアップグレードを適用します。詳細については、「Babelfish を新しいマイナーバージョンにアップグレードする」を参照してください。

次の表は、メジャーバージョンアップグレードをサポートできる Aurora PostgreSQL のバージョンと Babelfish のバージョンを示しています。

現在のソースバージョン		使用可能な最新のアップグレードターゲット		その他の使用可能なバージョン (マイナーバージョンアップグレード)		
Aurora PostgreSQL	Babelfish	Aurora PostgreSQL	Babelfish	Aurora PostgreSQL のバージョン (Babelfish のバージョン)		
15.5	3.4.0	16.1	4.0.0			
15.4	3.3.0	16.1	4.0.0			

現在のソースバージョン		使用可能な最新のアップグレードターゲット		その他の使用可能なバージョン (マイナーバージョンアップグレード)		
15.3	3.2.0	16.1	4.0.0			
15.2	3.1.0	16.1	4.0.0			
14.10	2.7.0	15.5	3.4.0			
14.9	2.6.0	15.5	3.4.0	15.4 (3.3.0)		
14.8	2.5.0	15.5	3.4.0	15.4 (3.3.0)	15.3 (3.2.0)	
14.7	2.4.0	15.5	3.4.0	15.4 (3.3.0)	15.3 (3.2.0)	15.2 (3.1.0)
14.6	2.3.0	15.5	3.4.0	15.4 (3.3.0)	15.3 (3.2.0)	15.2 (3.1.0)
13.9	1.5.0	14.6	2.3.0			
13.8	1.4.0	14.6	2.3.0			
13.7.1	1.3.1	14.6	2.3.0	13.8 (1.4)		
13.6.4	1.2.2	14.6	2.3.0	13.8 (1.4)	13.7 (1.3)	

Babelfish を新しいメジャーバージョンにアップグレードする前に

アップグレードには短時間の停止が必要な場合があります。そのため、メンテナンス期間中、または使用率の低い時間帯にアップグレードを実行またはスケジュールすることをお勧めします。

メジャーバージョンアップグレードを実行する前に

1. 「Babelfish のバージョンの確認」で概説されているコマンドを使用して、既存の Aurora PostgreSQL DB クラスターの Babelfish のバージョンを識別します。Aurora PostgreSQL のバージョンと Babelfish のバージョンの情報は PostgreSQL によって処理されるため、「To use the PostgreSQL port to query for version information」プロシージャに詳述されているステップに従って詳細を確認してください。
2. お使いのバージョンがメジャーバージョンアップグレードをサポートしているかどうかを確認してください。メジャーバージョンアップグレード機能をサポートするバージョンのリストについ

ては、「Babelfish を新しいマイナーバージョンにアップグレードする」を参照して、アップグレード前の必要なタスクを実行してください。

例えば、Babelfish バージョンが Aurora PostgreSQL 13.5 DB クラスターで実行されていて、Aurora PostgreSQL 15.2 にアップグレードする場合は、まず、すべてのマイナーリリースとパッチを適用して、クラスターを Aurora PostgreSQL 14.6 以降にアップグレードします。クラスターのバージョンが 14.6 以降になったら、メジャーバージョンアップグレードプロセスを続行します。

3. バックアップとして、現在の Babelfish DB クラスターの手動スナップショットを作成します。バックアップにより、クラスターを Aurora PostgreSQL バージョン、Babelfish バージョンに復元し、すべてのデータをアップグレード前の状態に復元できます。詳細については、「DB クラスタースナップショットの作成」を参照してください。このクラスターをアップグレード前の状態に復元する場合は、既存のカスタム DB クラスターパラメータグループを再び使用できるように残しておいてください。詳細については、「DB クラスターのスナップショットからの復元」および「パラメータグループに関する考慮事項」を参照してください。

4. ターゲット Aurora PostgreSQL DB バージョン用にカスタム DB クラスターのパラメータグループを準備します。現在の Babelfish for Aurora PostgreSQL DB クラスターから Babelfish のパラメータの設定を複製します。Babelfish のすべてのパラメータのリストについては、「Babelfish の DB クラスターパラメータグループ設定」を参照してください。メジャーバージョンアップグレードの場合、次のパラメータにはソース DB クラスターと同じ設定が必要です。アップグレードを成功させるには、すべての設定が同じである必要があります。

   - rds.babelfish_status
   - babelfishpg_tds.tds_default_numeric_precision
   - babelfishpg_tds.tds_default_numeric_scale
   - babelfishpg_tsql.database_name
   - babelfishpg_tsql. default_locale
   - babelfishpg_tsql.migration_mode
   - babelfishpg_tsql.server_collation_name

> **Warning**
>
> 新しい Aurora PostgreSQL バージョンのカスタム DB クラスターパラメータグループの Babelfish パラメータの設定が、アップグレードするクラスターのパラメータ値と一致しない場合、`ModifyDBCluster` 操作は失敗します。`InvalidParameterCombination`

エラーメッセージが AWS Management Console に、または `modify-db-cluster` AWS CLI コマンドの出力に表示されます。

5. AWS Management Console または AWS CLI コマンドを使用して、カスタム DB クラスターパラメータグループを作成します。アップグレードする Aurora PostgreSQL のバージョンに対応する Aurora PostgreSQL ファミリーを選択します。

> **Tip**
>
> パラメータグループは AWS リージョン レベルで管理されます。AWS CLI で作業する場合、コマンドで `--region` を指定する代わりに、デフォルトのリージョンで設定できます。AWS CLI の使用の詳細については、AWS Command Line Interface ユーザーガイドの「クイックセットアップ」を参照してください。

メジャーバージョンアップグレードの実行

1. Aurora PostgreSQL DB クラスターを新しいメジャーバージョンにアップグレードします。詳細については、「Aurora PostgreSQL エンジンを新しいメジャーバージョンにアップグレードする」を参照してください。
2. クラスターのライターインスタンスを再起動して、パラメータ設定を有効にします。

新しいメジャーバージョンにアップグレードした後

新しい Aurora PostgreSQL バージョンへのメジャーバージョンアップグレードの後、`IDENTITY` 列のあるテーブルの `IDENTITY` 値がアップグレード前の値よりも大きくなる (+32) 場合があります。その結果、このようなテーブルに次の行が挿入されると、生成された ID 列の値が +32 の数値にジャンプし、そこからシーケンスを開始します。この条件が Babelfish DB クラスターの機能に悪影響を及ぼすことはありません。ただし、必要に応じて、列の最大値に基づいてシーケンスオブジェクトをリセットできます。そのためには、`sqlcmd` または別の SQL Server クライアントを使用して、Babelfish ライターインスタンスの T-SQL ポートに接続します。詳細については、「SQL Server クライアントを使用した DB クラスターへの接続」を参照してください。

```
sqlcmd -S bfish-db.cluster-123456789012.aws-region.rds.amazonaws.com,1433 -U
    sa -P ******** -d dbname
```

接続したら、次の SQL コマンドを使用して、関連するシーケンスオブジェクトのシードに使用できるステートメントを生成します。この SQL コマンドは、単一データベースと複数データベースの両

方の Babelfish 構成で機能します。これら 2 つのデプロイモデルの詳細については、「1 つのデータベースまたは複数のデータベースでの babelfish の使用」を参照してください。

```
DECLARE @schema_prefix NVARCHAR(200) = ''
IF current_setting('babelfishpg_tsql.migration_mode') = 'multi-db'
    SET @schema_prefix = db_name() + '_'
SELECT 'SELECT setval(pg_get_serial_sequence(''' + @schema_prefix +
 schema_name(tables.schema_id)
    + '.' + tables.name + ''', ''' + columns.name + '''),(select max(' + columns.name +
 ')
    FROM ' + schema_name(tables.schema_id) + '.' + tables.name + '));
    'FROM sys.tables tables JOIN sys.columns
    columns ON tables.object_id = columns.object_id
    WHERE columns.is_identity = 1
GO
```

クエリによって一連の SELECT ステートメントが生成されます。これらを実行して、IDENTITY の最大値をリセットし、ギャップを埋めることができます。以下は、Babelfish クラスターで実行されているサンプルの SQL Server データベース Northwind を使用した場合の出力を示しています。

```
------------------------------------------------------------
SELECT setval(pg_get_serial_sequence('northwind_dbo.categories', 'categoryid'),(select
 max(categoryid)
    FROM dbo.categories));

SELECT setval(pg_get_serial_sequence('northwind_dbo.orders', 'orderid'),(select
 max(orderid)
    FROM dbo.orders));


SELECT setval(pg_get_serial_sequence('northwind_dbo.products', 'productid'),(select
 max(productid)
    FROM dbo.products));


SELECT setval(pg_get_serial_sequence('northwind_dbo.shippers', 'shipperid'),(select
 max(shipperid)
    FROM dbo.shippers));


SELECT setval(pg_get_serial_sequence('northwind_dbo.suppliers', 'supplierid'),(select
 max(supplierid)
    FROM dbo.suppliers));
```

```
(5 rows affected)
```

ステートメントを 1 つずつ実行して、シーケンス値をリセットします。

例: Babelfish DB クラスターをメジャーリリースにアップグレードする

この例では、Babelfish バージョン 1.2.2 を実行している Aurora PostgreSQL 13.6.4 DB クラスターを Aurora PostgreSQL 14.6 にアップグレードする方法を説明する一連の AWS CLI コマンドを見つけることができます。まず、Aurora PostgreSQL 14 用のカスタム DB クラスターパラメータグループを作成します。次に、Aurora PostgreSQL バージョン 13 のソースの値と一致するようにパラメータ値を変更します。最後に、ソースクラスターを変更して、アップグレードを実行します。詳細については、「Babelfish の DB クラスターパラメータグループ設定」を参照してください。このトピックでは、AWS Management Console を使用してアップグレードを実行する方法についても説明しています。

create-db-cluster-cluster-parameter-group CLI コマンドを使用して、新しいバージョン用の DB クラスターパラメータグループを作成します。

Linux、macOS、Unix の場合:

```
aws rds create-db-cluster-parameter-group \
    --db-cluster-parameter-group-name docs-lab-babelfish-apg-14 \
    --db-parameter-group-family aurora-postgresql14 \
    --description 'New custom parameter group for upgrade to new major version' \
    --region us-west-1
```

このコマンドを発行すると、カスタム DB クラスターパラメータグループが AWS リージョン に作成されます。以下のような出力が表示されます。

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "docs-lab-babelfish-apg-14",
        "DBParameterGroupFamily": "aurora-postgresql14",
        "Description": "New custom parameter group for upgrade to new major version",
        "DBClusterParameterGroupArn": "arn:aws:rds:us-west-1:111122223333:cluster-
pg:docs-lab-babelfish-apg-14"
    }
}
```

詳細については、「DB クラスターのパラメータグループの作成」を参照してください。

modify-db-cluster-parameter-group CLI コマンドを使用して、ソースクラスターと一致するように設定を変更します。

Windows の場合:

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name docs-lab-
babelfish-apg-14 ^
  --parameters
 "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot" ^
 "ParameterName=babelfishpg_tds.tds_default_numeric_precision,ParameterValue=38,ApplyMethod=pen
reboot" ^
 "ParameterName=babelfishpg_tds.tds_default_numeric_scale,ParameterValue=8,ApplyMethod=pending-
reboot" ^
 "ParameterName=babelfishpg_tsql.database_name,ParameterValue=babelfish_db,ApplyMethod=pending-
reboot" ^
 "ParameterName=babelfishpg_tsql.default_locale,ParameterValue=en-
US,ApplyMethod=pending-reboot" ^
 "ParameterName=babelfishpg_tsql.migration_mode,ParameterValue=single-
db,ApplyMethod=pending-reboot" ^
 "ParameterName=babelfishpg_tsql.server_collation_name,ParameterValue=sql_latin1_general_cp1_ci
reboot"
```

レスポンスは次の例のようになります。

```
{
    "DBClusterParameterGroupName": "docs-lab-babelfish-apg-14"
}
```

modify-db-cluster CLI コマンドを使用して、新しいバージョンと新しいカスタム DB クラスターパラメータグループを使用するようにクラスターを変更します。また、次のサンプルに示されているように、--allow-major-version-upgrade 引数を指定します。

```
aws rds modify-db-cluster \
--db-cluster-identifier docs-lab-bfish-apg-14 \
--engine-version 14.6 \
--db-cluster-parameter-group-name docs-lab-babelfish-apg-14 \
--allow-major-version-upgrade \
--region us-west-1 \
--apply-immediately
```

reboot-db-instance CLI コマンドを使用して、クラスターのライターインスタンスを再起動し、パラメータ設定を有効にします。

```
aws rds reboot-db-instance \
--db-instance-identifier docs-lab-bfish-apg-14-instance-1\
--region us-west-1
```

Babelfish 製品バージョンのパラメータを使用する

Babelfish 2.4.0 および 3.1.0 バージョンから、babelfishpg_tds.product_version という新しい Grand Unified Configuration (GUC) パラメータが導入されました。このパラメータでは、SQL Server 製品のバージョン番号を Babelfish の出力として設定できます。

パラメータは 4 つの部分からなるバージョン ID の文字列で、各部分は「.」で区切る必要があります。

構文

```
Major.Minor.Build.Revision
```

- メジャーバージョン: 11 ~ 16 の数字。
- マイナーバージョン: 0 ~ 255 の数字。
- ビルドバージョン: 0 ~ 65535 の数字。
- リビジョン: 0 および任意の正数。

Babelfish 製品バージョンのパラメータを設定する

コンソールで babelfishpg_tds.product_version パラメータを設定するには、クラスターパラメータグループを使用する必要があります。DB クラスターパラメータの変更方法の詳細については、「DB クラスターパラメータグループのパラメータの変更」を参照してください。

製品バージョンのパラメータを無効な値に設定した場合、変更は有効になりません。コンソールに新しい値が表示される場合がありますが、パラメータは以前の値を保持します。エンジンログファイルで、エラーメッセージの詳細を確認します。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster-parameter-group \
--db-cluster-parameter-group-name mydbparametergroup \
--parameters
 "ParameterName=babelfishpg_tds.product_version,ParameterValue=15.2.4000.1,ApplyMethod=immediat
```

Windows の場合:

```
aws rds modify-db-cluster-parameter-group ^
--db-cluster-parameter-group-name mydbparametergroup ^
--parameters
 "ParameterName=babelfishpg_tds.product_version,ParameterValue=15.2.4000.1,ApplyMethod=immediat
```

影響を受けるクエリとパラメータ

クエリ/パラメータ	結果	有効になるまでの時間
SELECT @@VERSION	ユーザー定義の SQL Server バージョンを返します (babelfishpg_tsql.version 値 = Default)	直ちに
SELECT SERVERPROPERTY('ProductVersion')	ユーザー定義の SQL Server バージョンを返します	直ちに
SELECT SERVERPROPERTY('ProductMajorVersion')	ユーザー定義の SQL Server バージョンのメジャーバージョンを返します	直ちに
PRELOGIN Response Message の VERSION トークン	サーバーは、ユーザー定義の SQL Server バージョンを含む PRELOGIN メッセージを返します	ユーザーが新しいセッションを作成すると有効になります
JDBC を使用する場合の LoginAck の SQLServer Version	DatabaseMetaData.getDatabaseProductVersion() は、ユーザー定義の SQL Server バージョンを返します	ユーザーが新しいセッションを作成すると有効になります

babelfishpg_tsql.version パラメータによるインターフェイス

@@VERSION の出力を設定するには、babelfishpg_tsql.version および babelfishpg_tds.product_version パラメータを使用します。次の例では、この 2 つのパラメータがどのように機能するかを示しています。

- babelfishpg_tsql.version パラメータが「default」で、babelfishpg_tds.product_version が 15.0.2000.8 の場合。
  - @@version の出力 — 15.0.2000.8。
- babelfishpg_tsql.version パラメータが 13.0.2000.8 に設定され、babelfishpg_tds.product_version パラメータが 15.0.2000.8 に設定されている場合。
  - @@version の出力 — 13.0.2000.8。

Babelfish for Aurora PostgreSQL リファレンス

トピック

Babelfish でサポートされていない機能

以下のテーブルとリストに、Babelfish で現在サポートされていない機能を示します。Babelfish のアップデートは Aurora PostgreSQL のバージョンに含まれています。詳細については、「Aurora PostgreSQL のリリースノート」を参照してください。

トピック

現在サポートされていない機能

次の表に、現在サポートされていない特定の機能に関する情報を示します。

機能または構文	説明
アセンブリモジュールと SQL 共通言語ランタイム (CLR) ルーチン	アセンブリモジュールおよび CLR ルーチンに関連する機能はサポートされていません。
列の属性	ROWGUIDCOL、SPARSE、FILESTREAM、および MASKED はサポートされていません。

機能または構文	説明
包含データベース	サーバーレベルではなくデータベースレベルで認証されたログインを含むデータベースはサポートされていません。
カーソル (更新可能)	更新可能なカーソルはサポートされていません。
カーソル (グローバル)	GLOBAL カーソルはサポートされていません。
カーソル (フェッチ動作)	次のカーソルフェッチ動作はサポートされていません: FETCH PRIOR、FIRST、LAST、ABSOLUTE、および RELATIVE
カーソルタイプの出力パラメータ	カーソルタイプの変数やパラメータは出力パラメータとしてはサポートされません (エラーが発生します)。
カーソルのオプション	SCROLL、KEYSET、DYNAMIC、FAST_FORWARD、SCROLL_LOCKS、OPTIMISTIC、TYPE_WARNING、および FOR UPDATE
データ暗号化	データ暗号化はサポートされていません。
データ層アプリケーション (DAC)	DAC パッケージ (.dacpac) ファイルまたは DAC バックアップ (.bacpac) ファイルによるデータ層アプリケーション (DAC) のインポートまたはエクスポート操作はサポートされていません。
DBCC コマンド	Microsoft SQL Server Database Console Commands (DBCC) はサポートされていません。DBCC CHECKIDENT は Babelfish 3.4.0 以降のリリースでサポートされています。
DROP IF EXISTS	この構文は、USER オブジェクトと SCHEMA オブジェクトではサポートされていません。TABLE、VIEW、PROCEDURE、FUNCTION、および DATABASE に対してサポートされています。
暗号化	組み込み関数とステートメントは、暗号化をサポートしていません。

機能または構文	説明
ENCRYPT_CLIENT_CERT 接続	クライアント証明書接続はサポートされていません。
EXECUTE AS ステートメント	このステートメントはサポートされていません。
EXECUTE AS SELF 句	この句は、関数、プロシージャ、またトリガーでサポートされていません。
EXECUTE AS USER 句	この句は、関数、プロシージャ、またトリガーでサポートされていません。
データベース名を参照する外部キー制約	データベース名を参照する外部キー制約はサポートされていません。
FORMAT	ユーザー定義型はサポートされていません。
100 を超えるパラメータを持つ関数宣言	100 を超えるパラメータを含む関数宣言はサポートされていません。
DEFAULT をパラメータ値として含む関数呼び出し	DEFAULT は、関数呼び出しでサポートされているパラメータ値ではありません。関数呼び出しのパラメータ値としての DEFAULT は、Babelfish 3.4.0 以降のリリースでサポートされています。
関数、外部定義	SQL CLR 関数を含む外部関数はサポートされていません。
グローバル一時テーブル (## で始まる名前のテーブル)	グローバル一時テーブルはサポートされていません。
グラフ機能	すべての SQL グラフ機能はサポートされていません。
先頭に複数の @ 文字が含まれる識別子 (可変またはパラメータ)	先頭に @ が複数ある識別子はサポートされていません。
@ または]] 文字を含む識別子、テーブル名、または列名	@ 記号ないし角括弧を含むテーブル名、または列名はサポートされていません。

機能または構文	説明
インラインインデックス	インラインインデックスはサポートされていません。
可変に名前が入っているプロシージャの呼び出し	可変をプロシージャ名として使用することはサポートされていません。
具体化されたビュー	マテリアライズドビューはサポートされていません。
NOT FOR REPLICATION 句	この構文は受け入れられ、無視されます。
ODBC エスケープ関数	ODBC エスケープ関数はサポートされていません。
パーティション	テーブルとインデックスのパーティショニングはサポートされていません。
DEFAULT をパラメータ値として含む関数呼び出し	DEFAULT は、サポートされているパラメータ値ではありません。関数呼び出しのパラメータ値としての DEFAULT は、Babelfish 3.4.0 以降のリリースでサポートされています。
100 を超えるパラメータを持つプロシージャ宣言	100 を超えるパラメータを含む宣言はサポートされていません。
プロシージャ、外部定義	SQL CLR プロシージャを含む外部定義のプロシージャはサポートされていません。
プロシージャバージョニング	プロシージャのバージョニングはサポートされていません。
RECOLLATE の手順	WITH RECOMPILE (DECLARE ステートメントと EXECUTE ステートメントと組み合わせて使用する場合) はサポートされていません。
リモートオブジェクト参照	4 つの部分で構成される名前を使用したプロシージャと関数の実行はサポートされていません。リモートオブジェクトでは、選択したクエリの 4 つの部分で構成されるオブジェクト名をサポートします。詳細については、「Babelfish の DB クラスターパラメータグループ設定」を参照してください。

機能または構文	説明
行レベルのセキュリティ	CREATE SECURITY POLICY およびインラインテーブル値関数を使用した行レベルのセキュリティはサポートされていません。
サービスブローカー機能	サービスブローカーの機能はサポートされていません。
SESSIONPROPERTY	サポートされていないプロパティ: ANSI_NULLS、ANSI_PADDING、ANSI_WARNINGS、ARITHABORT、CONCAT_NULL_YIELDS_NULL、および NUMERIC_ROUNDABORT
SET LANGUAGE	この構文は、english または us_english 以外の値ではサポートされていません。
SP_CONFIGURE	このシステムストアドプロシージャはサポートされていません。
SQL キーワード SPARSE	キーワード SPARSE は受け入れられ、無視されます。
テーブル値のコンストラクタ構文 (FROM 句)	サポートされていない構文は、FROM 句を使用して構築された派生テーブル用です。
一時テーブル	一時テーブルはサポートされていません。
一時プロシージャは自動的に削除されません	この関数はサポートされていません。
トリガー、外部定義	SQL 共通言語ランタイム (CLR) を含む外部関数はサポートされていません。
ストアドプロシージャ呼び出しの引用符で囲まれていない文字列値およびデフォルト値	ストアドプロシージャ呼び出しへの文字列パラメータ、および CREATE PROCEDURE の文字列パラメータのデフォルトはサポートされていません。

機能または構文	説明
WITHOUT SCHEMABINDING 句	SCHEMIBINDING なしのビューの作成はサポートされていませんが、ビューは、WITH SCHEMIBINDING が指定されているかのように作成されます。関数、プロシージャ、トリガーを作成するときに SCHEMABINDING を使用しても、無視され、エラーは表示されません。

サポートされていない設定

次の設定はサポートされていません。

- ANSI_NULL_DFLT_OFF をオンに設定
- ANSI_NULL_DFLT_ON をオフに設定
- ANSI_PADDING をオフに設定
- ANSI_WARNINGS をオフに設定
- ALITHABORT をオフに設定
- ARITHIGNORE をオンに設定
- CURSOR_CLOSE_ON_COMMIT をオンに設定
- SET NUMERIC_ROUNDABORT ON
- SET PARSEONLY ON (コマンドが正常に動作しない)
- SET FMTONLY ON (コマンドが想定どおりに動作しません。SELECT ステートメントの実行のみを抑制し、他のステートメントは抑制しません)。

サポートされていないコマンド

次のコマンドの一部の機能はサポートされていません。

- ADD SIGNATURE
- ALTER DATABAS、ALTER DATABASE SET
- BACKUP/RESTORE DATABASE/LOG
- BACPAC および DACPAC FILES RESTORE
- CREATE、ALTER、DROP AUTHORIZATION。ALTER AUTHORIZATION はデータベースオブジェクトでサポートされています。

- CREATE、ALTER、DROP AVAILABILITY GROUP
- CREATE、ALTER、DROP BROKER PRIORITY
- CREATE、ALTER、DROP COLUMN ENCRYPTION KEY
- CREATE、ALTER、DROP DATABASE ENCRYPTION KEY
- CREATE、ALTER、DROP、BACKUP CERTIFICATE
- CREATE AGGREGATE
- CREATE CONTRACT
- CHECKPOINT

サポートされていない列名と属性

次の列名はサポートされていません。

- $IDENTITY
- $ROWGUID
- IDENTITYCOL

サポートされていないデータ型

次のデータ型はサポートされていません。

- Geospatial (GEOGRAPHY と GEOMETRY)
- HIERARCHYID

サポートされていないオブジェクト型

次のオブジェクト型はサポートされていません。

- COLUMN MASTER KEY
- CREATE、ALTER EXTERNAL DATA SOURCE
- CREATE、ALTER、DROP DATABASE AUDIT SPECIFICATION
- CREATE、ALTER、DROP EXTERNAL LIBRARY
- CREATE、ALTER、DROP SERVER AUDIT
- CREATE、ALTER、DROP SERVER AUDIT SPECIFICATION

- CREATE、ALTER、DROP、OPEN/CLOSE SYMMETRIC KEY
- CREATE、DROP DEFAULT
- CREDENTIAL
- CRYPTOGRAPHIC PROVIDER
- DIAGNOSTIC SESSION
- インデックス付きビュー
- SERVICE MASTER KEY
- SYNONYM

サポートされていない関数

次の組み込み関数はサポートされていません。

集計関数

- APPROX_COUNT_DISTINCT
- CHECKSUM_AGG
- GROUPING_ID
- WITHIN GROUP 句を使用した STRING_AGG

暗号化関数

- CERTENCODED 関数
- CERTID 関数
- CERTPROPERTY 関数

メタデータ関数

- COLUMNPROPERTY
- TYPEPROPERTY
- SERVERPROPERTY 関数 — 次のプロパティはサポートされていません。
  - BuildClrVersion
  - ComparisonStyle
  - ComputerNamePhysicalNetBIOS

- HadrManagerStatus
- InstanceDefaultDataPath
- InstanceDefaultLogPath
- IsClustered
- IsHadrEnabled
- LCID
- NumLicenses
- ProcessID
- ProductBuild
- ProductBuildType
- ProductUpdateReference
- ResourceLastUpdateDateTime
- ResourceVersion
- ServerName
- SqlCharSet
- SqlCharSetName
- SqlSortOrder
- SqlSortOrderName
- FilestreamShareName
- FilestreamConfiguredLevel
- FilestreamEffectiveLevel

セキュリティ関数

- CERTPRIVATEKEY
- LOGINPROPERTY

ステートメント、演算子、その他の関数

- EVENTDATA 関数
- GET_TRANSMISSION_STATUS
- OPENXML

サポートされていない構文

次の構文はサポートされていません。

- ALTER DATABASE
- ALTER DATABASE SCOPED CONFIGURATION
- ALTER DATABASE SCOPED CREDENTIAL
- ALTER DATABASE SET HADR
- ALTER FUNCTION
- ALTER INDEX
- ALTER PROCEDURE statement
- ALTER SCHEMA
- ALTER SERVER CONFIGURATION
- ALTER SERVICE、BACKUP/RESTORE SERVICE MASTER KEY 句
- ALTER VIEW
- BEGIN CONVERSATION TIMER
- BEGIN DISTRIBUTED TRANSACTION
- BEGIN DIALOG CONVERSATION
- BULK INSERT
- CREATE COLUMNSTORE INDEX
- CREATE EXTERNAL FILE FORMAT
- CREATE EXTERNAL TABLE
- CREATE、ALTER、DROP APPLICATION ROLE
- CREATE、ALTER、DROP ASSEMBLY
- CREATE、ALTER、DROP ASYMMETRIC KEY
- CREATE、ALTER、DROP CREDENTIAL
- CREATE、ALTER、DROP CRYPTOGRAPHIC PROVIDER
- CREATE、ALTER、DROP ENDPOINT
- CREATE、ALTER、DROP EVENT SESSION
- CREATE、ALTER、DROP EXTERNAL LANGUAGE
- CREATE、ALTER、DROP EXTERNAL RESOURCE POOL
- CREATE、ALTER、DROP FULLTEXT CATALOG

- CREATE、ALTER、DROP FULLTEXT INDEX
- CREATE、ALTER、DROP FULLTEXT STOPLIST
- CREATE、ALTER、DROP MESSAGE TYPE
- CREATE、ALTER、DROP、OPEN/CLOSE、BACKUP/RESTORE MASTER KEY
- CREATE、ALTER、DROP PARTITION FUNCTION
- CREATE、ALTER、DROP PARTITION SCHEME
- CREATE、ALTER、DROP QUEUE
- CREATE、ALTER、DROP RESOURCE GOVERNOR
- CREATE、ALTER、DROP RESOURCE POOL
- CREATE、ALTER、DROP ROUTE
- CREATE、ALTER、DROP SEARCH PROPERTY LIST
- CREATE、ALTER、DROP SECURITY POLICY
- CREATE、ALTER、DROP SELECTIVE XML INDEX 句
- CREATE、ALTER、DROP SERVICE
- CREATE, ALTER, DROP SPATIAL INDEX
- CREATE、ALTER、DROP TYPE
- CREATE、ALTER、DROP XML INDEX
- CREATE、ALTER、DROP XML SCHEMA COLLECTION
- CREATE/DROP RULE
- CREATE、DROP WORKLOAD CLASSIFIER
- CREATE、ALTER、DROP WORKLOAD GROUP
- ALTER TRIGGER
- CREATE TABLE… GRANT 句
- CREATE TABLE… IDENTY 句
- CREATE USER – この構文はサポートされていません。PostgreSQL ステートメント CREATE USER は、SQL Server CREATE USER 構文と同等のユーザーを作成しません。詳細については、「Babelfish での T-SQL の違い」を参照してください。
- DENY
- END、MOVE CONVERSATION
- AS LOGIN または AT オプションで EXECUTE
- GET CONVERSATION GROUP

- GROUP BY ALL 句
- GROUP BY CUBE 句
- GROUP BY ROLLUP 句
- INSERT..。DEFAULT VALUES
- MERGE
- READTEXT
- REVERT
- SELECT PIVOT (ビュー定義、共通テーブル式、または結合で使用される場合を除き、3.4.0 以降のリリースでサポート)/UNPIVOT
- SELECT TOP x PERCENT WHERE x <> 100
- SELECT TOP..。WITH TIES
- SELECT… FOR BROWSE
- SELECT… FOR XML AUTO
- SELECT… FOR XML EXPLICIT
- SEND
- SET DATEFORMAT
- SET DEADLOCK_PRIORITY
- SET FMTONLY
- SET FORCEPLAN
- SET NUMERIC_ROUNDABORT ON
- SET OFFSETS
- SET REMOTE_PROC_TRANSACTIONS
- SET SHOWPLAN_TEXT
- SET SHOWPLAN_XML
- SET STATISTICS
- SET STATISTICS PROFILE
- SET STATISTICS TIME
- SET STATISTICS XML
- SHUTDOWN statement
- UPDATE STATISTICS
- UPDATETEXT

- Using EXECUTE to call a SQL function
- VIEW… CHECK OPTION clause
- VIEW..。VIEW_METADATA clause
- WAITFOR DELAY
- WAITFOR TIME
- WAITFOR、RECEIVE
- WITH XMLNAMESPACES construct
- WRITETEXT
- XPATH 式

バージョンごとに Babelfish でサポートされている機能

次の表に、さまざまな Babelfish バージョンでサポートされている T-SQL 機能を示します。サポートされていない機能のリストについては、Babelfish でサポートされていない機能を参照してください。Babelfish のリリースの詳細については、「Aurora PostgreSQL リリースノート」を参照してください。

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
4 parts object name references for SELECT statements	✓	✓	✓	✓	✓	✓	–	✓	✓	–	–	–	–	–	–
AS keyword in CREATE FUNCTION	✓	–	✓	–	–	–	–	–	–	–	–	–	–	–	–
ALTER AUTHORIZATION	✓	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
syntax to change database owner															
ALTER ROLE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ALTER USER...WITH LOGIN	✓	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–
AT TIME ZONE clause	✓	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–
Babelfish instance as a linked server	✓	✓	✓	✓	✓	✓	–	✓	✓	–	–	–	–	–	–

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
Comparison operators !< and !>	✓	–	✓	–	–	–	–	–	–	–	–	–	–	–	–
CREATE Instead of Triggers (DML) on SQL Server Views	✓	–	✓	–	–	–	–	–	–	–	–	–	–	–	–
CREATE ROLE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
CREATE TRIGGER	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Create unique indexes	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
Cross-database procedure execution	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
Cross-database references SELECT,SELECT..INTO, INSERT, UPDATE, DELETE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
Cursor ✓ ty ped parameter s for input parameter s only (not output)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Data migration using the bcp client utility	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
Datatypes TIMESTAMP, ROWVERSION (for usage information, see Features with limited implementation	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
DEFAULT keyword in calls to stored procedures and functions	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–
DBCC CHECKIDENT	✓	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–
DROP DATABASE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
DROP IF EXISTS (for SCHEMA, DATABASE, and USER objects)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
DROP INDEX index ON schema.table	✓	–	✓	–	–	–	–	–	–	–	–	–	–	–	–
DROP INDEX schema.table.index	✓	–	✓	–	–	–	–	–	–	–	–	–	–	–	–
DROP ROLE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
ENABLE/DISABLE TRIGGER	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–
FULL TEXT SEARCH	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–	–
Full Text Search with CONTAINS clause	✓	–	–	–	–	–	–	–	–	–	–	–	–	–	–
GRANT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Geometry and Geography spatial datatypes	✓	–	✓	–	–	–	–	–	–	–	–	–	–	–	–

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
GUC babelfish pg_tds.product_version	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
Identifiers with leading dot character	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
INSTEAD OF triggers on tables	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
INSTEAD OF triggers on views	✓	–	–	–	–	–	–	–	–	–	–	–	–	–	–
KILL	✓	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
PIVOT (supported from 3.4.0 and higher releases except when used in a view definition, a common table expression, or a join)	✓	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–
REVOKE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
SELECT... OFFSET... FETCH clauses	✓	–	✓	–	–	–	–	–	–	–	–	–	–	–	–
SELECT FOR JSON AUTO	✓	–	✓	–	–	–	–	–	–	–	–	–	–	–	–
SET BABELFISH_SHOWPLAN_ALL ON (and OFF)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SET BABELFISH_STATISTICS PROFILE ON (OFF)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
SET CONTEXT_INFO	✓	✓	✓	✓	✓	✓	–	✓	✓	–	–	–	–	–	–
SET LOCK_TIMEOUT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SET NO_BROWSETABLE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
SET rowcount	✓	✓	✓	✓	✓	✓	–	✓	✓	–	–	–	–	–	–
SET SHOWPLAN_ALL	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
SET STATISTICS IO	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ	✓	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE	✓	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–
SET TRANSACTION ISOLATION LEVEL syntax	✓	–	✓	–	–	–	–	–	–	–	–	–	–	–	–

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
SSMS Connecting with the object explorer connection dialog	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SSMS Data migration with the Import/Export Wizard	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
SSMS: Partial support for the object explorer	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
STDEV	✓	✓	✓	✓	✓	–	–	✓	✓	–	–	–	–	–	–
STDEVP	✓	✓	✓	✓	✓	–	–	✓	✓	–	–	–	–	–	–
Triggers with multiple DML actions can reference transition tables	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
T-SQL hints (join methods, index usage, MAXDOP)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
T-SQL square bracket syntax with the LIKE predicate	✓	✓	✓	✓	✓	–	–	✓	✓	–	–	–	–	–	–
VAR	✓	✓	✓	✓	✓	✓	–	✓	✓	–	–	–	–	–	–
VARP	✓	✓	✓	✓	✓	✓	–	✓	✓	–	–	–	–	–	–

Aurora and PostgreSQL features:

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
Aurora ML services	✓	–	✓	–	–	–	–	–	–	–	–	–	–	–	–
Database authentication with Kerberos using AWS Directory Service	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
Dump and restore	✓	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–
pg_stat_statement extension	✓	✓	✓	✓	✓	–	–	✓	✓	–	–	–	–	–	–
pgvector	✓	–	✓	–	–	–	–	–	–	–	–	–	–	–	–

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
Zero-down time patching (ZDP)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
T-SQL Built-in functions:															
APP_NAME	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
ATN2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
CHARINDEX	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
CHOOSE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
COL_LENGTH	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–
COL_NAME	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–
COLUMNS_UPDATED	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
COLUMNPROPERTY (CharMaxLen, AllowsNull only)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
CONCAT_WS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
CONTEXT_INFO	✓	✓	✓	✓	✓	–	✓	✓	–	–	–	–	–	–	–
CURSOR_STATUS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
DATABASE_PRINCIPAL_ID	✓	✓	✓	✓	✓	–	✓	✓	–	–	–	–	–	–	–
DATEADD	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
DATEDIFF	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
DATEDIFF_BIG	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
DATEFROMPARTS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
DATENAME	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
DATEPART	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
DATETIMEFROMPARTS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
DATETIME2FROMPARTS	✓	✓	✓	✓	✓	–	✓	✓	–	–	–	–	–	–	–
DATETIMEOFFSETFROMPARTS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
DATETRUNC	✓	✓	✓	–	–	–	✓	–	–	–	–	–	–	–	–
DATE_BUCKET	✓	✓	✓	–	–	–	✓	–	–	–	–	–	–	–	–
EOMONTH	✓	✓	✓	–	–	–	✓	–	–	–	–	–	–	–	–
EXECUTE AS CALLER	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
fn_listextendedproperty	✓	✓	✓	✓	✓	–	–	✓	✓	–	–	–	–	–	–
FOR JSON	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
FULLTEXTSERVICEPROPERTY	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
HAS_DBACCESS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
HAS_PERMS_BY_NAME	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
HOST_NAME	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
HOST_ID	✓	✓	✓	✓	✓	–	–	✓	✓	–	–	–	–	–	–
IDENTITY	✓	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–
IS_MEMBER	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
IS_ROLEMEMBER	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
IS_SRVROLEMEMBER	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ISJSON	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
JSON_MODIFY	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
JSON_QUERY	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
JSON_VALUE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NEXT VALUE FOR	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
OBJECT_DEFINITION	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
OBJECT_SCHEMA_NAME	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
OPENJSON	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
OPENQUERY	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
ORIGINAL_LOGIN	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
PARSENAME	✓	✓	✓	✓	✓	–	–	✓	✓	–	–	–	–	–	–
PATINDEX	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ROWCOUNT_BIG	✓	✓	✓	✓	✓	✓	–	✓	✓	–	–	–	–	–	–
SCHEMA_NAME	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SESSION_CONTEXT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
SESSION_USER	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SID_BINARY (returns NULL always)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
SMALLDATETIMEFROMPARTS	✓	✓	✓	✓	✓	–	–	✓	✓	✓	–	–	–	–	–

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
SQUARE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
STR	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
STRING_AGG	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–
STRING_SPLIT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SUSER_SID	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SUSER_SNAME	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SWITCHOFFSET	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–
SYSTEM_USER	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
TIMEFROMPARTS	✓	✓	✓	✓	✓	–	✓	✓	–	–	–	–	–	–	–
TODATETIMEOFFSET	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–
TO_CHAR	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
TRIGGER_NESTLEVEL (without arguments only)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
TRY_CONVERT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
TYPE_ID	✓	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–
TYPE_NAME	✓	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–
UPDATE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
T-SQL INFORMATION_SCHEMA catalogs															
CHECK_CONSTRAINTS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
COLUMN_DOMAIN_USAGE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
COLUMNS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
CONSTRAINT_COLUMN_USAGE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
DOMAINS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
KEY_COLUMN_USAGE	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–
ROUTINES	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
TABLES	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
TABLE_CONSTRAINTS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
VIEWS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
T-SQL System-defined @@ variables:															
@@CURSOR_ROWS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@DATEFIRST	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@DBTS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@ERROR	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
@@ERROR=213	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@FETCH_STATUS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@IDENTITY	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@LANGUAGE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@LOCK_TIMEOUT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@MAX_CONNECTIONS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@MAX_PRECISION	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@MICROSOFTVERSION	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@NESTLEVEL	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@PROCID	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
@@ROWCOUNT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@SERVERNAME	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@SERVICENAME	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@SPID	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@TRANCOUNT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
@@VERSION (note format difference as described in Babelfish での T-SQL の違い.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
T-SQL System stored procedures:															
sp_addextendedproperty	✓	✓	✓	✓	–	–	✓	✓	–	–	–	–	–	–	
sp_addlinkedserver	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
sp_addlinkedsrvlogin	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
sp_addrole	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
sp_addrolemember	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
sp_babelfish_volatility	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
sp_column_privileges	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_columns	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_columns_100	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_columns_managed	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_cursor	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
sp_cursor_list	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_cursorclose	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_cursorexecute	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_cursorfetch	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_cursoropen	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_cursoroption	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_cursorprepare	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_cursorprepexec	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_cursorunprepare	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
sp_databases	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_datatype_info	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_datatype_info_100	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_describe_cursor	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_describe_first_result_set	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_describe_undeclared_parameters	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_dropextendedproperty	✓	✓	✓	✓	–	–	✓	✓	–	–	–	–	–	–	–

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
sp_droplinkedsrvlogin	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
sp_droprole	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	
sp_droprolemember	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	
sp_dropserver	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	
sp_enum_oledb_providers	✓	✓	✓	✓	–	–	✓	✓	–	–	–	–	–	–	
sp_execute	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_execute_postgresql(CREATE, ALTER, DROP)	✓	✓	✓	✓	–	–	✓	✓	–	–	–	–	–	–	

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
sp_executesql	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_fkeys	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_getapplock	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_helpdb	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_helpdbfixedrole	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
sp_helplinkedsrvlogin	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
sp_helprole	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_helprolemember	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_helpsrvrolemember	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
sp_helpuser	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_linkedservers	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
sp_oledb_ro_usrname	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_pkeys	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_prefix	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_prepare	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_procedure_params_100_managed	–	✓	–	–	–	–	–	–	–	–	–	–	–	–	–
sp_releaseapplock	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_rename	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
sp_server_option(connect_timeout option)	✓	✓	✓	✓	✓	–	–	✓	✓	–	–	–	–	–	–
sp_set_session_context	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
sp_special_columns	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_sproc_columns	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_sproc_columns_100	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_statistics	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_statistics_100	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
sp_stored_procedures	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_table_privileges	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_tablecollations_100	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_tables	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_testlinkedserver	✓	✓	✓	✓	–	–	✓	✓	–	–	–	–	–	–	–
sp_unprepare	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_updateextendedproperty	✓	✓	✓	✓	–	–	✓	✓	–	–	–	–	–	–	–
sp_who	✓	✓	✓	✓	–	–	✓	✓	–	–	–	–	–	–	–
xp_qv	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
T-SQL Properties supported on the CONNECTIONPROPERTY system function															
auth_scheme	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
client_net_address	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
local_net_address	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
local_tcp_port	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
net_transport	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
protocol_type	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
physical_net_transport	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
T-SQL Properties supported on the OBJECTPROPERTY system function															
IsInlineFunction	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
IsScalarFunction	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
IsTableFunction	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
T-SQL Properties supported on the SERVERPROPERTY function															
Babelfish	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Collation	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
CollationID	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Edition	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
EditionID	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
EngineEdition	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
InstanceName	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
IsAdvancedAnalyticsInstalled	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
IsBigDataCluster	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
IsFullTextInstalled	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
IsIntegratedSecurityOnly	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
IsLocalDB	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
IsPolyBaseInstalled	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
IsSingleUser	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
IsXTPSupported	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
Japanese_CI_AI	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Japanese_CI_AS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Japanese_CS_AS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
LicenseType	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MachineName	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
ProductLevel	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
ProductMajorVersion	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ProductMinorVersion	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ProductUpdateLevel	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
ProductVersion	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ServerName	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SQL Server views supported by Babelfish															
information_schema.key_column_usage	✓	✓	✓	–	–	–	✓	–	–	–	–	–	–	–	–
information_schema.routines	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	
information_schema.schemata	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	
information_schema.sequences	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
sys.all_columns	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.all_objects	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.all_parameters	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
sys.all_sql_modules	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.all_views	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.columns	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.configurations	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.data_spaces	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.database_files	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
sys.database_mirroring	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.database_principals	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.database_role_members	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.databases	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.dm_exec_connections	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.dm_exec_sessions	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.dm_hadr_database_replica_states	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
sys.dm_os_host_info	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.endpoints	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.extended_properties	✓	✓	✓	✓	–	–	✓	✓	–	–	–	–	–	–	–
sys.indexes	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.schemas	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.server_principals	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.server_role_members	✓	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–
sys.sql_modules	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
sys.sysconfigures	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.syscurconfigs	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.syslogins	✓	✓	✓	✓	✓	–	✓	✓	–	–	–	–	–	–	–
sys.sysprocesses	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.sysusers	✓	✓	✓	✓	✓	–	✓	✓	–	–	–	–	–	–	–
sys.table_types	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.tables	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.types	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
sys.xml_schema_collections	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

T-SQL の機能または構文	4.1.0	4.0.0	3.5.0	3.4.0	3.3.0	3.2.0	3.1.0	2.8.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0
syslanguages	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sysobjects.crdate	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	

Babelfish for Aurora PostgreSQL プロシージャリファレンス

概要

Babelfish for Aurora PostgreSQL を実行している Amazon RDS DB インスタンスでは、以下のプロシージャを使用してクエリのパフォーマンスを向上させることができます。

- sp_babelfish_volatility
- sp_execute_postgresql

sp_babelfish_volatility

PostgreSQL 関数の変動性により、オプティマイザはクエリをより適切に実行でき、特定の句の一部で使用すると、クエリのパフォーマンスに大きな影響を与えます。

構文

```
sp_babelfish_volatility 'function_name', 'volatility'
```

引数

function_name (オプション)

この引数の値は、schema_name.function_name として 2 つの部分からなる名前で指定するか、function_name のみを指定することができます。function_name のみを指定した場合、スキーマ名は、現在のユーザーのデフォルトスキーマです。

変動性 (オプション)

PostgreSQL の変動性の有効な値は、stable、volatile、または immutable です。詳細については、「https://www.postgresql.org/docs/current/xfunc-volatility.html」を参照してください。

> **Note**
>
> sp_babelfish_volatility が複数の定義を持つ function_name で呼び出されると、エラーをスローします。

結果セット

パラメータが指定されていない場合、結果セットは schemaname、functionname、volatility 列に表示されます。

使用に関する注意事項

PostgreSQL 関数の変動性により、オプティマイザはクエリをより適切に実行でき、特定の句の一部で使用すると、クエリのパフォーマンスに大きな影響を与えます。

例

次の例は、簡単な関数を作成する方法を示し、後でさまざまな方法を使用してこれらの関数で sp_babelfish_volatility を使用する方法を示します。

```
1> create function f1() returns int as begin return 0 end
2> go
```

```
1> create schema test_schema
2> go
```

```
1> create function test_schema.f1() returns int as begin return 0 end
2> go
```

次の例は、関数の変動性を示しています。

```
1> exec sp_babelfish_volatility
2> go

schemaname  functionname volatility
----------- ------------ ----------
dbo         f1           volatile
test_schema f1           volatile
```

次の例は、関数の変動性を変更する方法を示しています。

```
1> exec sp_babelfish_volatility 'f1','stable'
2> go
1> exec sp_babelfish_volatility 'test_schema.f1','immutable'
2> go
```

function_name のみを指定すると、その関数のスキーマ名、関数名、および変動性が表示されます。次の例は、値を変更した後の関数の変動性を示しています。

```
1> exec sp_babelfish_volatility 'test_schema.f1'
2> go

schemaname  functionname volatility
----------- ------------ ----------
test_schema f1           immutable
```

```
1> exec sp_babelfish_volatility 'f1'
2> go

schemaname  functionname volatility
----------- ------------ ----------
dbo         f1           stable
```

引数を指定しなかった場合、現在のデータベースに存在する関数のリスト (スキーマ名、関数名、変動性) が表示されます。

```
1> exec sp_babelfish_volatility
2> go

schemaname  functionname volatility
----------- ------------ ----------
dbo         f1           stable
test_schema f1           immutable
```

sp_execute_postgresql

T-SQL エンドポイントから PostgreSQL ステートメントを実行できます。これにより、T-SQL ポートを終了しなくてもこれらのステートメントを実行できるため、アプリケーションがシンプルになります。

構文

```
sp_execute_postgresql [ @stmt = ] statement
```

引数

[@stmt] ステートメント

この引数のデータ型は varchar です。この引数は PG ダイアレクトステートメントを受け付けます。

> **Note**
>
> 引数として渡せる PG ダイアレクトステートメントは 1 つだけです。それ以外の場合は次のエラーが発生します。

```
1>exec sp_execute_postgresql 'create extension pg_stat_statements; drop extension
 pg_stat_statements'
2>go
```

```
Msg 33557097, Level 16, State 1, Server BABELFISH, Line 1
expected 1 statement but got 2 statements after parsing
```

使用に関する注意事項

CREATE EXTENSION

新しい拡張機能作成して現在のデータベースに読み込みます。

```
1>EXEC sp_execute_postgresql 'create extension [ IF NOT EXISTS ] <extension name>
 [ WITH ] [SCHEMA schema_name] [VERSION version]';
2>go
```

次の例は拡張機能の使用方法を示しています。

```
1>EXEC sp_execute_postgresql 'create extension pg_stat_statements with schema sys
 version "1.10"';
2>go
```

次のコマンドを使用して、拡張機能オブジェクトにアクセスします。

```
1>select * from pg_stat_statements;
2>go
```

> **Note**
>
> 拡張機能の作成時にスキーマ名を明示的に指定しない場合、エクステンションはデフォルトでパブリックスキーマにインストールされます。拡張機能オブジェクトにアクセスするには、以下のようにスキーマ修飾子を指定する必要があります。

```
1>select * from [public].pg_stat_statements;
2>go
```

サポートされている拡張機能

Aurora PostgreSQL で利用できる以下の拡張機能は、Babelfish で動作します。

- `pg_stat_statements`
- `tds_fdw`
- `fuzzystrmatch`

制限事項

- 拡張機能をインストールするには、ユーザーが T-SQL では sysadmin ロールを、postgres では rds_superuser ロールを持っている必要があります。
- 拡張機能は、ユーザーが作成したスキーマにはインストールできません。また、master、tempdb、msdb データベースの dbo スキーマやゲストスキーマにもインストールできません。
- CASCADE オプションはサポートされていません。

ALTER EXTENSION

ALTER 拡張機能を使用して新しい拡張機能のバージョンにアップグレードできます。

```
1>EXEC sp_execute_postgresql 'alter extension <extension name> UPDATE TO
 <new_version>';
2>go
```

制限事項

- 拡張機能のバージョンは ALTER Extension 拡張機能を使用してのみアップグレードできます。他のオペレーションはサポートされていません。

DROP EXTENSION

指定した拡張機能を削除します。`if exists` または `restrict` オプションを使用して拡張機能を削除することもできます。

```
1>EXEC sp_execute_postgresql 'drop extension <extension name>';
2>go
```

制限事項

- CASCADE オプションはサポートされていません。

Amazon Aurora PostgreSQL の管理

次のセクションでは、Amazon Aurora PostgreSQL DB クラスターのパフォーマンスとスケーリングの管理について説明します。また、他のメンテナンスタスクについても説明します。

トピック

- Aurora PostgreSQL DB インスタンスのスケーリング
- Aurora PostgreSQL DB インスタンスへの最大接続数
- Aurora PostgreSQL 用の一時ストレージの制限
- Huge pages for Aurora PostgreSQL
- 障害挿入クエリを使用した Amazon Aurora PostgreSQL のテスト
- Aurora PostgreSQL DB クラスターのボリュームステータスの表示
- stats_temp_directory の RAM ディスクを特定する
- PostgreSQL による一時ファイルの管理

Aurora PostgreSQL DB インスタンスのスケーリング

Aurora PostgreSQL DB インスタンスは、インスタンススケーリングと読み取りスケーリングの 2 つの方法でスケールできます。読み取りスケーリングの詳細については、「読み取りのスケーリング」を参照してください。

DB クラスター内の各 DB インスタンスで DB インスタンスクラスを変更することで、Aurora PostgreSQL DB クラスターのスケーリングが行えます。Aurora PostgreSQL は、Aurora 用に最適化された DB インスタンスクラスを複数サポートしています。サイズが 40 テラバイト (TB) より大きい Aurora クラスターには、db.t2 または db.t3 インスタンスクラスを使用しないでください。

> **Note**
>
> T DB インスタンスクラスは、開発サーバーおよびテストサーバー、またはその他の本稼働以外のサーバーにのみ使用することをお勧めします。T インスタンスクラスの詳細については、「DB インスタンスクラスタイプ」を参照してください。

スケーリングは、瞬時には行われません。別の DB インスタンスクラスへの変更を完了するには、15 分以上かかることがあります。このアプローチを使用して DB インスタンスクラスを変更する場合は、ユーザーに影響を与えないように、次回のスケジュールされたメンテナンス期間中に (すぐにではなく) 変更を適用します。

DB インスタンスクラスを直接変更する代わりに、Amazon Aurora の高可用性機能を使用してダウンタイムを最小限に抑えることができます。まず、Aurora レプリカをクラスターに追加します。レプリカを作成するときに、クラスターに使用する DB インスタンスクラスのサイズを選択します。Aurora レプリカがクラスターと同期されてから、新しく追加されたレプリカにフェイルオーバーします。詳細については、「Aurora レプリカ」および「Amazon Aurora PostgreSQL による高速フェイルオーバー」を参照してください。

Aurora PostgreSQL でサポートされている DB インスタンスクラスの詳細な仕様については、「DB インスタンスクラスでサポートされている DB エンジン」を参照してください。

Aurora PostgreSQL DB インスタンスへの最大接続数

Aurora PostgreSQL DB クラスターは、DB インスタンスクラスとその使用可能なメモリに基づいてリソースを割り当てます。DB クラスターへの各接続は、メモリや CPU など、これらのリソースの増分量を消費します。接続ごとに消費されるメモリは、クエリの種類、カウント、一時テーブルが使用されているかどうかによって異なります。アイドル接続でもメモリと CPU を消費します。これは、クエリが接続で実行されると、各クエリに対してより多くのメモリが割り当てられ、処理が停止しても完全に解放されないためです。したがって、アプリケーションがアイドル状態の接続を保持していないことを確認することをお勧めします。これらの各接続はリソースを浪費し、パフォーマンスに悪影響を及ぼします。詳細については、「アイドル状態の PostgreSQL 接続によって消費されるリソース」を参照してください。

Aurora PostgreSQL DB インスタンスによって許可されている接続の最大数は、その DB インスタンスのパラメータグループで指定された `max_connections` パラメータ値によって決まります。`max_connections` パラメータの理想的な設定は、アプリケーションが必要とするすべてのクライアント接続をサポートするもので、未使用の接続が余剰になっておらず、さらに AWS オートメーションをサポートするために少なくとも 3 つの接続があるものです。`max_connections` パラメータ設定を変更する前に、以下を考慮することをお勧めします。

- `max_connections` 値が小さすぎる場合、クライアントが接続を試みるときに Aurora PostgreSQL DB インスタンスには十分な接続が使用可能でない可能性があります。このような場合は、`psql` を使用して接続を試みると、次のようなエラーメッセージが表示されます。

```
psql: FATAL: remaining connection slots are reserved for non-replication superuser
 connections
```

- max_connections 値が実際に必要な接続数を超えている場合、未使用の接続によってパフォーマンスが低下する可能性があります。

デフォルト値である max_connections は、次の Aurora PostgreSQL LEAST 関数から派生されます。

```
LEAST({DBInstanceClassMemory/9531392},5000).
```

max_connections の値を変更する場合は、カスタム DB クラスターパラメータグループを作成して、その値を変更する必要があります。カスタム DB パラメータグループをクラスターに適用した後、新しい値が有効になるように、プライマリインスタンスを再起動してください。詳細については、Amazon Aurora PostgreSQL のパラメータおよびDB クラスターのパラメータグループの作成を参照してください。

> **Tip**
>
> アプリケーションが頻繁に接続を開いたり閉じたりする場合や、長時間の接続を多数開いたままにする場合は、Amazon RDS Proxy の使用を推奨します。RDS Proxy は、接続プーリングを使用してデータベース接続を安全かつ効率的に共有する、フルマネージドの高可用性データベースプロキシです。RDS Proxy の詳細については、Amazon RDS Proxy for Aurora の使用 を参照してください。

Aurora Serverless v2 インスタンスによるこのパラメータの処理方法については、「Aurora Serverless v2 の最大接続数」を参照してください。

Aurora PostgreSQL 用の一時ストレージの制限

Aurora PostgreSQL は、Aurora ストレージサブシステムにテーブルとインデックスを格納します。非永続的な一時ファイル用として、Aurora PostgreSQL は別の一時ストレージを使用します。これには、クエリ処理中の大きなデータセットのソートや、インデックスの作成オペレーションなどの目的に使用するファイルが含まれます。詳細については、記事「Aurora PostgreSQL 互換のインスタンスでローカルストレージの問題のトラブルシューティングを行う方法を教えてください」を参照してください。

これらのローカルストレージボリュームは、Amazon Elastic Block Store によってバックアップされ、より大きな DB インスタンスクラスを使用することで拡張できます。ストレージの詳細については、「Amazon Aurora ストレージと信頼性」を参照してください。NVMe 対応のインスタンスタイプと Aurora Optimized Reads 対応の一時オブジェクトを使用して、一時オブジェクトのローカルストレージを増やすこともできます。詳細については、「Amazon Optimized Reads による Aurora PostgreSQL のクエリパフォーマンスの向上」を参照してください。

> **Note**
>
> db.r5.2xlarge から db.r5.4xlarge へなど、DB インスタンスをスケーリングすると、`storage-optimization` イベントが表示される可能性があります。

次の表は、Aurora PostgreSQL DB インスタンスクラス別に使用可能な一時ストレージの最大量を示しています。Aurora の DB インスタンスクラスサポートの詳細については、「Aurora DB インスタンスクラス」を参照してください。

DB インスタンスクラス	使用可能な一時ストレージの最大量 (GiB)
db.x2g.16xlarge	1829
db.x2g.12xlarge	1606
db.x2g.8xlarge	1071
db.x2g.4xlarge	535
db.x2g.2xlarge	268
db.x2g.xlarge	134
db.x2g.large	67
db.r7g.16xlarge	1008
db.r7g.12xlarge	756
db.r7g.8xlarge	504
db.r7g.4xlarge	252

DB インスタンスクラス	使用可能な一時ストレージの最大量 (GiB)
db.r7g.2xlarge	126
db.r7g.xlarge	63
db.r7g.large	32
db.r6g.16xlarge	1008
db.r6g.12xlarge	756
db.r6g.8xlarge	504
db.r6g.4xlarge	252
db.r6g.2xlarge	126
db.r6g.xlarge	63
db.r6g.large	32
db.r6i.32xlarge	1829
db.r6i.24xlarge	1500
db.r6i.16xlarge	1008
db.r6i.12xlarge	748
db.r6i.8xlarge	504
db.r6i.4xlarge	249
db.r6i.2xlarge	124
db.r6i.xlarge	62
db.r6i.large	31
db.r5.24xlarge	1500

DB インスタンスクラス	使用可能な一時ストレージの最大量 (GiB)
db.r5.16xlarge	1008
db.r5.12xlarge	748
db.r5.8xlarge	504
db.r5.4xlarge	249
db.r5.2xlarge	124
db.r5.xlarge	62
db.r5.large	31
db.r4.16xlarge	960
db.r4.8xlarge	480
db.r4.4xlarge	240
db.r4.2xlarge	120
db.r4.xlarge	60
db.r4.large	30
db.t4g.large	16.5
db.t4g.medium	8.13
db.t3.large	16
db.t3.medium	7.5

> **Note**
>
> NVMe 対応のインスタンスタイプでは、利用可能な一時スペースを NVMe の合計サイズまで増やすことができます。詳細については、「Amazon Optimized Reads による Aurora PostgreSQL のクエリパフォーマンスの向上」を参照してください。

DB インスタンスで使用できる一時ストレージをモニタリングするには、`FreeLocalStorage` CloudWatch メトリクスを使用できます。詳細については、「Amazon Aurora の Amazon CloudWatch メトリクス」を参照してください。(これは Aurora Serverless v2 には適用されません。)

一部のワークロードでは、オペレーションを実行しているプロセスに割り当てるメモリ量を増やすことで、一時ストレージの量を減らすことができます。オペレーションに使用できるメモリを増やすには、PostgreSQL パラメータの work_mem または maintenance_work_mem の値を増やします。

Huge pages for Aurora PostgreSQL

Huge pages はメモリ管理機能です。DB インスタンスが、共有バッファで使用されるような、大きく連続したメモリチャンクで動作しているときのオーバーヘッドを軽減します。この PostgreSQL 機能は、現在利用可能なすべての Aurora PostgreSQL バージョンでサポートされています。

`Huge_pages` パラメータは、t3.medium,db.t3.large,db.t4g.medium,db.t4g.large インスタンスクラス以外のすべての DB インスタンスクラスについて、デフォルトでオンになっています。サポートされている Aurora PostgreSQL のインスタンスクラスでは、`huge_pages` パラメータ値を変更したり、この機能をオフにしたりすることはできません。

障害挿入クエリを使用した Amazon Aurora PostgreSQL のテスト

障害挿入クエリを使用して、Aurora PostgreSQL DB クラスターの耐障害性をテストできます。フォールト挿入クエリは、SQL コマンドとして Amazon Aurora インスタンスに発行されます。フォールトインジェクションクエリを使用すると、インスタンスをクラッシュさせて、フェイルオーバーと復旧をテストできます。また、Aurora レプリカの障害、ディスク障害、ディスクの輻輳をシミュレートできます。フォールトインジェクションクエリは、以下のように、使用可能なすべての Aurora PostgreSQL バージョンでサポートされています。

- Aurora PostgreSQL バージョン 12、13、14 以降
- Aurora PostgreSQL バージョン 11.7 以降
- Aurora PostgreSQL バージョン 10.11 以降

トピック

障害挿入クエリでクラッシュを指定すると、Aurora PostgreSQL DB インスタンスのクラッシュが強制的に実行されます。その他の障害挿入クエリでは、障害イベントのシミュレーションが実行されますが、そのイベントは発生しません。障害挿入クエリを送信すると、障害イベントのシミュレーションが発生する時間も指定されます。

Aurora レプリカのエンドポイントに接続することによって、Aurora レプリカインスタンスの 1 つに障害挿入クエリを送信できます。詳細については、「Amazon Aurora 接続管理」を参照してください。

インスタンスのクラッシュのテスト

障害挿入クエリ関数 `aurora_inject_crash()` を使用して、Aurora PostgreSQL インスタンスのクラッシュを強制的に発生させることができます。

この障害挿入クエリでは、フェイルオーバーが発生しません。フェイルオーバーをテストする場合、RDS コンソールで DB クラスターの [Failover] (フェイルオーバー) インスタンスアクションを選択するか、AWS CLI コマンドの failover-db-cluster、または RDS API の FailoverDBCluster オペレーションを使用します。

構文

```
SELECT aurora_inject_crash ('instance' | 'dispatcher' | 'node');
```

オプション

この障害挿入クエリでは、次のクラッシュタイプのいずれかを指定できます。クラッシュタイプでは大文字と小文字は区別されません。

インスタンス

Amazon Aurora インスタンスの PostgreSQL 互換データベースのクラッシュがシミュレートされます。

'ディスパッチャー'

Aurora DB クラスターのプライマリインスタンスにあるディスパッチャーのクラッシュがシミュレートされます。ディスパッチャー は Amazon Aurora DB クラスターのクラスターボリュームに対して更新を書き込みます。

ノード

PostgreSQL 互換データベースと Amazon Aurora インスタンスのディスパッチャーの両方のクラッシュがシミュレートされます。

Aurora レプリカの障害のテスト

障害挿入クエリ関数 `aurora_inject_replica_failure()` を使用して、Aurora レプリカの障害をシミュレートできます。

Aurora レプリカの障害により、指定した時間間隔で指定した割合だけ Aurora レプリカまたは DB クラスター内のすべての Aurora レプリカへのレプリケーションがブロックされます。指定した時間が終了すると、影響を受けた Aurora レプリカは自動的にプライマリインスタンスと同期されます。

[Syntax] (構文)

```
SELECT aurora_inject_replica_failure(
   percentage_of_failure,
   time_interval,
   'replica_name'
);
```

オプション

この障害挿入クエリでは、以下のパラメータを使用します。

percentage_of_failure

障害イベントの発生時にブロックするレプリケーションの割合。この値は 0 ~ 100 の倍精度にすることができます。0 を指定すると、レプリケーションはブロックされません。100 を指定すると、すべてのレプリケーションがブロックされます。

time_interval

Aurora レプリカの障害をシミュレートする時間。時間は秒単位で示されます。例えば、値が 20 の場合、シミュレーションは 20 秒間実行されます。

> **Note**
>
> Aurora レプリカの障害イベントの時間を指定するときには注意が必要です。指定する時間が長すぎ、障害イベントの発生時に書き込みインスタンスが大量のデータを書き込んだ場合、Aurora DB クラスターは Aurora レプリカがクラッシュしたものと見なし、レプリカを置き換える可能性があります。

replica_name

障害シミュレーションを挿入する Aurora レプリカ。1 つの Aurora レプリカの障害をシミュレートするには Aurora レプリカの名前を指定します。DB クラスターのすべての Aurora レプリカの障害をシミュレートするには、空の文字列を指定します。

レプリカ名を識別するには、`server_id` 関数の `aurora_replica_status()` 列を参照してください。次に例を示します。

```
postgres=> SELECT server_id FROM aurora_replica_status();
```

ディスクの障害のテスト

障害挿入クエリ関数 `aurora_inject_disk_failure()` を使用して、Aurora PostgreSQL DB クラスターのディスクの障害をシミュレートできます。

ディスク障害のシミュレーションでは、Aurora PostgreSQL DB クラスターがランダムにディスクセグメントをエラーとしてマークします。シミュレーションの実行中、これらのセグメントに対するリクエストはブロックされます。

[Syntax] (構文)

```
SELECT aurora_inject_disk_failure(
   percentage_of_failure,
   index,
   is_disk,
   time_interval
);
```

オプション

この障害挿入クエリでは、以下のパラメータを使用します。

percentage_of_failure

障害イベントの発生時にエラーとしてマークするディスクの割合。この値は 0 ~ 100 の倍精度にすることができます。0 を指定すると、ディスクのどの部分もエラーとしてマークされません。100 を指定すると、ディスク全体がエラーとしてマークされます。

index

障害イベントをシミュレートする特定の論理的なデータブロック。利用可能なデータの論理的なブロックまたはストレージノードの範囲を超えた場合は、指定できる最大インデックス値を示すエラーが表示されます。このエラーを回避するには、「Aurora PostgreSQL DB クラスターのボリュームステータスの表示」を参照してください。

is_disk

障害が論理的なブロックに挿入されるのかストレージノードに挿入されるのかを示します。true を指定すると、論理的なブロックに障害が挿入されます。false を指定すると、ストレージノードに障害が挿入されます。

time_interval

ディスク障害をシミュレートする時間。時間は秒単位で示されます。例えば、値が 20 の場合、シミュレーションは 20 秒間実行されます。

ディスクの輻輳のテスト

障害挿入クエリ関数 `aurora_inject_disk_congestion()` を使用して、Aurora PostgreSQL DB クラスターのディスクの輻輳をシミュレートできます。

ディスク輻輳のシミュレーションでは、Aurora PostgreSQL DB クラスターがランダムにディスクセグメントを輻輳としてマークします。これらのセグメントに対するリクエストは、シミュレーションの実行中、指定した最小遅延値と最大遅延値の間で遅延します。

[Syntax] (構文)

```
SELECT aurora_inject_disk_congestion(
   percentage_of_failure,
   index,
   is_disk,
   time_interval,
   minimum,
   maximum
);
```

オプション

この障害挿入クエリでは、以下のパラメータを使用します。

percentage_of_failure

障害イベントの発生時に輻輳としてマークするディスクの割合。これは 0 ~ 100 の間の倍精度値です。0 を指定すると、ディスクのどの部分も輻輳としてマークされません。100 を指定すると、ディスク全体が輻輳としてマークされます。

index

障害イベントのシミュレーションに使用する特定の論理的なデータブロックまたはストレージノード。

利用可能なデータの論理的なブロックまたはストレージノードの範囲を超えた場合は、指定できる最大インデックス値を示すエラーが表示されます。このエラーを回避するには、「Aurora PostgreSQL DB クラスターのボリュームステータスの表示」を参照してください。

is_disk

障害が論理的なブロックに挿入されるのかストレージノードに挿入されるのかを示します。true を指定すると、論理的なブロックに障害が挿入されます。false を指定すると、ストレージノードに障害が挿入されます。

time_interval

ディスクの輻輳のシミュレートにかかる時間。時間は秒単位で示されます。例えば、値が 20 の場合、シミュレーションは 20 秒間実行されます。

最小値、最大値

輻輳による遅延の最小値と最大値をミリ秒単位で指定します。有効な値の範囲は 0.0 ~ 100.0 ミリ秒です。シミュレーションを実行する間、輻輳としてマークされたディスクセグメントでは、最小値と最大値の間の範囲でランダムな遅延が発生します。最大値は最小値より大きくなければなりません。

Aurora PostgreSQL DB クラスターのボリュームステータスの表示

Amazon Aurora において、DB クラスターボリュームは、論理的なブロックのコレクションで構成されます。各論理的なブロックは、10 ギガバイトの割り当て済みストレージです。これらのブロックは保護グループと呼ばれます。

各保護グループのデータは、6 つの物理ストレージデバイス (ストレージノードと呼ばれる) にわたってレプリケートされます。これらのストレージノードは、DB クラスターがあるリージョン内の 3 つのアベイラビリティーゾーン (AZ) に割り当てられます。また、各ストレージノードには、DB クラスターボリュームに対する 1 つ以上の論理的なデータブロックが含まれます。保護グループおよびストレージノードの詳細については、AWS データベースブログの「Aurora ストレージエンジンの概要」を参照してください。一般的な Aurora クラスターボリュームについての詳細は、「Amazon Aurora ストレージと信頼性」を参照してください。

`aurora_show_volume_status()` 関数を使用して、次のサーバーステータス可変を返します。

- `Disks` — DB クラスターボリュームの論理的なデータブロックの総数。
- `Nodes` – DB クラスターボリュームのストレージノードの総数。

この `aurora_show_volume_status()` 関数を使用すると、`aurora_inject_disk_failure()` 障害挿入関数を使用するときにエラーを回避できます。`aurora_inject_disk_failure()` 障害挿入関数では、ストレージノード全体、またはストレージノード内の単一の論理的なデータブロックの障害をシミュレートします。この関数で、特定の論理的なデータブロックまたはストレージノードのインデックス値を指定します。ただし、DB クラスターボリュームで使用されている論理的なデータブロックまたはストレージノードの数より大きいインデックス値を指定すると、ステートメントよりエラーが返ります。障害挿入クエリの詳細については、「障害挿入クエリを使用した Amazon Aurora PostgreSQL のテスト」を参照してください。

> **Note**
>
> この `aurora_show_volume_status()` 関数は、Aurora PostgreSQL バージョン 10.11 で使用できます。Aurora PostgreSQL のバージョンの詳細については、「Amazon Aurora PostgreSQL リリースとエンジンのバージョン」を参照してください。

[Syntax] (構文)

```
SELECT * FROM aurora_show_volume_status();
```

例

```
customer_database=> SELECT * FROM aurora_show_volume_status();
 disks | nodes
-------+-------
```

```
    96 |     45
```

stats_temp_directory の RAM ディスクを特定する

PostgreSQL `stats_temp_directory` を保存する RAM ディスクに割り当てられたシステムメモリを指定するために Aurora PostgreSQL パラメータ `rds.pg_stat_ramdisk_size` を使用できます。RAM ディスクパラメータは、Aurora PostgreSQL 14 以下のすべてのバージョンで利用できます。

特定のワークロードでは、このパラメータを設定することでパフォーマンスが向上し、IO 要件を軽減することができます。`stats_temp_directory` の詳細については、PostgreSQL のドキュメントで「Run-time Statistics」(ランタイム統計) を参照してください。PostgreSQL バージョン 15 から、PostgreSQL コミュニティは動的共有メモリを使用するように切り替えました。そのため、`stats_temp_directory` に設定する必要はありません。

`stats_temp_directory` の RAM ディスクを有効にするには、`rds.pg_stat_ramdisk_size` パラメータを、DB クラスターで使用されるパラメータグループのゼロ以外の値に設定します。このパラメータは MB を表すため、整数値を使用する必要があります。式、数式、関数が `rds.pg_stat_ramdisk_size` パラメータに対して有効ではありません。変更が反映されるように、DB クラスターを再起動してください。パラメータの設定の詳細については、「「パラメータグループを使用する」」を参照してください。DB クラスターの再起動の詳細については、「Amazon Aurora DB クラスターまたは Amazon Aurora DB インスタンスの再起動」を参照してください。

例えば、次の AWS CLI コマンドは、RAM ディスクパラメータを 256 MB に設定します。

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name db-cl-pg-ramdisk-testing \
    --parameters "ParameterName=rds.pg_stat_ramdisk_size, ParameterValue=256,
 ApplyMethod=pending-reboot"
```

DB クラスターの再起動後、次のコマンドを実行して `stats_temp_directory` のステータスを確認します。

```
postgres=> SHOW stats_temp_directory;
```

コマンドは次の情報を返します。

```
stats_temp_directory
----------------------------
/rdsdbramdisk/pg_stat_tmp
```

```
(1 row)
```

PostgreSQL による一時ファイルの管理

PostgreSQL では、ソート操作とハッシュ操作を実行するクエリは、インスタンスメモリを使用して、work_mem パラメータで指定された値までの結果を格納します。インスタンスメモリが不足すると、結果を保存する一時ファイルが作成されます。これらは、クエリの実行を完了するためにディスクに書き込まれます。その後、これらのファイルは、クエリが完了すると自動的に削除されます。Aurora PostgreSQL では、これらのファイルはローカルストレージを他のログファイルと共有します。Aurora PostgreSQL DB クラスターのローカルストレージスペースをモニタリングするには、Amazon CloudWatch メトリックスで FreeLocalStorage を監視します。詳細については、「Troubleshoot local storage issues」(ローカルストレージの問題のトラブルシューティング) を参照してください。

以下のパラメータと関数を使用して、インスタンスの一時ファイルを管理することができます。

- **temp_file_limit** — このパラメータは、temp_files のサイズ (KB 単位) を超えるクエリをすべてキャンセルします。この制限により、クエリが延々と実行され、一時ファイルでディスクスペースが消費されるのを防ぐことができます。log_temp_files パラメータの結果を使用して値を推定できます。ベストプラクティスとして、ワークロードの動作を調べ、推定値に従って制限を設定してください。次の例は、クエリが制限を超えた場合にキャンセルされる様子を示しています。

  ```
  postgres=> select * from pgbench_accounts, pg_class, big_table;
  ```

  ```
  ERROR: temporary file size exceeds temp_file_limit (64kB)
  ```

- **log_temp_files** — このパラメータは、セッションの一時ファイルが削除されたときに postgresql.log にメッセージを送信します。このパラメータは、クエリが正常に完了した後にログを生成します。そのため、アクティブで長時間実行されるクエリのトラブルシューティングには役立たない可能性があります。

  次の例は、クエリが正常に完了すると、エントリが postgresql.log ファイルに記録され、一時ファイルがクリーンアップされることを示しています。

  ```
  2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:LOG:
   temporary file: path "base/pgsql_tmp/pgsql_tmp31236.5", size 140353536
  ```

```
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:STATEMENT:
  select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid limit 10;
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:LOG:
 temporary file: path "base/pgsql_tmp/pgsql_tmp31236.4", size 180428800
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:STATEMENT:
  select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid limit 10;
```

- **pg_ls_tmpdir** — この関数は RDS for PostgreSQL 13 以降で使用でき、現在の一時ファイルの使用状況を可視化できます。完了したクエリは、関数の結果には表示されません。次の例では、この関数の結果を表示できます。

```
postgres=> select * from pg_ls_tmpdir();
```

```
      name       |    size    |      modification
-----------------+------------+------------------------
 pgsql_tmp8355.1 | 1072250880 | 2023-02-06 22:54:56+00
 pgsql_tmp8351.0 | 1072250880 | 2023-02-06 22:54:43+00
 pgsql_tmp8327.0 | 1072250880 | 2023-02-06 22:54:56+00
 pgsql_tmp8351.1 |  703168512 | 2023-02-06 22:54:56+00
 pgsql_tmp8355.0 | 1072250880 | 2023-02-06 22:54:00+00
 pgsql_tmp8328.1 |  835031040 | 2023-02-06 22:54:56+00
 pgsql_tmp8328.0 | 1072250880 | 2023-02-06 22:54:40+00
(7 rows)
```

```
postgres=> select query from pg_stat_activity where pid = 8355;

query
-------------------------------------------------------------------------------------------
select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid
(1 row)
```

ファイル名には、一時ファイルを生成したセッションの処理 ID (PID) が含まれます。次の例のような、より高度なクエリでは、各 PID の一時ファイルの合計が実行されます。

```
postgres=> select replace(left(name, strpos(name, '.')-1),'pgsql_tmp','') as pid,
 count(*), sum(size) from pg_ls_tmpdir() group by pid;
```

```
 pid  | count |   sum
------+-------+------------------
 8355 |     2 | 2144501760
 8351 |     2 | 2090770432
 8327 |     1 | 1072250880
 8328 |     2 | 2144501760
(4 rows)
```

- **pg_stat_statements** — pg_stat_statements パラメータを有効にすると、呼び出しごとの一時ファイルの平均使用量を表示できます。次の例に示すように、クエリの query_id を特定し、それを使用して一時ファイルの使用状況を調べることができます。

```
postgres=> select queryid from pg_stat_statements where query like 'select a.aid from
 pgbench%';
```

```
       queryid
----------------------
 -7170349228837045701
(1 row)
```

```
postgres=> select queryid, substr(query,1,25), calls, temp_blks_read/calls
 temp_blks_read_per_call, temp_blks_written/calls temp_blks_written_per_call from
 pg_stat_statements where queryid = -7170349228837045701;
```

```
       queryid        |          substr           | calls | temp_blks_read_per_call |
 temp_blks_written_per_call
----------------------+---------------------------+-------+-------------------------
+---------------------------
 -7170349228837045701 | select a.aid from pgbench |    50 |                  239226 |
                    388678
```

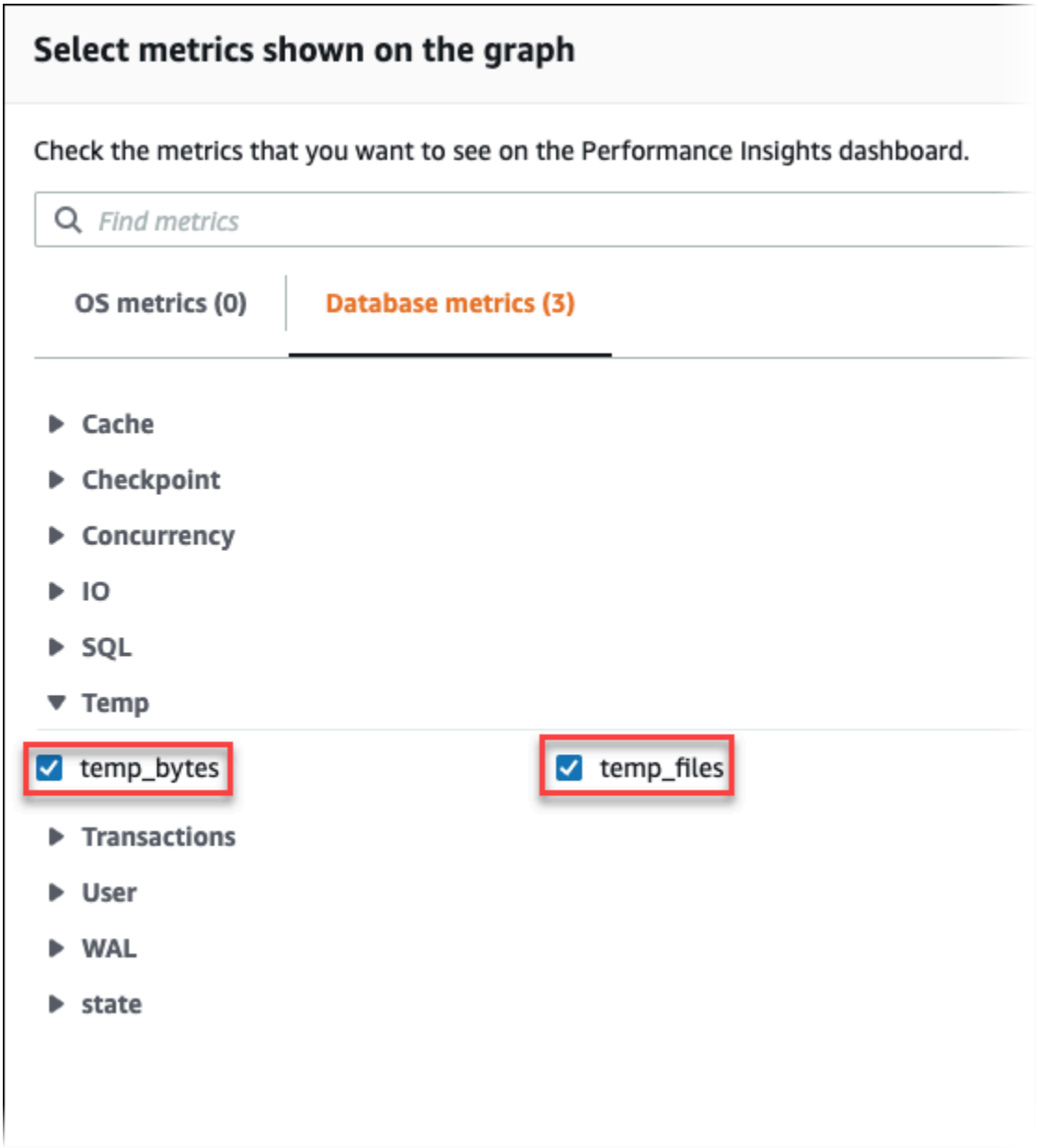
```
(1 row)
```

- **Performance Insights** — Performance Insights ダッシュボードで、temp_bytes と temp_files のメトリクスをオンにすると、一時ファイルの使用状況を確認できます。次に、これら両方のメトリクスの平均と、それらがクエリワークロードにどのように対応しているかを確認できます。Performance Insights 内のビューには、一時ファイルを生成しているクエリが具体的に表示されません。ただし、Performance Insights と `pg_ls_tmpdir` に示されるクエリを組み合わせると、クエリワークロードの変化をトラブルシューティング、分析、判断できます。

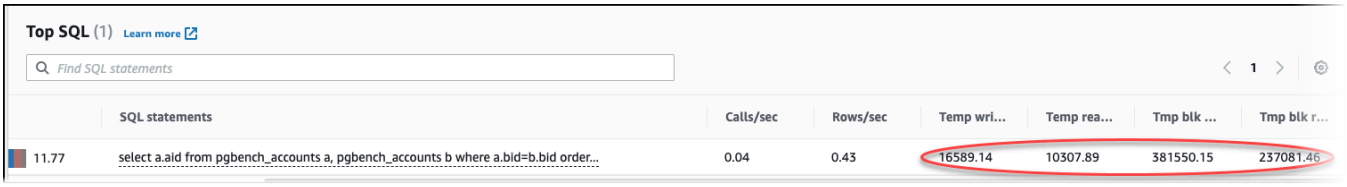
  Performance Insights を使用してメトリクスとクエリを分析する方法については、「Performance Insights ダッシュボードを使用してメトリクスを分析する」を参照してください

  Performance Insights を使用して一時ファイルの使用状況を確認するには

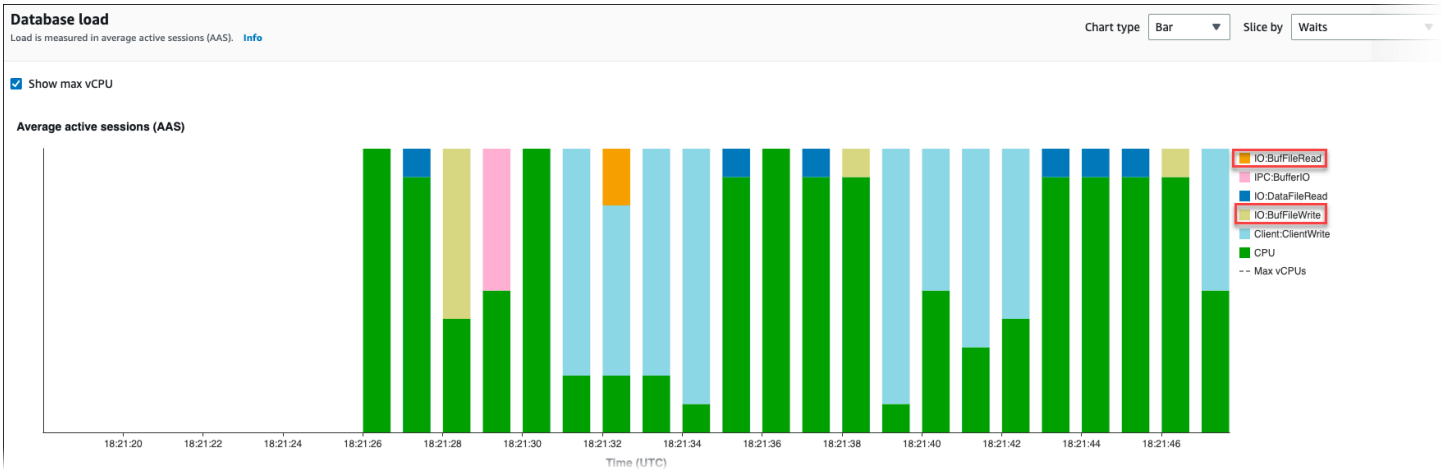
  1. Performance Insights ダッシュボードで、[メトリクスを管理] を選択します。
  2. 次の画像に示すように、[データベースメトリクス] を選択して、[temp_bytes] と [temp_files] を選択します。



3. [トップ SQL] タブで、[設定] アイコンを選択します。
4. [設定] ウィンドウで、[トップ SQL] タブに以下の統計が表示されるようにして、[続行] を選択します。
   - Temp writes/sec
   - Temp reads/sec
   - Tmp blk write/call
   - Tmp blk read/call
5. 次の例に示すように、`pg_ls_tmpdir` で示したクエリと組み合わせると、この一時ファイルが抜き出されます。



IO:BufFileRead と IO:BufFileWrite イベントは、ワークロードの上位クエリが一時ファイルを頻繁に作成するときに発生します。Performance Insights を使用することで、「データベース負荷」と「上位 SQL」セクションの「平均アクティブセッション (AAS)」を参照して、IO:BufFileRead と IO:BufFileWrite を待機中の上位クエリの特定が可能になります。



Performance Insights を使用して上位クエリを分析して、待機イベント別にロードする方法については、「[トップ SQL] タブの概要」を参照してください テンポラリファイルの使用量と関連する待機イベントの増加の原因となるクエリを特定し、調整する必要があります。これらの待機イベントと修復方法の詳細については、「」「IO:BufFileRead および IO:BufFileWrite」を参照してください。

> **Note**
>
> work_mem パラメータは、ソート操作がメモリ不足になり、結果が一時ファイルに書き込まれるタイミングを制御します。このパラメータの設定をデフォルト値より高く変更しないことをお勧めします。変更すると、すべてのデータベースセッションでより多くのメモリを消費することになります。また、複雑な結合とソートを実行する単一セッションでは、それぞれの処理がメモリを消費する並列処理を実行できます。
> ベストプラクティスとして、複数の結合とソートを含む大規模なレポートがある場合は、SET work_mem コマンドを使用してこのパラメータをセッションレベルで設定します。そうすれば、変更は現在のセッションにのみ適用され、値がグローバルに変更されることはありません。

Aurora PostgreSQL の待機イベントでのチューニング

待機イベントは Aurora PostgreSQL の重要なチューニングツールです。セッションがリソースを待っている理由とその内容がわかれば、ボトルネックを減少できます。このセクションの情報を使用して、考えられる原因と修正措置を見つけることができます。このセクションに進む前に、Aurora の基本的な概念、特に以下のトピックを理解することを強くお勧めします。

- Amazon Aurora ストレージと信頼性
- Aurora DB クラスターのパフォーマンスとスケーリングの管理

> **Important**
>
> このセクションの待機イベントは Aurora PostgreSQL 固有のものです。このセクションの情報は、RDS for PostgreSQL ではなく Amazon Aurora のチューニングにのみ使用してください。
>
> このセクションの待機イベントの一部は、これらのデータベースエンジンのオープンソースバージョンに対応するものがありません。その他の待機イベントは、オープンソースエンジンのイベントと同名ですが、動作は異なります。例えば、Amazon Aurora のストレージは、オープンソースのストレージとは異なる動作をするため、ストレージ関連の待機イベントはリソース条件が異なることを示します。

トピック

- Aurora PostgreSQL チューニングの基本概念
- Aurora PostgreSQL の待機イベント
- クライアント: ClientRead
- クライアント: ClientWrite
- CPU
- IO:BufFileRead および IO:BufFileWrite
- IO:DataFileRead
- IO:XactSync
- IPC:DamRecordTxAck
- Lock:advisory

- Lock:extend
- Lock:Relation
- Lock:transactionid
- Lock:tuple
- LWLock:buffer_content (BufferContent)
- LWLock:buffer_mapping
- LWLock:BufferIO (IPC:BufferIO)
- LWLock:lock_manager
- LWLock:MultiXact
- Timeout:PgSleep

Aurora PostgreSQL チューニングの基本概念

Aurora PostgreSQL データベースをチューニングする前に、待機イベントは何か、そしてなぜ発生するのかを確認してください。Aurora PostgreSQL のベーシックメモリとディスクアーキテクチャも確認します。役立つアーキテクチャの図表については、PostgreSQLウィキブックを参照してください。

トピック

- Aurora PostgreSQL の待機イベント
- Aurora PostgreSQL メモリ
- Aurora PostgreSQL プロセス

Aurora PostgreSQL の待機イベント

待機イベントは、セッションが待っているリソースを示します。例えば、待機イベントは `Client:ClientRead`Aurora PostgreSQL がクライアントからのデータの受信を待っているときに発生します。セッションが待機する一般的なリソースには、次のようなものがあります。

- バッファへのシングルスレッドアクセス (例えば、セッションがバッファを変更しようとした場合など)
- 別のセッションによって現在ロックされている行
- 読み込まれたデータファイル

- ログファイルの書き込み

例えば、クエリを満たすために、セッションで完全なテーブルスキャンを実行することがあります。データがまだメモリ上にない場合、セッションはディスク I/O が完了するまで待機します。バッファがメモリに読み込まれるときは、他のセッションが同じバッファにアクセスしているため、セッションは待機しなければならないことがあります。データベースは、事前定義された待機イベントを使用して待機を記録します。これらのイベントはカテゴリに分類されます。

待機イベント自体では、パフォーマンスの問題は表示されません。例えば、要求されたデータがメモリ上にない場合は、ディスクからデータを読み出す必要があります。あるセッションが更新のために行をロックすると、別のセッションはその行を更新できるようにロック解除されるまで待機します。コミットは、ログファイルへの書き込みが完了するまで待機する必要があります。待機は、データベースが正常に機能するために不可欠です。

一般的に、大量の待機イベントはパフォーマンスの問題を示します。そのような場合、待機イベントデータを使用して、セッションが時間を費やしている場所を特定できます。例えば、通常は分単位で実行されるレポートが数時間実行される場合、合計待機時間に最も多く寄与する待機イベントを特定できます。上位の待機イベントの原因を特定できる場合は、パフォーマンス向上のための変更を実行できることがあります。例えば、別のセッションによってロックされている行をセッションが待っている場合、ロックセッションを終了させることができます。

Aurora PostgreSQL メモリ

Aurora PostgreSQL メモリは、共有とローカルに分かれています。

トピック

- Aurora PostgreSQL の共有メモリ
- Aurora PostgreSQL のローカルメモリ

Aurora PostgreSQL の共有メモリ

Aurora PostgreSQL は、インスタンスの起動時に共有メモリを割り当てます。共有メモリは複数のサブエリアに分割されています。以下では、その中でも特に重要なものについて説明します。

トピック

- 共有バッファ
- ログ先行書き込み (WAL) バッファ

共有バッファ

共有バッファプールは、アプリケーション接続によって使用されている、または使用されていたすべてのページを保持する Aurora PostgreSQL メモリ領域です。ページは、ディスクブロックのメモリバージョンです。共有バッファプールは、ディスクから読み込まれたデータブロックをキャッシュします。プールは、ディスクからデータを再読み取りする必要性を減らし、データベースの運用効率を向上させます。

すべてのテーブルとインデックスは、固定サイズのページの配列として格納されます。各ブロックには、行に対応する複数のタプルが含まれています。タプルはどのページにも格納できます。

共有バッファプールには有限メモリがあります。新しいリクエストがメモリにないページを必要とし、メモリがもう存在しない場合、Aurora PostgreSQL は使用頻度の低いページを削除してリクエストに対応します。削除ポリシーは、クロックスイープアルゴリズムによって実装されます。

`shared_buffers`パラメータは、サーバーがデータをキャッシュするメモリ量を決定します。

ログ先行書き込み (WAL) バッファ

ログ先行書き込み (WAL) バッファは、Aurora PostgreSQL が後で永続的ストレージに書き込むトランザクションデータを保持します。WAL メカニズムを使用すると、Aurora PostgreSQL は次のことを実行できます。

- 障害発生後のデータリカバリ
- ディスクへの頻繁な書き込みを回避し、ディスク I/O を削減

クライアントがデータを変更すると、Aurora PostgreSQL は WAL バッファに変更内容を書き込みます。クライアントが`COMMIT`を発すると、WAL ライタプロセスはトランザクションデータを WAL ファイルに書き込みます。

`wal_level`パラメータは、WAL に書き込まれる情報量を決定します。

Aurora PostgreSQL のローカルメモリ

すべてのバックエンドプロセスは、クエリ処理にローカルメモリを割り当てます。

トピック

- ワークメモリ領域
- メンテナンス作業用メモリ領域

- テンポラリバッファ領域

ワークメモリ領域

ワークメモリ領域ソートとハッシュを実行するクエリのテンポラリデータを保持します。例えば、ORDER BY文節を持つクエリはソートを実行します。クエリは、ハッシュ結合と集約でハッシュテーブルを使用します。

テンポラリディスクファイルに書き込む前に、内部ソート操作とハッシュテーブルで使用するメモリ量を指定するwork_memパラメータです。デフォルト値は 4 MB です。複数のセッションを同時に実行でき、各セッションでメンテナンスオペレーションを並行して行うことができます。このため、使用されるワークメモリの合計は、work_mem設定の何倍にもなることがあります。

メンテナンス作業用メモリ領域

メンテナンス作業用メモリ領域は、メンテナンスオペレーション用のデータをキャッシュします。これらの操作には、バキューム処理、インデックス作成、外部キーの追加が含まれます。

maintenance_work_memパラメータは、メンテナンスオペレーションで使用されるメモリの最大量を指定します。デフォルト値は 64 MB です。データベースセッションでは、一度に 1 つのメンテナンスオペレーションしか実行できません。

テンポラリバッファ領域

テンポラリバッファ領域は、データベースセッションごとにテンポラリテーブルをキャッシュします。

各セッションは、指定した制限まで、必要に応じてテンポラリバッファを割り当てます。セッションが終了すると、サーバーはバッファをクリアします。

temp_buffersパラメータは、各セッションで使用されるテンポラリバッファの最大数を設定します。セッション内でテンポラリテーブルを初期に使用する前に、temp_buffers値を変更できます。

Aurora PostgreSQL プロセス

Aurora PostgreSQL は複数のプロセスを使用します。

トピック

- Postmaster プロセス
- バックエンドプロセス

- バックグラウンドプロセス

Postmaster プロセス

Postmaster プロセスは、Aurora PostgreSQL を起動したときに初期のプロセスがスタートされます。Postmaster プロセスには、主に次のようなロールがあります。

- バックグラウンドプロセスのフォークとモニタリング
- クライアントプロセスから認証要求を受信し、データベースが要求を処理する前に認証する

バックエンドプロセス

Postmaster がクライアント要求を認証する場合、Postmaster は新しいバックエンドプロセスをフォークします。これは postgres プロセスとも呼ばれます。1 つのクライアントプロセスが 1 つのバックエンドプロセスに接続されます。クライアントプロセスとバックエンドプロセスは、Postmaster プロセスの介入なしに直接通信します。

バックグラウンドプロセス

Postmaster プロセスは、異なるバックエンドタスクを実行するいくつかのプロセスをフォークします。より重要なものとしては、以下のとおりです。

- WALライター

  Aurora PostgreSQL は WAL (ログ先行書き込み) バッファのデータをログファイルに書き込みます。ログ先行書き込みの原理は、データベースがそれらの変更を説明するログレコードをディスクに書き込むまで、データベースがデータファイルに変更を書き込むことができないということです。WAL メカニズムはディスク I/O を削減し、Aurora PostgreSQL が障害後にデータベースを回復するためにログを使用できるようにします。
- バックグラウンドライター

  このプロセスは、メモリバッファからデータファイルにダーティ (変更された) ページを定期的に書き込みます。バックエンドプロセスがメモリ上でページを変更すると、ページがダーティになります。
- オートバキュームデーモン

  最新のデーモンには以下の構成要素があります。

  - オートバキュームランチャー

- オートバキュームワーカープロセス

オートバキュームをオンにすると、多数の挿入、更新、または削除されたタプルがあるテーブルを確認します。デーモンには、次のようなロールがあります。

- 更新または削除された行によって占有されているディスク領域をリカバリまたは再利用する
- プランナーで使用する統計情報を更新する
- トランザクション ID のラップアラウンドによる古いデータの損失からの保護

オートバキューム機能は、VACUUMとANALYZEコマンドの実行を自動化するもので、VACUUMにはスタンダードとフルのバリエーションがあります。スタンダードバキュームは、他のデータベースオペレーションと並行して実行されます。 VACUUM FULLは、作業中のテーブルを排他的にロックする必要があります。そのため、同じテーブルにアクセスするオペレーションと並行して実行することはできません。VACUUMは相当量の I/O トラフィックを作成し、他のアクティブなセッションのパフォーマンスが低下する原因となることがあります。

Aurora PostgreSQL の待機イベント

次の表では、パフォーマンスの問題を最もよく示す Aurora PostgreSQL の待機イベントと、最も一般的な原因および修正処置をリストアップしています。次の待機イベントは、Amazon Aurora PostgreSQL のイベントのリストのサブセットです。

待機イベント	Definition
クライアント: ClientRead	このイベントは、Aurora PostgreSQL がクライアントからのデータ受信を待っているときに発生します。
クライアント: ClientWrite	このイベントは、Aurora PostgreSQL がクライアントへのデータ書き込みを待っているときに発生します。
CPU	このイベントは、スレッドが CPU でアクティブになっているか、CPU の待機中に発生します。
IO:BufFileRead および IO:BufFileWrite	これらのイベントは、Aurora PostgreSQL がテンポラリファイルを作成するときに発生します。

待機イベント	Definition
IO:DataFileRead	このイベントは、バックエンドプロセスが必要なページをストレージから読み込む際に、ページが共有メモリで使用できないために接続が待機したときに発生します。
IO:XactSync	このイベントは、データベースが、Aurora ストレージサブシステムが通常のトランザクションのコミットを承認するか、準備されたトランザクションのコミットまたはロールバックの承認を待っているときに発生します。
IPC:DamRecordTxAck	このイベントは、Aurora PostgreSQL がデータベースアクティビティストリーミングを使用するセッションでアクティビティストリーミングイベントを生成し、そのイベントが耐久性の高さを持つのを待機するときに発生します。
Lock:advisory	このイベントは、PostgreSQL アプリケーションがロックを使用して、複数のセッションにわたるアクティビティを調整するときに発生します。
Lock:extend	このイベントは、バックエンドプロセスがリレーション拡張のためにロックするのを待機中に、他のプロセスが同じ目的でそのリレーションをロックしているときに発生します。
Lock:Relation	このイベントは、他のトランザクションによって現在ロックされているテーブルまたはビューに対するロックを取得するためにクエリが待っているときに発生します。
Lock:transactionid	このイベントは、トランザクションが行レベルロックを待っているときに発生します。
Lock:tuple	このイベントは、バックエンドプロセスがタプルのロック取得を待機中の場合に発生します。

待機イベント	Definition
LWLock:buffer_content (BufferContent)	このイベントは、セッションがデータページのメモリ内への読み取りまたは書き込みのために待機中、そのデータページが他のセッションで書き込むためにロックされているときに発生します。
LWLock:buffer_mapping	このイベントは、セッションがデータブロックを共有バッファプール内のバッファに関連付けるのを待っているときに発生します。
LWLock:BufferIO (IPC:BufferIO)	このイベントは、Aurora PostgreSQL または RDS for PostgreSQL が、ページへの同時アクセスを試みたときに、他のプロセスが入出力 (I/O) オペレーションを完了するのを待っているときに発生します。
LWLock:lock_manager	このイベントは、Aurora PostgreSQL エンジンが、高速パスロックが不可能な場合に共有ロックのメモリ領域を維持し、ロックの割り当て、チェック、および解放を行うときに発生します。
LWLock:MultiXact	このタイプのイベントは、Aurora PostgreSQL がセッションを開いたままにして、テーブル内の同じ行を含む複数のトランザクションを完了するときに発生します。待機イベントは、複数のトランザクション処理のどの側面で、つまり、LWLock:MultiXactOffsetSLRU、LWLock:MultiXactOffsetBuffer、LWLock:MultiXactMemberSLRU、または LWLock:MultiXactMemberBuffer のいずれで待機イベントが発生しているかを示します。
Timeout:PgSleep	このイベントは、サーバープロセスが`pg_sleep`機能を呼び出し、スリープタイムアウトを待っているときに発生します。

クライアント: ClientRead

`Client:ClientRead`イベントは、Aurora PostgreSQL がクライアントからのデータの受信を待っているときに発生します。

トピック

サポート対象エンジンバージョン

この待機イベント情報は、Aurora PostgreSQL バージョン 10 以降でサポートされています。

Context

Aurora PostgreSQL DB クラスターは、クライアントからのデータ受信を待っています。Aurora PostgreSQL DB クラスターは、クライアントにさらにデータを送信する前に、クライアントからデータを受信する必要があります。クラスターがクライアントからデータを受信する前に待機する時間が`Client:ClientRead`イベントとなります。

待機時間が増加する原因の可能性

`Client:ClientRead`上位待機中に表示されるイベントの一般的な原因には、次のものがあります。

ネットワークレイテンシーの増加

Aurora PostgreSQL DB クラスターとクライアントの間のネットワークレイテンシーが増加することがあります。ネットワークレイテンシーが高いほど、DB クラスターがクライアントからデータを受信するために必要な時間が長くなります。

クライアントへの負荷の増大

クライアント側で CPU プレッシャーまたはネットワーク飽和が発生している可能性があります。クライアント側の負荷が増加すると、クライアントから Aurora PostgreSQL DB クラスターへのデータの転送が遅延する可能性があります。

過剰なネットワークラウンドトリップ

Aurora PostgreSQL DB クラスターとクライアントの間のネットワークラウンドトリップが多くなると、クライアントから Aurora PostgreSQL DB クラスターへのデータの転送が遅延する可能性があります。

大規模なコピーオペレーション

コピーオペレーション中、データはクライアントのファイルシステムから Aurora PostgreSQL DB クラスターに転送されます。DB クラスターに大量のデータを送信すると、クライアントから DB クラスターへのデータの転送が遅延する可能性があります。

アイドル状態のクライアントの接続

Aurora PostgreSQL DB インスタンスへの接続は、トランザクション状態でアイドル状態であり、クライアントからのデータ送信またはコマンドの発行を待っています。この状態は、`Client:ClientRead` イベントの増加につながることがあります。

接続プーリングに使用される pgBouncer

pgBouncer には`pkt_buf`という低レベルネットワーク構成設定があり、デフォルトでは 4,096 に設定されています。ワークロードが 4,096 バイトを超えるクエリパケットを pgBouncer を介して 送信する場合は、`pkt_buf`8,192 に設定することをお勧めします。新しい設定で`Client:ClientRead`イベントの数が減らない場合は、`pkt_buf`を16,384 や 32,768 など、より大きな値に設定にすることをお勧めします。クエリテキストが大きい場合は、大きな設定を使用すると特に効果的です。

アクション

待機イベントの原因に応じたさまざまなアクションをお勧めします。

トピック

- クライアントをクラスターと同じアベイラビリティーゾーンと VPC サブネットに配置します。
- クライアントのスケーリング
- 現行世代のインスタンスを使用
- ネットワーク帯域幅の増加
- ネットワークパフォーマンスの最大値をモニタリングする
- 「トランザクションのアイドル」状態のトランザクションをモニタリングする

クライアントをクラスターと同じアベイラビリティーゾーンと VPC サブネットに配置します。

ネットワークレイテンシーを減らしてネットワークスループットを向上するには、Aurora PostgreSQL DB クラスターと同じアベイラビリティーゾーンおよび仮想プライベートクラウド (VPC) サブネットにクライアントを配置します。クライアントが、DB クラスターにできる限り地理的に近い場所に配置されていることを確認してください。

クライアントのスケーリング

Amazon CloudWatch またはその他のホストメトリクスを使用して、クライアント側が現在 CPU またはネットワーク帯域幅、またはその両方によって制約を受けているかどうかを判断します。クライアント側が制約を受けている場合は、それに応じてクライアントをスケーリングします。

現行世代のインスタンスを使用

場合によっては、ジャンボフレームをサポートする DB インスタンスクラスを使用していない可能性があります。Amazon EC2 でアプリケーションを実行している場合は、クライアント側に現行世代のインスタンスを使用することを検討してください。また、クライアントのOSで最大送信単位 (MTU) を設定します。この技術では、ネットワークラウンドトリップの数を減らし、ネットワークスループットを向上させることができます。詳細については、Linux インスタンス用 Amazon EC2 ユーザーガイドの「ジャンボフレーム (9001 MTU)」を参照してください。

DB インスタンスクラスの詳細については、「Aurora DB インスタンスクラス」を参照してください。Amazon EC2 インスタンスタイプと同等の DB インスタンスクラスを決定するには、db.Amazon EC2 インスタンスタイプの前に配置します。例えば、`r5.8xlarge`Amazon EC2 インスタンスは`db.r5.8xlarge`DB インスタンスクラスと同等です。

ネットワーク帯域幅の増加

`NetworkReceiveThroughput`および`NetworkTransmitThroughput`の Amazon CloudWatch メトリクスを使用して、DB クラスター上の着信および発信ネットワークトラフィックをモニタリングします。これらのメトリックは、ネットワーク帯域幅がワークロードに十分であるかどうかを判断するのに役立ちます。

ネットワーク帯域幅が十分でない場合は、増加してください。AWSクライアントまたは DB インスタンスがネットワーク帯域幅の制限に達している場合、帯域幅を増やす唯一の方法は、DB インスタンスのサイズを増加することことです。

CloudWatch のメトリクスの詳細については、「Amazon Aurora の Amazon CloudWatch メトリクス」を参照してください。

ネットワークパフォーマンスの最大値をモニタリングする

Amazon EC2 クライアントを使用している場合、Amazon EC2 は、集約されたインバウンドとアウトバウンドのネットワーク帯域幅を含む、ネットワークパフォーマンスメトリックの最大値を提供します。また、パケットが期待どおりに返されることを確認する接続追跡、ドメインネームシステム (DNS) などのサービスへのリンクローカルサービスアクセスも提供します。これらの最大値をモニタリングするには、現在の拡張ネットワークドライバーを使用し、クライアントのネットワークパフォーマンスをモニタリングします。

詳細については、Linux インスタンス用 Amazon EC2 ユーザーガイド」の「Amazon EC2 インスタンスのネットワークパフォーマンスをモニタリング」、「Windows インスタンス用 Amazon EC2 ユーザーガイド」の「Amazon EC2 インスタンスのネットワークパフォーマンスをモニタリング」を参照してください。

「トランザクションのアイドル」状態のトランザクションをモニタリングする

`idle in transaction`接続の数が増えているかどうかをチェックします。これを行うには、`pg_stat_activity`テーブルの`state`列をモニタリングします。次のようなクエリを実行することで、接続出典を特定できる場合があります。

```
select client_addr, state, count(1) from pg_stat_activity
where state like 'idle in transaction%'
group by 1,2
order by 3 desc
```

クライアント: ClientWrite

`Client:ClientWrite`イベントは、Aurora PostgreSQL がクライアントへのデータの書き込みを待っているときに発生します。

トピック

- サポート対象エンジンバージョン
- Context
- 待機時間が増加する原因の可能性
- アクション

サポート対象エンジンバージョン

この待機イベント情報は、Aurora PostgreSQL バージョン 10 以降でサポートされています。

Context

クライアントプロセスは、クラスターがさらにデータを送信する前に、Aurora PostgreSQL DB クラスターから受信したすべてのデータを読み込む必要があります。クライアントにより多くのデータを送信する前にクラスターが待機する時間は、`Client:ClientWrite`イベントになります。

Aurora PostgreSQL DB クラスターとクライアント間のネットワークスループットが低下すると、このイベントが発生することがあります。クライアントの CPU プレッシャーとネットワークの飽和により、このイベントが発生することがあります。CPU プレッシャーとは、CPU が完全に使用されており、CPU 時間を待っているタスクがあることです。ネットワーク飽和度とは、データベースとクライアント間のネットワークが、処理できるデータ以上のデータを伝送しているときです。

待機時間が増加する原因の可能性

`Client:ClientWrite`上位待機中に表示されるイベントの一般的な原因には、次のものがあります。

ネットワークレイテンシーの増加

Aurora PostgreSQL DB クラスターとクライアントの間のネットワークレイテンシーが増加することがあります。ネットワークレイテンシーが高いほど、クライアントからデータを受信するために必要な時間が長くなります。

クライアント側への負荷の増加

クライアント側で CPU プレッシャーまたはネットワーク飽和が発生する可能性があります。クライアントの負荷が増加すると、Aurora PostgreSQL DB クラスターからのデータの受信が遅延します。

クライアントに送信される大量のデータ

Aurora PostgreSQL DB クラスターがクライアントに大量のデータを送信している可能性があります。クライアントは、クラスターのデータ送信と同じ速度ではデータを受信できない場合があります。大きなテーブルのコピーなどのアクティビティは、`Client:ClientWrite`イベントの増加につながることがあります。

アクション

待機イベントの原因に応じたさまざまなアクションをお勧めします。

トピック

- クライアントをクラスターと同じアベイラビリティーゾーンと VPC サブネットに配置します。
- 現行世代のインスタンス
- クライアントに送信するデータ量を減らします。
- クライアントのスケーリング

クライアントをクラスターと同じアベイラビリティーゾーンと VPC サブネットに配置します。

ネットワークレイテンシーを減らしてネットワークスループットを向上するには、Aurora PostgreSQL DB クラスターと同じアベイラビリティーゾーンおよび仮想プライベートクラウド (VPC) サブネットにクライアントを配置します。

現行世代のインスタンス

場合によっては、ジャンボフレームをサポートする DB インスタンスクラスを使用していない可能性があります。Amazon EC2 でアプリケーションを実行している場合は、クライアント側に現行世代のインスタンスを使用することを検討してください。また、クライアントのOSで最大送信単位 (MTU) を設定します。この技術では、ネットワークラウンドトリップの数を減らし、ネットワークスループットを向上させることができます。詳細については、Linux インスタンス用 Amazon EC2 ユーザーガイドの「ジャンボフレーム (9001 MTU)」を参照してください。

DB インスタンスクラスの詳細については、「Aurora DB インスタンスクラス」を参照してください。Amazon EC2 インスタンスタイプと同等の DB インスタンスクラスを決定するには、db.Amazon EC2 インスタンスタイプの前に配置します。例えば、`r5.8xlarge`Amazon EC2 インスタンスは`db.r5.8xlarge`DB インスタンスクラスと同等です。

クライアントに送信するデータ量を減らします。

可能であれば、Aurora PostgreSQL DB クラスターがクライアントに送信するデータ量を減らすようにアプリケーションを調整します。このような調整を行うと、クライアントの CPU やネットワークの競合を軽減します。

クライアントのスケーリング

Amazon CloudWatch またはその他のホストメトリクスを使用して、クライアント側が現在 CPU またはネットワーク帯域幅、またはその両方によって制約を受けているかどうかを判断します。クライアント側が制約を受けている場合は、それに応じてクライアントをスケーリングします。

CPU

この待機イベントは、スレッドが CPU でアクティブであるか CPU の待機中に発生します。

トピック

サポート対象エンジンバージョン

この待機イベント情報は、Aurora PostgreSQL バージョン 9.6 以降に関連します。

Context

中央処理装置 (CPU)は、命令を実行するコンピュータのコンポーネントです。例えば、CPU 命令は演算処理を実行し、メモリ上でデータを交換します。クエリがデータベースエンジンを介して実行する命令の数が増えると、クエリの実行にかかる時間が長くなります。CPU スケジューリングは、CPU にプロセス時間を与えています。スケジューリングは、OS のカーネルによってオーケストレーションされます。

トピック

この待機の発生時期を確認する方法

このCPU待機イベントは、バックエンドプロセスが CPU でアクティブであるか、CPU を待っていることを示します。クエリに次の情報が表示されると、発生していることがわかります。

- 「pg_stat_activity.state」 列には値activeがあります。
- pg_stat_activityのwait_event_typeおよびwait_eventの列は、両方ともnullです。

CPU を使用中または待機中のバックエンドプロセスを確認するには、次のクエリを実行します。

```
SELECT *
FROM   pg_stat_activity
```

```
WHERE  state = 'active'
AND    wait_event_type IS NULL
AND    wait_event IS NULL;
```

DbLoadCPU メトリクス

CPU の Performance Insights のメトリクスは DBLoadCPU です。DBLoadCPUの値は、Amazon CloudWatch メトリクスの値とは異なる場合がありますCPUUtilization。後者のメトリクスは、データベースインスタンスのハイパーバイザーから収集されます。

os.cpuUtilization メトリック

Performance Insights OS のメトリクスは、CPU 使用率に関する詳細情報を提供します。例えば、次のメトリクスを表示できます。

- os.cpuUtilization.nice.avg
- os.cpuUtilization.total.avg
- os.cpuUtilization.wait.avg
- os.cpuUtilization.idle.avg

Performance Insights は、データベースエンジンによる CPU 使用率をos.cpuUtilization.nice.avgのように報告します。

CPU スケジューリングの原因の可能性

OS の観点から、CPU はアイドルスレッドを実行していないときはアクティブです。CPU は計算の実行中はアクティブですが、メモリ I/O の待機中もアクティブになります。このタイプの I/O は、一般的なデータベースワークロードを支配します。

以下の条件が満たされると、プロセスは CPU でスケジュールされるのを待機する可能性があります。

- CloudWatchCPUUtilizationメトリクスは 100% に近いです。
- 平均ロードは vCPUs の数よりも大きく、ロードが重いことを示しています。このメトリクスは、loadAverageMinutePerformance Insights の OS メトリクスセクションで見ることができます。

待機時間が増加する原因の可能性

CPU 待機イベントが通常よりも頻繁に発生する場合は、パフォーマンスの問題を示している可能性があり、典型的な原因は次のとおりです。

トピック

突然のスパイクの原因の可能性

突然のスパイクの原因として最も可能性の高いものは次のとおりです。

- アプリケーションがデータベースへの同時接続を開きすぎています。このシナリオは「接続ストーム」と呼ばれます。
- アプリケーションのワークロードは、次のいずれかの方法で変更されました。
  - 新しいクエリ
  - データセットのサイズの増加
  - インデックスのメンテナンスまたは作成
  - 新しい関数
  - 新しいオペレーター
  - パラレルクエリ実行の増加
- クエリ実行プランが変更されました。場合によっては、変更によってバッファが増加することがあります。例えば、以前はインデックスを使用していたクエリが、現在はシーケンシャルスキャンを使用します。この場合、同じ目標を達成するには、クエリがより多くの CPU を必要とします。

長期の高周波の原因の可能性

長期間にわたって再発するイベントの原因として最も可能性の高いもの:

- CPU で同時に実行されているバックエンドプロセスが多すぎます。これらのプロセスは、パラレルワーカーにすることができます。
- クエリのパフォーマンスは、大量のバッファを必要とするため最適ではありません。

コーナーケース

考えられる原因のいずれも実際の原因ではない場合は、以下のような状況が発生することがあります。

- CPU がプロセスを入れ替えています。
- CPU コンテキストスイッチングが増加しました。
- Aurora PostgreSQL コードに待機イベントがありません。

アクション

CPU 待機イベントがデータベースアクティビティを占領している場合でも、必ずしもパフォーマンスの問題を示すわけではありません。パフォーマンスが低下した場合にのみ、このイベントに応答します。

トピック

- データベースが CPU の増加原因かどうかを調べる
- 接続数が増加したかどうかを判断する
- ワークロードの変更に対応

データベースが CPU の増加原因かどうかを調べる

`os.cpuUtilization.nice.avg`Performance Insights のメトリクスを検証します。この値が CPU 使用率よりはるかに小さい場合、データベース以外のプロセスが CPU の主な原因となっています。

接続数が増加したかどうかを判断する

`DatabaseConnections`Amazon CloudWatch のメトリクスを検証します。アクションは、CPU の待機イベントが増加した期間中の数値の増減によって異なります。

接続数が増加した

接続数が増えた場合は、CPU を消費しているバックエンドプロセスの数と vCPUs の数を比較します。以下のシナリオが考えられます。

- CPU を消費するバックエンドプロセスの数が、vCPUs の数より少なくなっています。

  この場合、接続数は問題ではありません。ただし、それでも CPU 使用率を下げようとすることがあります。

- CPU を消費するバックエンドプロセスの数が vCPUs の数を超えています。

  このような場合は、以下のオプションを検討します。

  - データベースに接続されているバックエンドプロセスの数を減らします。例えば、RDS Proxy などの接続プーリングソリューションを実装します。詳細については、「Amazon RDS Proxy for Aurora の使用」を参照してください。
  - インスタンスサイズをアップグレードして vCPUs の数を増やします。
  - 一部の読み取り専用ワークロードをリーダーノードにリダイレクトします (該当する場合)。

接続は増加しなかった

blks_hitPerformance Insights のメトリクスを検証します。blks_hitと CPU 使用率の増加の相関関係を探してください。以下のシナリオが考えられます。

- CPU 使用率とblks_hitが相関しています。

  この場合、CPU 使用率にリンクされている上位 SQL ステートメントを検索し、プランの変更を検討します。以下のいずれかの対策を使用できます。

  - 計画をマニュアルで説明し、予想される実行プランと比較します。
  - 秒単位のブロックヒット数とローカルブロックヒット数の増加を確認します。Performance Insights ダッシュボードの上位 SQLセクションで、Preferences (設定) を選択します。
- CPU 使用率とblks_hitには相関関係がありません。

  このような場合は、次のいずれかに該当するかどうかを判断します。

  - アプリケーションは、データベースとの接続と切断を高速で行っています。

    log_connectionsおよびlog_disconnectionsをオンにして、PostgreSQL のログを分析します。pgbadgerログアナライザの使用を検討します。詳細については、「https://github.com/darold/pgbadger」を参照してください。
  - OS はオーバーロード状態です。

    この場合、Performance Insights は、バックエンドプロセスが通常よりも長い時間 CPU を消費していることを示しています。Performance Insights のos.cpuUtilizationメトリクスサイトまたはCPUUtilizationCloudWatch のメトリクスで エビデンスを探します。OS がオーバーロード状態になっている場合は、拡張モニタリングのメトリックを参照してさらに診断します。具体的には、プロセスリストと各プロセスが消費する CPU の割合を確認します。
  - 上位 SQL ステートメントが消費する CPU が多すぎます。

CPU 使用率とリンクするステートメントを検証し、CPU の使用率を減らせるかどうかを確認します。EXPLAINコマンドを実行し、最も影響が大きいプランノードにフォーカスします。PostgreSQL の実行計画ビジュアライザーの使用を検討してください。このツールを試すには、http://explain.dalibo.com/を参照してください。

ワークロードの変更に対応

ワークロードが変更された場合は、次のタイプの変更を探します。

新しいクエリ

新しいクエリが想定されているかどうかを確認します。その場合は、その実行計画と秒単位の実行数が想定されていることを確認してください。

データセットのサイズの増加

パーティショニングが未実装の場合は、それが役立つかどうかを判断します。この戦略では、クエリで取得する必要があるページ数を減らすことができます。

インデックスのメンテナンスまたは作成

メンテナンスのスケジュールが想定されているかどうかを確認します。ベストプラクティスは、ピークアクティビティ以外のメンテナンスアクティビティをスケジュールすることです。

新しい関数

これらの機能がテスト中に想定したとおりに動作するかどうかを確認します。具体的には、秒単位の実行数が想定されているかどうかを確認します。

新しいオペレーター

テスト中に想定どおりに動作するかどうかを確認します。

パラレルクエリの実行の増加

以下のいずれかの状況が発生するかどうかを確認します。

- 関連する関係やインデックスのサイズが突然大きくなり、min_parallel_table_scan_sizeまたはmin_parallel_index_scan_sizeは大きく異なるようになりました。
- 「parallel_setup_costまたはparallel_tuple_cost」に最近変更が加えられました。
- 「max_parallel_workersまたはmax_parallel_workers_per_gather」に最近変更が加えられました。

IO:BufFileRead および IO:BufFileWrite

`IO:BufFileRead` と `IO:BufFileWrite` イベントは、Aurora PostgreSQL がテンポラリファイルを作成するときに発生します。作業メモリパラメータが現在の定義より多くのメモリを必要とするオペレーションは、テンポラリデータを永続的ストレージに書き込みます。この操作は「spilling to disk (ディスクへの流出)」と呼ばれることがあります。

トピック

サポート対象エンジンバージョン

この待機イベント情報は、Aurora PostgreSQL のすべてのバージョンでサポートされています。

Context

`IO:BufFileRead`そして`IO:BufFileWrite`は、作業メモリ領域とメンテナンス作業用メモリ領域に関連します。これらのローカルメモリ領域の詳細については、「ワークメモリ領域」と「メンテナンス作業用メモリ領域」を参照してください。

デフォルト値は `work_mem` 4 MB です。一つのセッションがパラレルにオペレーションを実行する場合、パラレル処理を行う各ワーカーは 4 MB のメモリを使用します。このため、`work_mem`を慎重に設定してください。値を大きくしすぎると、多くのセッションを実行しているデータベースがメモリを過剰に消費することがあります。値を低く設定しすぎると、Aurora PostgreSQL はローカルストレージにテンポラリファイルを作成します。これらのテンポラリファイルのためのディスク I/O により、パフォーマンスが低下する可能性があります。

次のようなイベントが発生する場合、データベースがテンポラリファイルを生成している可能性があります。

1. 可用性の急激な低下
2. 空き領域の高速リカバリ

また、「チェーンソー」のパターンが表示されるかもしれません。このパターンは、データベースが小さなファイルを常に作成していることを示す可能性があります。

待機時間が増加する原因の可能性

一般に、これらの待機イベントは、work_memまたはmaintenance_work_memパラメータが割り当てられるよりも多くのメモリを消費するオペレーションによって発生します。補うために、オペレーションはテンポラリファイルに書き込みます。IO:BufFileReadそしてIO:BufFileWriteイベントの一般的な原因には、次のようなものがあります。

作業用メモリ領域に存在するメモリより多くのメモリを必要とするクエリ

次の特性を持つクエリは、作業メモリ領域を使用します。

- ハッシュ結合
- ORDER BY 句
- GROUP BY 句
- DISTINCT
- Window 関数
- CREATE TABLE AS SELECT
- REFRESH MATERIALIZED VIEW

メンテナンス作業メモリ領域に存在するメモリより多くのメモリを必要とするステートメント

次のステートメントは、メンテナンス作業メモリ領域を使用します。

- CREATE INDEX
- CLUSTER

アクション

待機イベントの原因に応じたさまざまなアクションをお勧めします。

トピック

- 問題の特定
- ジョイントクエリを検証する
- ORDER BY クエリと GROUP BY クエリを検証する
- DISTINCT オペレーションの使用を避ける
- GROUP BY 関数の代わりにウィンドウ関数の使用を検討してください。

- マテリアライズドビューと CTAS ステートメントの調査
- インデックスの作成時に pg_repack を使用する
- テーブルをクラスター化するときに maintenance_work_mem を増やす
- IO:BufFileRead および IO:BufFileWrite を防ぐためにメモリを調整します

問題の特定

Performance Insights がオンではない状況で、IO:BufFileReadとIO:BufFileWriteが通常よりも頻繁に発生している疑いがあると想定します。次を実行してください。

1. FreeLocalStorageAmazon CloudWatch のメトリクスを検証します。
2. ギザギザのトゲが連なるチェーンソーパターンを探してみてください。

チェーンソーパターンは、多くの場合はテンポラリファイルに関連付けられ、ストレージの迅速な消費と解放を示します。このパターンが見られたら、「Performance Insights」 をオンにします。Performance Insights を使用すると、待機イベントがいつ発生し、どのクエリに関連するかを特定できます。解決策は、イベントを引き起こす特定のクエリによって異なります。

または、パラメータlog_temp_filesを設定します。このパラメータは、しきい値 KB を超えるテンポラリファイルを生成するすべてのクエリを ログに記録します。値が 0 の場合、Aurora PostgreSQL すべてのテンポラリファイルを ログに記録します。値が 1024 の場合、Aurora PostgreSQL は 1 MB を超えるテンポラリファイルを生成するすべてのクエリを ログに記録します。log_temp_filesについての詳細は、PostgreSQL ドキュメントのError reporting and logging を参照してください。

ジョイントクエリを検証する

アプリケーションでは、おそらくジョイントを使用しています。例えば、次のクエリは 4 つのテーブルをジョイントします。

```
SELECT *
       FROM order
 INNER JOIN order_item
       ON (order.id = order_item.order_id)
 INNER JOIN customer
       ON (customer.id = order.customer_id)
 INNER JOIN customer_address
       ON (customer_address.customer_id = customer.id AND
           order.customer_address_id = customer_address.id)
```

```
 WHERE customer.id = 1234567890;
```

テンポラリファイル使用量が急増する原因は、クエリ自体の問題の可能性があります。例えば、壊れた節はジョイントを適切にフィルタリングしない可能性があります。次の例では 2 番目の内部ジョイントを考えてみましょう。

```
SELECT *
       FROM order
 INNER JOIN order_item
       ON (order.id = order_item.order_id)
 INNER JOIN customer
       ON (customer.id = customer.id)
 INNER JOIN customer_address
       ON (customer_address.customer_id = customer.id AND
           order.customer_address_id = customer_address.id)
 WHERE customer.id = 1234567890;
```

前のクエリが誤ってcustomer.idをcustomer.idにジョイントし、すべての顧客とすべての注文の間にデカルト積を生成します。このタイプの偶発的なジョイントは、大きなテンポラリファイルを生成します。テーブルのサイズによっては、デカルトクエリでストレージがいっぱいになることもあります。以下の条件を満たす場合は、アプリケーションにデカルトジョインが生成される場合があります。

- ストレージの可用性が大きく急激に低下し、その後、高速リカバリが起こります。
- インデックスは作成されていません。
- CREATE TABLE FROM SELECTステートメントは発行されていません。
- マテリアライズドビューはリフレッシュされません。

テーブルが適切なキーを使用してジョイントされているかどうかを確認するには、クエリおよびオブジェクト関係マッピングディレクティブを調べます。アプリケーションの特定のクエリは常に呼び出されるわけではなく、一部のクエリは動的に生成されることに注意してください。

ORDER BY クエリと GROUP BY クエリを検証する

場合によっては、ORDER BY節を使用するとテンポラリファイルが過剰になる可能性があります。以下のガイドラインを検討します。

- 順序付けが必要な場合のみ、ORDER BYに列を含めてください。このガイドラインは、数千行を返し、ORDER BY節で多数の列を指定するクエリでは特に重要です。

- ORDER BY節が同じ昇順または降順の列にマッチする場合、高速化するためにインデックスの作成を検討します。パーシャルインデックスのほうが小さいため好ましいです。小さいインデックスは、より迅速に読み込まれ、トラバースされます。
- NULL 値を受け入れることができる列のインデックスを作成する場合は、NULL 値をインデックスの最後に格納するか、先頭に格納するかを検討します。

  可能であれば、結果セットをフィルタリングして、順序付けが必要な行の数を減らします。WITH節ステートメントまたはサブクエリを使用する場合、内部クエリが結果セットを生成し、外部クエリに渡すことに注意してください。クエリが行をより多くフィルタリングすると、クエリが行う必要がある順序付けは減ります。
- 完全な結果セットを取得する必要がない場合は、LIMIT節を使用します。例えば、上位 5 行だけが必要な場合、LIMIT節を使用したクエリは結果を生成し続けることはありません。このように、クエリに必要なメモリとテンポラリファイルが減ります。

GROUP BY 句を使用するクエリは、テンポラリファイルを要求することもできます。GROUP BY クエリは、次のような関数を使用して値を要約します。

- COUNT
- AVG
- MIN
- MAX
- SUM
- STDDEV

GROUP BYクエリをチューニングするには、ORDER BYクエリの推奨事項に従ってください。

DISTINCT オペレーションの使用を避ける

可能であれば、DISTINCTオペレーションを使用して重複した行を削除することは避けてください。クエリが返す行が不要かつ重複していればいるほど、VDISTINCTオペレーションのコストは高くなります。可能であれば、異なるテーブルに対して同じフィルターを使用している場合でも、WHERE節でフィルターを追加してください。クエリをフィルタリングして正しく結合すると、パフォーマンスが向上し、リソースの使用量が削減されます。また、誤ったレポートや結果を防ぐことができます。

DISTINCTを同じテーブルの複数の行に使用する必要がある場合、複合インデックスの作成を検討してください。インデックスに複数の列をグループ化すると、個別の行を評価する時間を短縮で

きます。また、Amazon Aurora PostgreSQL バージョン 10 以降を使用している場合は、CREATE STATISTICSコマンドを使用して複数の列間で統計を関連付けられます。

GROUP BY 関数の代わりにウィンドウ関数の使用を検討してください。

GROUP BYを使用すると、結果セットを変更し、集計結果を取得できます。ウィンドウ関数を使用すると、結果セットを変更せずにデータを集計できます。ウィンドウ関数は、OVER句を使用して、クエリによって定義されたセット間で計算を実行し、ある行を別の行に関連付けます。ウィンドウ関数に含まれるすべてのGROUP BY関数は使用できますが、次のような関数も使用可能です。

- RANK
- ARRAY_AGG
- ROW_NUMBER
- LAG
- LEAD

ウィンドウ関数によって生成されるテンポラリファイルの数を最小限に抑えるには、2 つの異なる集計が必要な場合は同じ結果セットの重複を削除してください。次のクエリについて考えます。

```
SELECT sum(salary) OVER (PARTITION BY dept ORDER BY salary DESC) as sum_salary
     , avg(salary) OVER (PARTITION BY dept ORDER BY salary ASC) as avg_salary
  FROM empsalary;
```

WINDOW節のクエリは、次のように書き換えることができます。

```
SELECT sum(salary) OVER w as sum_salary
         , avg(salary) OVER w as_avg_salary
    FROM empsalary
  WINDOW w AS (PARTITION BY dept ORDER BY salary DESC);
```

デフォルトでは、Aurora PostgreSQL 実行プランナーは類似したノードを統合し、オペレーションが重複しないようにします。ただし、ウィンドウブロックに明示的な宣言を使用すると、クエリをより簡単に維持できます。また、重複を防止するとパフォーマンスの向上につながることがあります。

マテリアライズドビューと CTAS ステートメントの調査

マテリアライズドビューがリフレッシュされると、クエリが実行されます。このクエリには、GROUP BY、ORDER BY、DISTINCTのような操作を含めることができます。リフレッシュ中に、大量のテン

ポラリファイルや待機イベントIO:BufFileWriteおよびIO:BufFileReadが発生することがあります。同様に、SELECTに基づいてテーブルを作成すると、CREATE TABLEステートメントはクエリを実行します。必要なテンポラリファイルを減らすには、クエリを最適化します。

インデックスの作成時に pg_repack を使用する

インデックスを作成すると、エンジンは結果セットを順序付けます。テーブルのサイズが大きくなり、インデックスで指定された列の値が多様化していくと、テンポラリファイルはより多くの領域を必要とします。ほとんどの場合、メンテナンス作業のメモリ領域を変更しなければ、大きなテーブルのテンポラリファイルの作成を防ぐことはできません。詳細については、「メンテナンス作業用メモリ領域」を参照してください。

大きなインデックスを再作成するときに考えられる回避策としては、pg_repack ツールを使用することが挙げられます。詳細については、「pg_repack のドキュメント」で「最小限のロックで PostgreSQL データベース内のテーブルを再編成する」を参照してください。

テーブルをクラスター化するときに maintenance_work_mem を増やす

CLUSTERコマンドは、index_nameで指定した既存のインデックスに基づいて、table_nameで指定したテーブルをクラスター化します。Aurora PostgreSQL は、指定されたインデックスの順序に一致するようにテーブルを物理的に再作成します。

磁気ストレージが普及していたころは、ストレージのスループットが限られていたため、クラスター化が一般的でした。今では、SSD ベースのストレージが一般的となり、クラスター化はあまり一般的ではなくなっています。ただし、テーブルをクラスター化すると、テーブルのサイズ、インデックス、クエリなどによってパフォーマンスが多少向上することがあります。

CLUSTERコマンドを実行して、待機イベントIO:BufFileWrite、IO:BufFileReadをモニタリングし、maintenance_work_memをチューニングします。メモリサイズをかなり大きくしてください。高い値は、エンジンがクラスター化オペレーションのためにより多くのメモリを使用できることを意味します。

IO:BufFileRead および IO:BufFileWrite を防ぐためにメモリを調整します

状況によっては、メモリのチューニングが必要です。次の要件のバランスをとることが目標です。

- work_mem値 (「ワークメモリ領域」を参照)
- 割り引いた後の残りのメモリ shared_buffers 値 (「バッファプール」を参照)
- max_connectionsで制限されるオープンおよび使用中の最大接続数

作業メモリ領域のサイズを拡大する

状況によっては、セッションで使用されるメモリを増やすことが唯一の選択肢となることもあります。クエリが正しく記述され、ジョイントに正しいキーを使用している場合は、work_mem値の増加を検討してください。詳細については、「ワークメモリ領域」を参照してください。

クエリが生成するテンポラリファイルの数を調べるには、log_temp_files を 0 に設定します。work_mem 値を口グで識別される最大値まで上げると、クエリでテンポラリファイルが生成されるのを防ぎます。ただし、work_memは各接続またはパラレルワーカーにプランノードあたりの最大値を設定します。データベースに 5,000 の接続があり、それぞれが 256 MiB のメモリを使用する場合、エンジンは 1.2 TiB の RAM を必要とします。そのため、インスタンスのメモリが不足する可能性があります。

共有バッファプールに十分なメモリを予約する

データベースでは、作業用メモリ領域だけでなく、共有バッファプールなどのメモリ領域が使用されます。work_memを増加する前に、これらの追加メモリ領域の要件を考慮してください。バッファープールの詳細については、「バッファプール」を参照してください。

例えば、Aurora PostgreSQL インスタンスクラスが db.r5.2xlarge であると仮定します。このクラスには 64 GiB のメモリがあります。デフォルトでは、メモリの 75% が共有バッファプール用に予約されています。共有メモリ領域に割り当てられた量を引くと、16,384 MB が残ります。OS やエンジンもメモリを必要とするため、残りのメモリを作業メモリ領域にのみ割り当てないでください。

work_memに割り当て可能なメモリはインスタンスクラスによって異なります。より大きなインスタンスクラスを使用すると、より多くのメモリが使用できます。ただし、前の例では 16 GiB 以上は使用できません。そうでなければ、メモリ不足に陥ったときにインスタンスが使用できなくなります。インスタンスを利用できない状態から回復するには、Aurora PostgreSQL オートメーションサービスが自動的に再起動します。

接続の数を管理する

データベースインスタンスでの同時接続が 5,000 とします。各接続では、work_memのうち少なくとも 4 MiB を使用します。接続に必要なメモリ消費量が多いと、パフォーマンスが低下する可能性があります。これに対して、次のオプションがあります。

- より大きなインスタンスクラスにアップグレードします。
- 接続プロキシまたはプーラーを使用することで、データベースの同時接続の数を減らします。

プロキシの場合は、アプリケーションに基づいて Amazon RDS プロキシ、pgBouncer、または接続プーラーを検討してください。この解決策は CPU ロードを軽減します。また、すべての接続が作業メモリ領域を必要とする場合のリスクも軽減します。データベース接続数が少ない場合は、work_memの値を増やすことができます。このように、IO:BufFileReadそしてIO:BufFileWrite待機イベントの発生を減らします。また、作業メモリ領域で待っているクエリが大幅に高速化します。

IO:DataFileRead

IO:DataFileReadイベントは、バックエンドプロセスが必要なページを読み込む際に、ページが共有メモリで使用できないため接続が待機したときに発生します。

トピック

- サポート対象エンジンバージョン
- Context
- 待機時間が増加する原因の可能性
- アクション

サポート対象エンジンバージョン

この待機イベント情報は、Aurora PostgreSQL のすべてのバージョンでサポートされています。

Context

すべてのクエリおよびデータ操作 (DML) オペレーションは、バッファプール内のページにアクセスします。読み取りを誘発できるステートメントには、SELECT、UPDATE、DELETEがあります。例えば、UPDATEは、テーブルまたはインデックスからページを読み取ることができます。要求または更新中のページが共有バッファプールにない場合、この読み取りはIO:DataFileReadイベントにつながることがあります。

共有バッファプールは有限のため、いっぱいになる可能性があります。この場合、メモリ上にないページをリクエストすると、データベースは強制的にディスクからブロックを読み取ることになります。IO:DataFileReadイベントが頻繁に発生する場合は、共有バッファプールが小さすぎるとワークロードに対応できない可能性があります。この問題は、バッファプールに収まらない多数の行読み取るSELECTクエリでは深刻です。バッファプールの詳細については、「バッファプール」を参照してください。

待機時間が増加する原因の可能性

IO:DataFileReadイベントの一般的な原因は以下のとおりです。

接続スパイク

複数の接続で同じ数の IO:DataFileRead 待機イベントが発生することがあります。この場合、スパイク (突然大きく増加) が IO:DataFileRead イベントで発生する可能性があります。

シーケンシャルスキャンを実行する SELECT および DML ステートメント

アプリケーションが新しいオペレーションを実行している可能性があります。または、新しい実行計画のために既存の操作がオペレーションされる可能性があります。このような場合は、seq_scan値より大きいテーブル (特に大きなテーブル) を探します。pg_stat_user_tablesクエリでそれらを探してください。より多くの読み取りオペレーションを生成しているクエリを追跡するには、エクステンションpg_stat_statementsを使用します。

大規模なデータセットの CTAS および CREATE INDEX

CTASはCREATE TABLE AS SELECTステートメントです。大規模なデータセットを出典として使用して CTAS を実行する場合、または大きなテーブルにインデックスを作成する場合は、IO:DataFileReadイベントが発生する可能性があります。インデックスを作成するとき、データベースはシーケンシャルスキャンを使用してオブジェクト全体を読み取る必要があります。CTAS は、ページがメモリ上にないときにIO:DataFileリードを生成します。

複数のバキュームワーカーが同時に実行されている

バキュームワーカーは、マニュアルまたは自動でトリガーできます。積極的なバキューム戦略の採用をお勧めします。ただし、テーブルに多数の更新または削除された行がある場合、IO:DataFileRead待機が増加します。スペース確保後、IO:DataFileReadに費やすバキューム時間が減少します。

大量データの取り込み

アプリケーションで大量のデータを取り込むと、ANALYZEオペレーションが頻繁に発生する可能性があります。ANALYZEプロセスは、オートバキュームランチャーによって、あるいはマニュアルでトリガーすることができます。

ANALYZEオペレーションは、テーブルのサブセットを読み取ります。30にdefault_statistics_target値を掛けたものがスキャンを要するページ数です。詳細については、PostgreSQL ドキュメントをご参照ください。default_statistics_targetパラメータは 1 ~ 10,000 の範囲の値を指定でき、デフォルトは 100 です。

リソースの枯渇

インスタンスのネットワーク帯域幅や CPU が消費されると、IO:DataFileReadイベントはより頻繁に発生する可能性があります。

アクション

待機イベントの原因に応じたさまざまなアクションをお勧めします。

トピック

- 待機を生成するクエリの述語フィルターをチェックする
- メンテナンス作業の影響を最小化する
- 多数の接続に対応する

待機を生成するクエリの述語フィルターをチェックする

IO:DataFileRead待機イベントを生成する特定のクエリを特定するとします。これらは、次の方法を使用して識別できることがあります。

- Performance Insights
- エクステンションpg_stat_statementsで提供されるようなカタログビュー
- カタログビューpg_stat_all_tablesで、定期的な物理読み取り回数の増加を示す場合
- pg_statio_all_tablesビューで、_readカウンターの増加が示されている場合

これらのクエリの述語 (WHERE 節) でどのフィルターが使用されるかを決定することをお勧めします。次のガイドラインに従ってください。

- EXPLAIN コマンドを実行します。出力では、使用されているスキャンのタイプを特定します。シーケンシャルスキャンは必ずしも問題を示すわけではありません。シーケンシャルスキャンを使用するクエリは、フィルターを使用するクエリと比較して、自然により多くのIO:DataFileReadイベントを生成します。

  WHERE節に記載された列がインデックスされているかどうかを確認します。されていない場合、この列のインデックスの作成を検討してください。この方法では、シーケンシャルスキャンを回避し、IO:DataFileReadイベントの発生を減らすことができます。制限付きフィルターがあってもシーケンシャルスキャンが実行される場合は、適切なインデックスが使用されているかどうかを評価します。

- クエリが非常に大きなテーブルにアクセスしているかどうかを確認します。場合によっては、テーブルをパーティション化するとクエリで必要なパーティションのみを読み取ることができ、パフォーマンスが向上することがあります。
- ジョイント操作からカーディナリティ (行の合計数) を検証します。フィルターに渡すWHERE節の値がどれほど制限的であるかに注意してください。可能であれば、クエリをチューニングして、計画の各ステップで渡される行数を減らします。

メンテナンス作業の影響を最小化する

VACUUMやANALYZEのようなメンテナンスオペレーションは重要です。これらのメンテナンス作業に関連するIO:DataFileRead待機イベントを見つけても、それらをオフにしないことをお勧めします。次のようなアプローチにより、これらの操作の影響を最小限に抑えることができます。

- オフピーク時にメンテナンス操作をマニュアルで実行します。この方法では、データベースが自動操作のしきい値に達するのを防ぎます。
- 非常に大きなテーブルの場合は、テーブルのパーティション化を検討してください。この方法により、メンテナンスオペレーションのオーバーヘッドが削減されます。データベースは、メンテナンスが必要なパーティションにのみアクセスします。
- 大量のデータを取り込む場合は、自動分析機能を無効にすることを検討してください。

オートバキューム機能は、次の数式が真の場合、テーブルに対して自動的にトリガーされます。

```
pg_stat_user_tables.n_dead_tup > (pg_class.reltuples x autovacuum_vacuum_scale_factor)
 + autovacuum_vacuum_threshold
```

ビューpg_stat_user_tablesとカタログpg_classには複数の行があります。1 行は、テーブル内の 1 つの行に対応できます。この公式は、reltuplesが特定のテーブル用だと仮定しています。パラメータ autovacuum_vacuum_scale_factor (デフォルトは 0.20) と autovacuum_vacuum_threshold (デフォルトでは 50 タプル) は通常、インスタンス全体に対してグローバルに設定されます。ただし、特定のテーブルに対して異なる値を設定できます。

トピック

- 不要な領域を消費しているテーブルを探す
- 不要なスペースを消費しているインデックスを見つける
- オートバキュームの対象となるテーブルを見つける

不要な領域を消費しているテーブルを探す

必要以上の領域を消費しているテーブルを見つけるには、次のクエリを実行します。このクエリが rds_superuser ロールを持たないデータベースユーザーロールによって実行されると、そのユーザーロールが読み取り権限を持っているテーブルに関する情報のみが返されます。このクエリは、PostgreSQL バージョン 12 以降のバージョンでサポートされています。

```
WITH report AS (
   SELECT   schemaname
           ,tblname
           ,n_dead_tup
           ,n_live_tup
           ,block_size*tblpages AS real_size
           ,(tblpages-est_tblpages)*block_size AS extra_size
           ,CASE WHEN tblpages - est_tblpages > 0
              THEN 100 * (tblpages - est_tblpages)/tblpages::float
              ELSE 0
            END AS extra_ratio, fillfactor, (tblpages-est_tblpages_ff)*block_size AS
 bloat_size
           ,CASE WHEN tblpages - est_tblpages_ff > 0
              THEN 100 * (tblpages - est_tblpages_ff)/tblpages::float
              ELSE 0
            END AS bloat_ratio
           ,is_na
    FROM (
          SELECT  ceil( reltuples / ( (block_size-page_hdr)/tpl_size ) ) +
 ceil( toasttuples / 4 ) AS est_tblpages
                 ,ceil( reltuples / ( (block_size-page_hdr)*fillfactor/
(tpl_size*100) ) ) + ceil( toasttuples / 4 ) AS est_tblpages_ff
                 ,tblpages
                 ,fillfactor
                 ,block_size
                 ,tblid
                 ,schemaname
                 ,tblname
                 ,n_dead_tup
                 ,n_live_tup
                 ,heappages
                 ,toastpages
                 ,is_na
            FROM (
                   SELECT ( 4 + tpl_hdr_size + tpl_data_size + (2*ma)
```

```
                                - CASE WHEN tpl_hdr_size%ma = 0 THEN ma ELSE
 tpl_hdr_size%ma END
                                - CASE WHEN ceil(tpl_data_size)::int%ma = 0 THEN ma ELSE
 ceil(tpl_data_size)::int%ma END
                             ) AS tpl_size
                             ,block_size - page_hdr AS size_per_block
                             ,(heappages + toastpages) AS tblpages
                             ,heappages
                             ,toastpages
                             ,reltuples
                             ,toasttuples
                             ,block_size
                             ,page_hdr
                             ,tblid
                             ,schemaname
                             ,tblname
                             ,fillfactor
                             ,is_na
                             ,n_dead_tup
                             ,n_live_tup
                           FROM (
                                SELECT  tbl.oid                       AS tblid
                                       ,ns.nspname                    AS schemaname
                                       ,tbl.relname                   AS tblname
                                       ,tbl.reltuples                 AS reltuples
                                       ,tbl.relpages                  AS heappages
                                       ,coalesce(toast.relpages, 0)   AS toastpages
                                       ,coalesce(toast.reltuples, 0)  AS toasttuples
                                       ,psat.n_dead_tup               AS n_dead_tup
                                       ,psat.n_live_tup               AS n_live_tup
                                       ,24                            AS page_hdr
                                       ,current_setting('block_size')::numeric AS
 block_size

 ,coalesce(substring( array_to_string(tbl.reloptions, ' ') FROM
 'fillfactor=([0-9]+)')::smallint, 100) AS fillfactor
                                       ,CASE WHEN version()~'mingw32' OR version()~'64-
bit|x86_64|ppc64|ia64|amd64' THEN 8 ELSE 4 END        AS ma
                                       ,23 + CASE WHEN MAX(coalesce(null_frac,0)) > 0
 THEN ( 7 + count(*) ) / 8 ELSE 0::int END              AS tpl_hdr_size
                                       ,sum( (1-coalesce(s.null_frac, 0)) *
 coalesce(s.avg_width, 1024) )                                  AS tpl_data_size
                                       ,bool_or(att.atttypid =
 'pg_catalog.name'::regtype) OR count(att.attname) <> count(s.attname)         AS is_na
```

```
                                    FROM  pg_attribute       AS att
                                    JOIN  pg_class           AS tbl    ON (att.attrelid =
 tbl.oid)
                                    JOIN  pg_stat_all_tables AS psat   ON (tbl.oid =
 psat.relid)
                                    JOIN  pg_namespace       AS ns     ON (ns.oid =
 tbl.relnamespace)
                               LEFT JOIN  pg_stats           AS s      ON
 (s.schemaname=ns.nspname AND s.tablename = tbl.relname AND s.inherited=false AND
 s.attname=att.attname)
                               LEFT JOIN  pg_class           AS toast  ON
 (tbl.reltoastrelid = toast.oid)
                                   WHERE  att.attnum > 0
                                     AND  NOT att.attisdropped
                                     AND  tbl.relkind = 'r'
                                GROUP BY  tbl.oid, ns.nspname, tbl.relname,
 tbl.reltuples, tbl.relpages, toastpages, toasttuples, fillfactor, block_size, ma,
 n_dead_tup, n_live_tup
                                ORDER BY  schemaname, tblname
                              ) AS s
                  ) AS s2
       ) AS s3
 ORDER BY bloat_size DESC
)
  SELECT *
    FROM report
   WHERE bloat_ratio != 0
 -- AND schemaname = 'public'
 -- AND tblname = 'pgbench_accounts'
;

-- WHERE NOT is_na
--   AND tblpages*((pst).free_percent + (pst).dead_tuple_percent)::float4/100 >= 1
```

アプリケーション内のテーブルとインデックスの肥大化をチェックできます。詳細については、「

PostgreSQL のマルチバージョン同時実行制御 (MVCC) を使用すると、データの整合性を維持できます。PostgreSQL MVCC は、トランザクションがコミットまたはロールバックされるまで、更新または削除された行 (タプルとも呼ばれます) の内部コピーを保存することによって機能します。この保存された内部コピーはユーザーからは見えません。ただし、これらの非表示のコピーが VACUUM または AUTOVACUUM ユーティリティによって定期的にクリーンアップされない場合、テーブルが肥

大化する可能性があります。テーブルの肥大化をチェックしないと、ストレージコストの増加や処理速度の低下につながる可能性があります。

多くの場合、Aurora の VACUUM または AUTOVACUUM のデフォルト設定は、不要なテーブルの肥大化を処理するのに十分なものです。ただし、次のような状況がアプリケーションで発生している場合は、肥大化がないかどうかを確認することをお勧めします。

- VACUUM プロセスの間で、比較的短時間で多数のトランザクションを処理している。

- パフォーマンスが低下し、ストレージが不足している。

はじめに、dead タプルでどの程度のスペースを使用しているか、テーブルとインデックスの肥大化をクリーンアップすることでどれだけ回復が見込めるか、最大限正確な情報を収集してください。そのためには、pgstattuple 拡張機能を使用して、Aurora クラスターの統計情報を収集します。詳細については、「pgstattuple」を参照してください。pgstattuple 拡張機能を使用する権限は、pg_stat_scan_tables ロールとデータベースのスーパーユーザーに限定されます。

Aurora で pgstattuple 拡張機能を作成するには、クライアントセッション (psql や pgAdmin など) をクラスターに接続し、次のコマンドを使用します。

```
CREATE EXTENSION pgstattuple;
```

プロファイルする各データベースで、拡張機能を作成します。拡張機能を作成したら、コマンドラインインターフェイス (CLI) を使用して、使用できないスペースをどの程度再利用できるかを測定します。統計を収集する前に、AUTOVACUUM を 0 に設定してクラスターパラメータグループを変更します。0 に設定すると、アプリケーションによって残されたデットタプルを Aurora が自動的にクリーンアップすることを防止しますが、結果の精度に影響を与える可能性があります。次のコマンドを入力して、単純なテーブルを作成します。

```
postgres=> CREATE TABLE lab AS SELECT generate_series (0,100000);
```

```
SELECT 100001
```

次の例では、DB クラスターの AUTOVACUUM をオンにしてクエリを実行しています。dead_tuple_count は 0 で、これは AUTOVACUUM が PostgreSQL データベースから古いデータやタプルを削除したことを示します。

pgstattuple を使用してテーブルに関する情報を収集するには、クエリにテーブルの名前またはオブジェクト識別子 (OID) を指定します。

```
postgres=> SELECT * FROM pgstattuple('lab');
```

```
table_len  | tuple_count | tuple_len | tuple_percent | dead_tuple_count |
 dead_tuple_len | dead_tuple_percent | free_space | free_percent
-----------+-------------+-----------+---------------+------------------
+----------------+--------------------+------------+--------------
3629056    | 100001      | 2800028   | 77.16         | 0                | 0
  | 0                 | 16616      | 0.46
(1 row)
```

次のクエリでは、AUTOVACUUM をオフにして、テーブルから 25,000 行を削除するコマンドを入力しています。その結果、dead_tuple_count は 25000 に増加しました。

```
postgres=> DELETE FROM lab WHERE generate_series < 25000;
```

```
DELETE 25000
```

```
SELECT * FROM pgstattuple('lab');
```

アプリケーションを中断せずに肥大化をモニタリングする

Aurora クラスターの設定は、ほとんどのワークロードでベストプラクティスが実現できるように最適化されています。ただし、アプリケーションや使用パターンに合わせてクラスターを最適化する場合もあります。この場合、ビジー状態のアプリケーションを中断することなく pgstattuple 拡張機能を使用できます。そのためには、次のステップを実行します。

1. Aurora インスタンスのクローンを作成します。
2. パラメータファイルを変更して、クローンの AUTOVACUUM をオフにします。
3. サンプルワークロードまたは pgbench (PostgreSQL でベンチマークテストを実行するためのプログラム) を使用してクローンをテストしながら pgstattuple クエリを実行します。詳細については、「pgbench」を参照してください。

アプリケーションを実行して結果を確認したら、復元したコピーに pg_repack または VACUUM FULL を使用して違いを比較します。dead_tuple_count、dead_tuple_len、dead_tuple_percent のいずれかが大幅に減少している場合は、本稼働クラスターのバキュームスケジュールを調整して肥大化を最小限に抑えています。

テンポラリテーブルでの肥大化の回避

アプリケーションでテンポラリテーブルを作成する場合、そのテンポラリテーブルが不要になった場合はアプリケーションから削除してください。自動バキュームプロセスでは、テンポラリテーブルを検索しません。テンポラリテーブルをそのままにしておくと、データベースがすぐに肥大化してしまう可能性があります。さらに、この肥大化はシステムテーブルにまで及ぶ可能性があります。システムテーブルは、pg_attribute や pg_depend などの PostgreSQL オブジェクトや属性を追跡する内部テーブルです。

テンポラリテーブルが不要になったら、TRUNCATE ステートメントを使用してテーブルを空にして、スペースを空けることができます。次に、pg_attribute テーブルと pg_depend テーブルのバキューム処理を手動で実行します。これらのテーブルをバキューム処理することで、テンポラリテーブルを継続的に作成、切り捨て、削除しても、タプルが増えることでシステムが肥大化することがなくなります。

次の構文を追加して、コンテンツがコミットされた際に新しい行を削除することで、テンポラリテーブルを作成する際にこの問題を回避できます。

```
CREATE TEMP TABLE IF NOT EXISTS table_name(table_description) ON COMMIT DELETE ROWS;
```

ON COMMIT DELETE ROWS 句は、トランザクションがコミットされたときにテンポラリテーブルを切り捨てます。

インデックスの肥大化の回避

テーブルのインデックス付きフィールドを変更した場合、インデックスを更新すると、そのインデックスに 1 つまたは複数のデットタプルが生成されます。デフォルトでは、自動バキューム処理によってインデックスの肥大化をクリーンアップしますが、このクリーンアップにはかなりの時間とリソースが必要です。テーブルの作成時にインデックスクリーンアップ設定を指定するには、vacuum_index_cleanup 句を含めてください。デフォルトでは、テーブルの作成時にこの句は AUTO に設定されます。つまり、サーバーは、テーブルのバキューム処理時にインデックスのクリーンアップが必要かどうかを判断します。句を ON に設定すると、特定のテーブルのインデックスクリーンアップを有効になります。OFF に設定すると、そのテーブルのインデックスクリーンアップを無効にできます。インデックスのクリーンアップをオフにすると時間を節約できる可能性がありますが、インデックスが肥大化する可能性があることに注意してください。

コマンドラインでテーブルのバキューム処理を行うと、インデックスのクリーンアップを手動で制御できます。テーブルをバキュームしてデッドタプルをインデックスから削除するには、INDEX_CLEANUP 句を ON の値とテーブル名で記述します。

```
acctg=> VACUUM (INDEX_CLEANUP ON) receivables;
```

```

```

```
INFO: aggressively vacuuming "public.receivables"
```

```
VACUUM
```

インデックスを消去せずにテーブルをバキューム処理するには、OFF の値を指定します。

```
acctg=> VACUUM (INDEX_CLEANUP OFF) receivables;
```

```

```

```
INFO: aggressively vacuuming "public.receivables"
```

```
VACUUM
```

」を参照してください。

不要なスペースを消費しているインデックスを見つける

IO:DataFileRead 2391

不要な領域を消費しているインデックスを見つけるには、次のクエリを実行します。

```
-- WARNING: run with a nonsuperuser role, the query inspects
```

```
-- WARNING: rows with is_na = 't' are known to have bad statistics ("name" type is not
 supported).
-- This query is compatible with PostgreSQL 8.2 and later.

SELECT current_database(), nspname AS schemaname, tblname, idxname,
 bs*(relpages)::bigint AS real_size,
  bs*(relpages-est_pages)::bigint AS extra_size,
  100 * (relpages-est_pages)::float / relpages AS extra_ratio,
  fillfactor, bs*(relpages-est_pages_ff) AS bloat_size,
  100 * (relpages-est_pages_ff)::float / relpages AS bloat_ratio,
  is_na
  -- , 100-(sub.pst).avg_leaf_density, est_pages, index_tuple_hdr_bm,
  -- maxalign, pagehdr, nulldatawidth, nulldatahdrwidth, sub.reltuples, sub.relpages
  -- (DEBUG INFO)
FROM (
  SELECT coalesce(1 +
       ceil(reltuples/floor((bs-pageopqdata-pagehdr)/(4+nulldatahdrwidth)::float)), 0
       -- ItemIdData size + computed avg size of a tuple (nulldatahdrwidth)
    ) AS est_pages,
    coalesce(1 +
       ceil(reltuples/floor((bs-pageopqdata-pagehdr)*fillfactor/
(100*(4+nulldatahdrwidth)::float))), 0
    ) AS est_pages_ff,
    bs, nspname, table_oid, tblname, idxname, relpages, fillfactor, is_na
    -- , stattuple.pgstatindex(quote_ident(nspname)||'.'||quote_ident(idxname)) AS
 pst,
    -- index_tuple_hdr_bm, maxalign, pagehdr, nulldatawidth, nulldatahdrwidth,
 reltuples
    -- (DEBUG INFO)
  FROM (
    SELECT maxalign, bs, nspname, tblname, idxname, reltuples, relpages, relam,
 table_oid, fillfactor,
      ( index_tuple_hdr_bm +
          maxalign - CASE -- Add padding to the index tuple header to align on MAXALIGN
            WHEN index_tuple_hdr_bm%maxalign = 0 THEN maxalign
            ELSE index_tuple_hdr_bm%maxalign
          END
        + nulldatawidth + maxalign - CASE -- Add padding to the data to align on
 MAXALIGN
            WHEN nulldatawidth = 0 THEN 0
            WHEN nulldatawidth::integer%maxalign = 0 THEN maxalign
            ELSE nulldatawidth::integer%maxalign
          END
      )::numeric AS nulldatahdrwidth, pagehdr, pageopqdata, is_na
```

```
      -- , index_tuple_hdr_bm, nulldatawidth -- (DEBUG INFO)
    FROM (
      SELECT
        i.nspname, i.tblname, i.idxname, i.reltuples, i.relpages, i.relam, a.attrelid
AS table_oid,
        current_setting('block_size')::numeric AS bs, fillfactor,
        CASE -- MAXALIGN: 4 on 32bits, 8 on 64bits (and mingw32 ?)
          WHEN version() ~ 'mingw32' OR version() ~ '64-bit|x86_64|ppc64|ia64|amd64'
THEN 8
          ELSE 4
        END AS maxalign,
        /* per page header, fixed size: 20 for 7.X, 24 for others */
        24 AS pagehdr,
        /* per page btree opaque data */
        16 AS pageopqdata,
        /* per tuple header: add IndexAttributeBitMapData if some cols are null-able */
        CASE WHEN max(coalesce(s.null_frac,0)) = 0
          THEN 2 -- IndexTupleData size
          ELSE 2 + (( 32 + 8 - 1 ) / 8)
          -- IndexTupleData size + IndexAttributeBitMapData size ( max num filed per
index + 8 - 1 /8)
        END AS index_tuple_hdr_bm,
        /* data len: we remove null values save space using it fractionnal part from
stats */
        sum( (1-coalesce(s.null_frac, 0)) * coalesce(s.avg_width, 1024)) AS
nulldatawidth,
        max( CASE WHEN a.atttypid = 'pg_catalog.name'::regtype THEN 1 ELSE 0 END ) > 0
AS is_na
      FROM pg_attribute AS a
        JOIN (
          SELECT nspname, tbl.relname AS tblname, idx.relname AS idxname,
            idx.reltuples, idx.relpages, idx.relam,
            indrelid, indexrelid, indkey::smallint[] AS attnum,
            coalesce(substring(
              array_to_string(idx.reloptions, ' ')
               from 'fillfactor=([0-9]+)')::smallint, 90) AS fillfactor
          FROM pg_index
            JOIN pg_class idx ON idx.oid=pg_index.indexrelid
            JOIN pg_class tbl ON tbl.oid=pg_index.indrelid
            JOIN pg_namespace ON pg_namespace.oid = idx.relnamespace
          WHERE pg_index.indisvalid AND tbl.relkind = 'r' AND idx.relpages > 0
        ) AS i ON a.attrelid = i.indexrelid
        JOIN pg_stats AS s ON s.schemaname = i.nspname
```

```
          AND ((s.tablename = i.tblname AND s.attname =
 pg_catalog.pg_get_indexdef(a.attrelid, a.attnum, TRUE))
          -- stats from tbl
          OR  (s.tablename = i.idxname AND s.attname = a.attname))
          -- stats from functionnal cols
        JOIN pg_type AS t ON a.atttypid = t.oid
      WHERE a.attnum > 0
      GROUP BY 1, 2, 3, 4, 5, 6, 7, 8, 9
    ) AS s1
  ) AS s2
    JOIN pg_am am ON s2.relam = am.oid WHERE am.amname = 'btree'
) AS sub
-- WHERE NOT is_na
ORDER BY 2,3,4;
```

オートバキュームの対象となるテーブルを見つける

自動バキュームの対象となるテーブルを見つけるには、次のクエリを実行します。

```
--This query shows tables that need vacuuming and are eligible candidates.
--The following query lists all tables that are due to be processed by autovacuum.
-- During normal operation, this query should return very little.
WITH  vbt AS (SELECT setting AS autovacuum_vacuum_threshold
              FROM pg_settings WHERE name = 'autovacuum_vacuum_threshold')
    , vsf AS (SELECT setting AS autovacuum_vacuum_scale_factor
              FROM pg_settings WHERE name = 'autovacuum_vacuum_scale_factor')
    , fma AS (SELECT setting AS autovacuum_freeze_max_age
              FROM pg_settings WHERE name = 'autovacuum_freeze_max_age')
    , sto AS (SELECT opt_oid, split_part(setting, '=', 1) as param,
                split_part(setting, '=', 2) as value
              FROM (SELECT oid opt_oid, unnest(reloptions) setting FROM pg_class) opt)
SELECT
    '"'||ns.nspname||'"."'||c.relname||'"' as relation
    , pg_size_pretty(pg_table_size(c.oid)) as table_size
    , age(relfrozenxid) as xid_age
    , coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
 autovacuum_freeze_max_age
    , (coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
        coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) *
 c.reltuples)
        as autovacuum_vacuum_tuples
    , n_dead_tup as dead_tuples
FROM pg_class c
JOIN pg_namespace ns ON ns.oid = c.relnamespace
```

```
JOIN pg_stat_all_tables stat ON stat.relid = c.oid
JOIN vbt on (1=1)
JOIN vsf ON (1=1)
JOIN fma on (1=1)
LEFT JOIN sto cvbt ON cvbt.param = 'autovacuum_vacuum_threshold' AND c.oid =
 cvbt.opt_oid
LEFT JOIN sto cvsf ON cvsf.param = 'autovacuum_vacuum_scale_factor' AND c.oid =
 cvsf.opt_oid
LEFT JOIN sto cfma ON cfma.param = 'autovacuum_freeze_max_age' AND c.oid = cfma.opt_oid
WHERE c.relkind = 'r'
AND nspname <> 'pg_catalog'
AND (
    age(relfrozenxid) >= coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
    or
    coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
      coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) * c.reltuples
 <= n_dead_tup
    -- or 1 = 1
)
ORDER BY age(relfrozenxid) DESC;
```

多数の接続に対応する

Amazon CloudWatch をモニタリングすると、DatabaseConnectionsメトリックスパイクが見つかることがあります。この増加は、データベースへの接続数が増加していることを示します。次のようなアプローチを推奨します。

- アプリケーションが各インスタンスで開くことができる接続の数を制限します。アプリケーションが組み込み接続プール機能を備えている場合は、適切な数の接続を設定します。インスタンス内の vCPUs が効果的にパラレル化できる数値を基準にします。

  アプリケーションで接続プール機能を使用しない場合は、Amazon RDS プロキシまたは代替の使用を検討してください。このアプローチにより、アプリケーションはロードバランサーとの複数の接続を開くことができます。その後、バランサーは、データベースとの制限された数の接続を開くことができます。パラレルで実行される接続が少なくなると、DB インスタンスのカーネル内のコンテキスト切り替えが減少します。クエリの進行が速くなり、待機イベントが減少するはずです。詳細については、「Amazon RDS Proxy for Aurora の使用」を参照してください。
- 可能であれば、Aurora PostgreSQL のリーダーノードと RDS for PostgreSQL のリードレプリカを活用してください。アプリケーションが読み取り専用のオペレーションを実行するときは、これらのリクエストを読み取り専用のエンドポイントに送信します。この方法は、アプリケーションのリクエストをすべてのリーダーノードに分散させ、ライターノードの I/O 負荷を軽減します。

- DB インスタンスのスケールアップを検討します。大容量のインスタンスクラスはより多くのメモリを提供するため、Aurora PostgreSQL ではページを保持するためのより大きな共有バッファプールを提供します。サイズが大きければ、DB インスタンスが接続処理する vCPUs も多くなります。特に、IO:DataFileRead待機イベントを発生させているオペレーションが書き込みの場合、vCPU の増設は有効です。

IO:XactSync

IO:XactSyncイベントは、データベースが Aurora ストレージサブシステムが通常のトランザクションのコミットを承認するか、準備済みトランザクションのコミットまたはロールバックの承認を待機中に発生します 準備済みトランザクションは、PostgreSQL の 二相コミットサポートの一部です。

トピック

- サポート対象エンジンバージョン
- Context
- 待機時間が増加する原因の可能性
- アクション

サポート対象エンジンバージョン

この待機イベント情報は、Aurora PostgreSQL のすべてのバージョンでサポートされています。

Context

イベントIO:XactSyncは、インスタンスが、 Aurora ストレージサブシステムがトランザクションデータの処理の確認を待つ時間を費やしていることを示します。

待機時間が増加する原因の可能性

IO:XactSyncイベントが通常より頻繁に発生する場合は、パフォーマンスの問題を示していることがあります。典型的な原因は次のとおりです。

ネットワークの飽和

クライアントと DB インスタンス間のトラフィック、またはストレージサブシステムへのトラフィックが、ネットワーク帯域幅に対して重すぎる可能性があります。

CPU プレッシャー

ワークロードが重いと、Aurora ストレージデーモンが十分な CPU 時間を得られていない可能性があります。

アクション

待機イベントの原因に応じたさまざまなアクションをお勧めします。

トピック

- リソースをモニタリングする
- CPU のスケールアップ
- ネットワーク帯域幅の増加
- コミットの回数を減らす

リソースをモニタリングする

`IO:XactSync`イベント増加の原因を特定するには、次のメトリックをチェックします。

- `WriteThroughput`および`CommitThroughput` - 書き込みスループットまたはコミットスループットの変化は、ワークロードの増加を示している可能性があります。
- `WriteLatency`および`CommitLatency` - 書き込みレイテンシーまたはコミットレイテンシーの変更は、ストレージサブシステムがより多くの作業の実行を要求されてていることを示している可能性があります。
- `CPUUtilization` - インスタンスの CPU 使用率が 90% を超える場合、Aurora ストレージデーモンが CPU 上で十分な時間を得ていない可能性があります。この場合、I/O パフォーマンスが低下します。

これらのメトリクスの詳細については、Amazon Aurora のインスタンスレベルのメトリクスを参照してください。

CPU のスケールアップ

CPU 不足の問題に対処するには、CPU 容量が多いインスタンスタイプへの変更を検討してください。DB インスタンスクラスの CPU 容量については、「Aurora 用の DB インスタンスクラスのハードウェア仕様」を参照してください。

ネットワーク帯域幅の増加

インスタンスがネットワーク帯域幅制限に達しているかどうかを判断するには、次の他の待機イベントを確認します。

- `IO:DataFileRead`、`IO:BufferRead`、`IO:BufferWrite`、および`IO:XactWrite`— 大量の I/O を使用するクエリは、これらの待機イベントをより多く発生させることがあります。
- `Client:ClientRead`および`Client:ClientWrite` - 大量のクライアント通信を持つクエリでは、これらの待機イベントをより多く発生させることがあります。

ネットワーク帯域幅に問題がある場合は、より多くのネットワーク帯域幅を持つインスタンスタイプへの変更を検討してください。DB インスタンスクラスのネットワークパフォーマンスについては、「Aurora 用の DB インスタンスクラスのハードウェア仕様」を参照してください。

コミットの回数を減らす

コミットの数を減らすには、ステートメントをトランザクションブロックにまとめます。

IPC:DamRecordTxAck

`IPC:DamRecordTxAck`イベントは、データベースアクティビティストリーミングを使用するセッションで Aurora PostgreSQL がアクティビティストリーミングイベントを生成し、そのイベントが耐久性の高いを持つのを待機中に発生します。

トピック

- 関連するエンジンのバージョン
- Context
- 原因
- アクション

関連するエンジンのバージョン

この待機イベント情報は、すべての Aurora PostgreSQL 10.7 以降の 10 バージョン、11.4 以降の 11 バージョン、および 12 と 13 のすべてのバージョンに関連しています。

Context

同期モードでは、データベースのパフォーマンスよりもアクティビティストリーミングイベントの持続性が優先されます。イベントの耐久性の高い書き込みを待機中、セッションは他のデータベースアクティビティをブロックし、IPC:DamRecordTxAck待機イベントを発生させます。

原因

IPC:DamRecordTxAck最上位待機にイベントが表示される最も一般的な原因は、データベースアクティビティストリーミング (DAS) 機能が総合的な監査であることです。SQL アクティビティが高くなるほど、記録の必要があるアクティビティストリーミングイベントが発生します。

アクション

待機イベントの原因に応じたさまざまなアクションを実行することをお勧めします。

- SQL ステートメントの数を減らすか、データベースアクティビティストリーミングをオフにします。これにより、耐久性の高い書き込みが必要なイベント数が減ります。
- 非同期モードに変更します。これにより、IPC:DamRecordTxAck待機イベントでの競合を減らすのに役立ちます。

  ただし、DAS 機能は、非同期モードではすべてのイベントの耐久性の高いを保証することはできません。

Lock:advisory

Lock:advisoryイベントは、PostgreSQL アプリケーションがロックを使用して複数のセッション全体のアクティビティを調整するときに発生します。

トピック

関連するエンジンのバージョン

この待機イベント情報は、Aurora PostgreSQL バージョン 9.6 以降に関連しています。

Context

PostgreSQL アドバイザリロックは、ユーザーのアプリケーションコードによって明示的にロックおよびロック解除を実行するアプリケーションレベルの協調的ロックです。アプリケーションは PostgreSQL アドバイザリロックを使用して、複数のセッションにまたがるアクティビティを調整できます。通常のオブジェクトレベルまたは行レベルのロックとは異なり、アプリケーションはロックのライフタイムを完全に制御できます。詳細については、PostgreSQL ドキュメントの Advisory Locks (アドバイザリロック) を参照してください。

アドバイザリロックは、トランザクションが終了する前に解放されるか、トランザクション間のセッションで保持されます。これは、CREATE INDEXステートメントによって取得されたテーブルへのアクセス排他ロックなど、暗黙のうちにシステムで強制されるロックには当てはまりません。

アドバイザリロックの取得 (ロック) およびリリース (ロック解除) に使用される関数の説明については、「PostgreSQL のドキュメント」のアドバイザリロックの関数を参照してください。

アドバイザリロックは、通常の PostgreSQL ロックシステムの上に実装され、pg_locksシステムビューで表示できます。

原因

このロックタイプは、明示的に使用するアプリケーションによって排他的に制御されます。クエリの一部として各行に対して取得されるアドバイザリロックは、ロックの急増や、長期的な蓄積を引き起こすことがあります。

これらの効果は、クエリが返すよりも多くの行でロックを取得する方法でクエリが実行されると発生します。アプリケーションは最終的にすべてのロックを解放する必要がありますが、返されない行でロックが取得された場合、アプリケーションはすべてのロックを見つけることができません。

PostgreSQL のドキュメントの「アドバイザリロック」からの例を紹介します。

```
SELECT pg_advisory_lock(id) FROM foo WHERE id > 12345 LIMIT 100;
```

この例では、LIMIT節がクエリの出力を停止できるのは、内部で行が選択され、その ID 値がロックされた後のみです。これは、データ量の増加により、プランナーが開発中にテストされなかった別の実行プランを選択した場合に突然発生することがあります。この場合の構築アップは、アプリケーションがロックされた各ID値に明示的にpg_advisory_unlockを呼び出すことによって発生します。ただし、この場合、返されなかった行において取得されたロックのセットを見つけることはでき

ません。ロックはセッションレベルで取得されるため、トランザクションの終了時に自動的に解放されません。

ブロックされたロック試行のスパイクは、意図しない競合が原因の可能性があります。このような競合では、アプリケーションの無関係な部分が、誤って同じロック ID スペースを共有します。

アクション

アドバイザリロックのアプリケーション使用状況を確認し、アプリケーションフロー内のいつどこで各タイプのアドバイザリロックが取得および解放されるのか、詳しく説明します。

セッションが取得したロックが多すぎるか、長時間実行しているセッションがロックを早期に解放しないために、ロックの蓄積が遅くなっているかどうかを調べます。pg_terminate_backend(pid)を使用してセッションを終了すると、セッションレベルロックの遅い蓄積を修正できます。

アドバイザリロックを待機中のクライアントがpg_stat_activity、wait_event_type=Lock、wait_event=advisoryに表示されます。同じpidのpg_locksシステムビューへのクエリを実行し、locktype=advisoryとgranted=fを検索することで、特定のロック値を取得できます。

pg_locksに対してgranted=tを持つ同じアドバイザリロックへのクエリを実行することで、ブロックしているセッションを特定することができます。

```
SELECT blocked_locks.pid AS blocked_pid,
         blocking_locks.pid AS blocking_pid,
         blocked_activity.usename AS blocked_user,
         blocking_activity.usename AS blocking_user,
         now() - blocked_activity.xact_start AS blocked_transaction_duration,
         now() - blocking_activity.xact_start AS blocking_transaction_duration,
         concat(blocked_activity.wait_event_type,':',blocked_activity.wait_event) AS
 blocked_wait_event,
         concat(blocking_activity.wait_event_type,':',blocking_activity.wait_event) AS
 blocking_wait_event,
         blocked_activity.state AS blocked_state,
         blocking_activity.state AS blocking_state,
         blocked_locks.locktype AS blocked_locktype,
         blocking_locks.locktype AS blocking_locktype,
         blocked_activity.query AS blocked_statement,
         blocking_activity.query AS blocking_statement
    FROM pg_catalog.pg_locks blocked_locks
```

```
   JOIN pg_catalog.pg_stat_activity blocked_activity ON blocked_activity.pid =
blocked_locks.pid
   JOIN pg_catalog.pg_locks blocking_locks
       ON blocking_locks.locktype = blocked_locks.locktype
       AND blocking_locks.DATABASE IS NOT DISTINCT FROM blocked_locks.DATABASE
       AND blocking_locks.relation IS NOT DISTINCT FROM blocked_locks.relation
       AND blocking_locks.page IS NOT DISTINCT FROM blocked_locks.page
       AND blocking_locks.tuple IS NOT DISTINCT FROM blocked_locks.tuple
       AND blocking_locks.virtualxid IS NOT DISTINCT FROM blocked_locks.virtualxid
       AND blocking_locks.transactionid IS NOT DISTINCT FROM
blocked_locks.transactionid
       AND blocking_locks.classid IS NOT DISTINCT FROM blocked_locks.classid
       AND blocking_locks.objid IS NOT DISTINCT FROM blocked_locks.objid
       AND blocking_locks.objsubid IS NOT DISTINCT FROM blocked_locks.objsubid
       AND blocking_locks.pid != blocked_locks.pid
   JOIN pg_catalog.pg_stat_activity blocking_activity ON blocking_activity.pid =
blocking_locks.pid
   WHERE NOT blocked_locks.GRANTED;
```

すべてのアドバイザリロック API 関数には、1 つのbigint引数または2つのinteger引数の 2 組の引数があります。

- bigint の引数が 1 つの API 関数では、上位 32 ビットが pg_locks.classid、下位 32 ビットが pg_locks.objid となります。
- integerが2つある API 関数の場合、第 1 引数はpg_locks.classid、第 2 引数はpg_locks.objidとなります。

pg_locks.objsubid値はどの API フォームが使用されたかを示し、1は 1 つのbigint引数、2は 2 つのinteger引数を意味します。

Lock:extend

Lock:extendイベントは、バックエンドプロセスがリレーションを拡張するためにロックするのを待っているときに、別のプロセスが同じ目的でそのリレーションをロックしていると発生します。

トピック

- サポート対象エンジンバージョン
- Context
- 待機時間が増加する原因の可能性

- アクション

サポート対象エンジンバージョン

この待機イベント情報は、Aurora PostgreSQL のすべてのバージョンでサポートされています。

Context

イベントLock:extendは、バックエンドプロセスがリレーションの拡張する間、他のバックエンドプロセスがそのリレーションを拡張するのを待っている間にロックを保持することを示しています。リレーションを拡張できるのは一度に 1 つのプロセスだけなので、システムはLock:extend待機イベントを発生させます。INSERT、COPY、UPDATEのオペレーションでこのイベントを生成することができます。

待機時間が増加する原因の可能性

Lock:extendイベントが通常より頻繁に発生する場合は、パフォーマンスの問題を示していることがあります。典型的な原因は次のとおりです。

同じテーブルへの同時挿入または更新の急増

同じテーブルに挿入または更新するクエリの同時セッションが増加する可能性があります。

ネットワーク帯域幅の不足

DB インスタンスのネットワーク帯域幅が、現在のワークロードのストレージ通信ニーズに対して不十分な可能性があります。これは、Lock:extendイベントの増加を引き起こすストレージレイテンシーの原因となることがあります。

アクション

待機イベントの原因に応じたさまざまなアクションをお勧めします。

トピック

- 同じリレーションへの同時挿入と更新を減らす
- ネットワーク帯域幅の増加

同じリレーションへの同時挿入と更新を減らす

まず、tup_inserted、tup_updatedメトリクスの増加と、それに伴うこの待機イベントの増加があるかどうか判断します。その場合は、挿入および更新オペレーションで競合性が高いリレーションをチェックします。これを判断するには、n_tup_insおよびn_tup_updフィールドでの値のpg_stat_all_tablesビューへのクエリを実行します。pg_stat_all_tablesビューの詳細については、PostgreSQL ドキュメントの「pg_stat_statements」を参照してください。

ブロックおよびブロックされたクエリの詳細については、pg_stat_activity次の例のとおりにクエリを実行します。

```
SELECT
    blocked.pid,
    blocked.usename,
    blocked.query,
    blocking.pid AS blocking_id,
    blocking.query AS blocking_query,
    blocking.wait_event AS blocking_wait_event,
    blocking.wait_event_type AS blocking_wait_event_type
FROM pg_stat_activity AS blocked
JOIN pg_stat_activity AS blocking ON blocking.pid = ANY(pg_blocking_pids(blocked.pid))
where
blocked.wait_event = 'extend'
and blocked.wait_event_type = 'Lock';

   pid  | usename  |            query             | blocking_id |
            blocking_query                          | blocking_wait_event |
 blocking_wait_event_type
  ------+----------+------------------------------+-------------
+-------------------------------------------------------------------
+---------------------+--------------------------
   7143 |  myuser  | insert into tab1 values (1); |        4600 | INSERT INTO tab1 (a)
 SELECT s FROM generate_series(1,1000000) s; | DataFileExtend      | IO
```

Lock:extendイベントの増加に寄与するリレーションを特定したら、次の方法を使用して競合を減らします。

- パーティション化によって同じテーブルの競合を減らせるかどうかを調べます。挿入または更新されたタプルを異なるパーティショニングすると、競合を減らすことができます。パーティショニングについては、「pg_partman エクステンションによる PostgreSQL パーティションの管理」を参照してください。

- 待機イベントが主に更新アクティビティによるものである場合は、リレーションのフィルファクタ値を減らすことを検討してください。これにより、更新時に新しいブロックのリクエストを減らすことができます。フィルファクタとは、テーブルページをパッキングするための最大容量を決定する、テーブルの格納パラメータです。これは、ページの総容量に対するパーセンテージで表されます。フィルファクタパラメータの詳細については、PostgreSQL ドキュメントの「CREATE TABLE」を参照してください。

  > **Important**
  >
  > この値を変更すると、ワークロードによってはパフォーマンスに悪影響を及ぼす可能性があるため、フィルファクタを変更する場合は、システムのテストを強くお勧めします。

ネットワーク帯域幅の増加

書き込みレイテンシーが増加しているかどうかを確認するには、`WriteLatency`CloudWatch でメトリクスをチェックします。増加している場合は、Amazon CloudWatch の`WriteThroughput`、`ReadThroughput`メトリクスを使用し、DB クラスターのストレージに関するトラフィックをモニタリングしてください。これらのメトリックは、ネットワーク帯域幅がワークロードのストレージアクティビティに十分かどうかを判断するのに役立ちます。

ネットワーク帯域幅が不足している場合は、増加してください。DB インスタンスがネットワーク帯域幅の制限に達している場合、帯域幅を増やす唯一の方法は DB インスタンスのサイズを大きくすることです。

CloudWatch のメトリクスの詳細については、「Amazon Aurora の Amazon CloudWatch メトリクス」を参照してください。DB インスタンスクラスごとの DB エンジンサポートについては、「Aurora 用の DB インスタンスクラスのハードウェア仕様」を参照してください。

Lock:Relation

`Lock:Relation`イベントは、別のトランザクションによって現在ロックされているテーブルまたはビュー (リレーション) のロックを取得するためにクエリが待っているときに発生します。

トピック

- サポート対象エンジンバージョン
- Context
- 待機時間が増加する原因の可能性

- アクション

サポート対象エンジンバージョン

この待機イベント情報は、Aurora PostgreSQL のすべてのバージョンでサポートされています。

Context

ほとんどの PostgreSQL コマンドは、テーブル内のデータへの同時アクセスを制御するために、暗黙のうちにロックを使用します。また、これらのロックは、アプリケーションコード内でLOCKコマンドによって明示的に使用することもできます。多くのロックモードは互いに互換性がないため、同じオブジェクトにアクセスしようとしているときにトランザクションをブロックすることがあります。これが起こると、Aurora PostgreSQL はLock:Relationイベントを生成します。一般的な例をいくつか以下に示します。

- ACCESS EXCLUSIVEのような排他的なロックは、すべての同時アクセスをブロックできます。DROP TABLE、TRUNCATE、VACUUM FULL、CLUSTERなどのデータ定義言語 (DDL) オペレーションは、暗黙のうちにACCESS EXCLUSIVEロックを取得します。ACCESS EXCLUSIVE は、明示的にモードを指定しない LOCK TABLE ステートメントのデフォルトのロックモードでもあります。
- テーブル上でCREATE INDEX (without CONCURRENT)を使用すると、ROW EXCLUSIVEロックを取得するデータ操作言語 (DML) ステートメントUPDATE、DELETE、INSERTと競合します。

テーブルレベルのロックと競合するロックモードの詳細については、PostgreSQL ドキュメントの「明示的なロック」を参照してください。

ブロックされたクエリとトランザクションは、通常、次のいずれかの方法でブロックを解除します。

- クエリのブロック: アプリケーションがクエリをキャンセルするか、ユーザーがプロセスを終了できます。また、セッションのステートメントタイムアウトやデッドロック検出メカニズムによって、エンジンがクエリを強制終了させることもできます。
- トランザクションのブロック: トランザクションが ROLLBACK または COMMIT を実行すると、トランザクションはブロックを停止します。ロールバックは、クライアントまたはネットワークの問題によってセッションが切断されたり、終了したときにも自動的に行われます。セッションは、データベースエンジンがシャットダウンされたり、システムがメモリ不足になったりしたときに終了できます。

待機時間が増加する原因の可能性

Lock:Relation イベントが通常よりも頻繁に発生する場合、パフォーマンスの問題を示している可能性があります。代表的な原因としては、以下が挙げられます。

テーブルロックの競合による同時セッションの増加

競合するロックモードで同じテーブルをロックするクエリによる同時セッションの数が増加する可能性があります。

メンテナンスオペレーション

VACUUMやANALYZEのようなヘルスメンテナンスオペレーションは、競合するロックの数を大幅に増加させる可能性があります。VACUUM FULLはACCESS EXCLUSIVEのロックを、ANALYSEはSHARE UPDATE EXCLUSIVEのロックを取得します。どちらのタイプのロックも、Lock:Relation待機イベントを引き起こすことがあります。また、マテリアライズドビューのリフレッシュなどのアプリケーションデータのメンテナンスオペレーションも、ブロックされたクエリとトランザクションを増加することもあります。

リーダーインスタンスをロックする

ライターとリーダーが保持しているリレーションロックの間に矛盾がある可能性があります。現在は、ACCESS EXCLUSIVE リレーションロックのみが、リーダーインスタンスにレプリケートされます。ただし、ACCESS EXCLUSIVE リレーションロックは、リーダーが保持する ACCESS SHARE リレーションロックと競合します。これにより、リーダーのロックリレーション待機イベントが増加する可能性があります。

アクション

待機イベントの原因に応じたさまざまなアクションをお勧めします。

トピック

- SQL ステートメントのブロックによる影響を軽減
- メンテナンスオペレーションの影響を最小限に抑える
- リーダーロックをチェックする

SQL ステートメントのブロックによる影響を軽減

SQL ステートメントのブロックによる影響を軽減するには、可能なところではアプリケーションコードを修正します。ブロックを減らすための 2 つの一般的な方法は以下のとおりです。

- NOWAIT オプションを使用する: SELECT や LOCK ステートメントなど、一部の SQL コマンドはこのオプションをサポートしています。NOWAIT指示文は、ロックをすぐに取得できない場合、ロックへのクエリをキャンセルします。この方法は、ブロックされたセッションが、その後ろにあるブロックされたセッションが積み重なるのを防ぐのに役立ちます。

  例えば、トランザクション A がトランザクション B に保持されているロックを待っているとします。ここで、B がトランザクション C によってロックされているテーブルのロックをリクエストすると、トランザクション C が完了するまでトランザクション A がブロックされる可能性があります。ただし、トランザクション B が C のロックを要求するときにNOWAITを使用する場合、トランザクションBは迅速に失敗し、トランザクション A が無期限に待機する必要がないことを保証できます。

- SET lock_timeout を使用する: lock_timeout 値を設定して、SQL ステートメントがリレーションでロックを取得するのを待機する時間を制限します。指定されたタイムアウト時間内にロックが取得されなかった場合、ロックを要求したトランザクションはキャンセルされます。この値はセッションレベルで設定します。

メンテナンスオペレーションの影響を最小限に抑える

VACUUMやANALYZEのようなメンテナンスオペレーションは重要です。これらのメンテナンス作業に関連するLock:Relation待機イベントを見つけても、それらをオフにしないことをお勧めします。次のようなアプローチにより、これらの操作の影響を最小限に抑えることができます。

- オフピーク時にメンテナンス操作をマニュアルで実行します。
- オートバキュームタスクによるLock:Relation待機をへらすには、必要なオートバキュームチューニングを実行します。オートバキュームのチューニングについては、Amazon RDS ユーザーガイドの「Amazon RDS での PostgreSQL オートバキュームの使用」を参照してください。

リーダーロックをチェックする

ライターとリーダーの同時セッションが、お互いをブロックするロックを保持しているかどうかを確認できます。これを行う 1 つの方法は、ロックのタイプとリレーションを返すクエリを実行することです。この表では、ライターセッション (左側の列) とリーダーセッション (右側の列) という 2 つの同時セッションに対する一連のクエリを見つけることができます。

再生プロセスは、リーダークエリをキャンセルする前に max_standby_streaming_delay の期間待機します。例に示すように、100 ms のロックタイムアウトは、デフォルトの

max_standby_streaming_delay である30 秒をはるかに下回っています。ロックは問題になる前にタイムアウトします。

ライターセッション

```
export WRITER=aurorapg1.1234567891
0.us-west-1.rds.amazonaws.com

psql -h $WRITER
psql (15devel, server 10.14)
Type "help" for help.
```

リーダーセッション

```
export READER=aurorapg2.1234567891
0.us-west-1.rds.amazonaws.com

psql -h $READER
psql (15devel, server 10.14)
Type "help" for help.
```

ライターセッションはt1ライターインスタンスにテーブルを作成します。ACCESS EXCLUSIVE ロックは、ライターで競合するクエリがないことを前提に、ライターで即座に取得されます。

```
postgres=> CREATE TABLE t1(b
 integer);
CREATE TABLE
```

リーダーセッションは 100 ミリ秒のロックタイムアウトインターバルを設定します。

```
postgres=> SET lock_timeout=100;
SET
```

リーダーセッションはt1リーダーインスタンスでテーブルからデータを読み込もうとします。

```
postgres=> SELECT * FROM t1;
 b
---
(0 rows)
```

ライターセッションがt1をドロップします。

```
postgres=> BEGIN;
BEGIN
postgres=> DROP TABLE t1;
DROP TABLE
postgres=>
```

ライターセッション リーダーセッション

クエリがタイムアウトし、リーダーでキャンセルされます。

```
postgres=> SELECT * FROM t1;
ERROR:  canceling statement due to
 lock timeout
LINE 1: SELECT * FROM t1;
                      ^
```

リーダーセッションがpg_locksおよびpg_stat_activity へのクエリを実行し、エラーの原因を特定します。その結果、aurora wal replayプロセスがテーブルt1でACCESS EXCLUSIVE をホールドしていることを示します。

```
postgres=> SELECT locktype, relation,
 mode, backend_type
postgres-> FROM pg_locks l, pg_stat_a
ctivity t1
postgres-> WHERE l.pid=t1.pid AND
 relation = 't1'::regclass::oid;
 locktype | relation |        mode
    |   backend_type
----------+----------+----------
-----------+------------------
 relation | 68628525 | AccessExc
lusiveLock | aurora wal replay
(1 row)
```

Lock:transactionid

Lock:transactionidイベントは、トランザクションが行レベルのロックを待っているときに発生します。

トピック

- サポート対象エンジンバージョン
- Context
- 待機時間が増加する原因の可能性
- アクション

サポート対象エンジンバージョン

この待機イベント情報は、Aurora PostgreSQL のすべてのバージョンでサポートされています。

Context

イベントLock:transactionidは、トランザクションが、同時に実行されているトランザクションにすでに付与された行レベルのロックを取得しようとすると発生します。を示すセッションは、Lock:transactionid待機イベントがこのロックのためにブロックされていることを示します。COMMITまたはROLLBACKステートメントでブロックされているトランザクションの完了後、ブロックされたトランザクションを続行できます。

Aurora PostgreSQL のマルチバージョン同時実行制御セマンティクスは、リーダーがライターを、ライターがリーダーをブロックしないことを保証します。行レベルの競合が発生するには、ブロックおよびブロックされたトランザクションで、次のタイプの競合するステートメントを発行する必要があります。

- UPDATE
- SELECT … FOR UPDATE
- SELECT … FOR KEY SHARE

ステートメントSELECT … FOR KEY SHAREは特殊なケースです。データベースは、レファレンスの整合性のパフォーマンスを最適化するために、FOR KEY SHARE節を使用します。行に対する行レベルロックは、行を参照している他のテーブルのINSERT、UPDATE、DELETEコマンドをブロックできます。

待機時間が増加する原因の可能性

このイベントが通常よりも頻繁に発生する場合、通常はUPDATE、SELECT … FOR UPDATE、またはSELECT … FOR KEY SHAREステートメントが以下の条件と組み合わさることが原因です。

トピック

- 同時実行数が多い
- トランザクションでのアイドル状態
- トランザクションの実行時間が長い

同時実行数が多い

Aurora PostgreSQL は、きめ細かい行レベルのロックセマンティクスを使用できます。以下の条件が満たされると、行レベルの同時実行が発生する可能性が高くなります。

- 同時性の高いワークロードは、同じ行で同時実行します。
- 同時実行数が増加します。

トランザクションでのアイドル状態

時々、pg_stat_activity.state列にはidle in transaction値が表示されます。この値は、トランザクションをスタートしていても、まだCOMMITまたはROLLBACKを発行していないセッションに表示されます。pg_stat_activity.state値がactiveではない場合、pg_stat_activityに表示されるクエリは、実行を終了した最新のクエリになります。ブロックされているセッションは、開いているトランザクションがロックを保持しているため、クエリを積極的に処理しません。

アイドル状態のトランザクションが行レベルロックを取得した場合は、他のセッションがそのロックを取得するのを妨害する可能性があります。この状態は、待機イベントLock:transactionidの頻発につながります。問題を診断するには、pg_stat_activityそしてpg_locksからの出力を検証します。

トランザクションの実行時間が長い

実行時間が長いトランザクションは、長時間ロックされます。これらの長時間のロックは、他のトランザクションの実行をブロックすることがあります。

アクション

行ロックはUPDATE、SELECT … FOR UPDATE、またはSELECT … FOR KEY SHAREステートメント間の競合です。解決策を試す前に、これらのステートメントが同じ行で実行されているかどうかを調べます。この情報をもとに、次のセクションで説明する戦略を選択してください。

トピック

- 同時実行数の多さに対応
- アイドル状態のトランザクションに対応する
- 長時間実行されるトランザクションへの対応

同時実行数の多さに対応

同時実行数が問題になる場合は、以下から 1 つの方法を試行します。

- アプリケーションの同時実行数を減らします。例えば、アクティブなセッションの数を減らします。
- 接続プールを実装します。RDS プロキシを使用して接続をプールする方法については、「Amazon RDS Proxy for Aurora の使用」を参照してください。
- アプリケーションまたはデータモデルを設計し、UPDATEおよびSELECT … FOR UPDATEステートメントの競合を避けてください。また、SELECT … FOR KEY SHAREステートメントにアクセスされる外部キーの数を減らすこともできます。

アイドル状態のトランザクションに対応する

pg_stat_activity.stateがidle in transactionを示している場合は、以下の方法を使用します。

- 可能な限り、オートコミットをオンにします。この方法では、COMMITまたはROLLBACKを待っている間に、トランザクションが他のトランザクションをブロックすることを防ぎます。
- COMMIT、ROLLBACK、またはENDが不足しているコードパスを検索します。
- アプリケーションの例外処理ロジックに、常に有効なend of transactionへのパスが設定されていることを確認してください。
- COMMITまたはROLLBACKでトランザクションを完了した後、アプリケーションがクエリの結果を処理することを確認します。

長時間実行されるトランザクションへの対応

長時間実行されるトランザクションがLock:transactionidの発生を頻繁に引き起こす場合、以下の方法を試します。

- 長時間実行されるトランザクションが行をロックしないようにします。
- 可能であれば、オートコミットを実装してクエリの長さを制限します。

Lock:tuple

Lock:tuple イベントは、バックエンドプロセスがタプルのロックを取得するのを待っているときに発生します。

トピック

サポート対象エンジンバージョン

この待機イベント情報は、Aurora PostgreSQL のすべてのバージョンでサポートされています。

Context

イベントLock:tupleは、バックエンドがタプルのロック取得を待っている間、別のバックエンドが同じタプルで競合するロックを保持していることを示します。次の表では、セッションがLock:tupleイベントを生成するシナリオを示します。

[Time (時間)	セッション 1	セッション 2	セッション 3
t1	トランザクションをスタートします。		
t2	行 1 を更新します。		
t3		行 1 を更新します。セッションは、タプルの排他ロックを取得してから、セッション1がコミットまたはロールバックによってロックを解放するのを待機します。	
t4			行 1 を更新します。セッションは、セッション 2 がタプルの排他ロックを解放するのを待機します。

または、ベンチマークツールpgbenchを使用してこの待機イベントをシミュレートすることができます。多数の同時セッションを構成して、カスタムSQLファイルでテーブル内の同じ行を更新します。

競合するロックモードの詳細については、「PostgreSQL のドキュメント」の「明示的なロック」を参照してください。pgbenchの詳細については、「PostgreSQL のドキュメント」の「pgbench」を参照してください。

待機時間が増加する原因の可能性

このイベントが通常よりも頻繁に表示される場合、パフォーマンスの問題を示している可能性があり、典型的な原因は次のとおりです。

- UPDATEまたはDELETEステートメントの実行により、同じタプルへの競合するロックを取得しようとしている同時セッションが多数あります。
- 同時実行数の多いセッションでは、FOR UPDATEまたはFOR NO KEY UPDATEロックモードを使用してSELECTステートメントを実行します。
- さまざまな要因により、アプリケーションまたは接続プールが同じオペレーションを実行するため、より多くのセッションを開きます。新しいセッションが同じ行を変更しようとすると、DBロードのスパイクが発生してLock:tupleが表示されることがあります。

詳細については、「PostgreSQL のドキュメント」の「行レベルのロック」を参照してください。

アクション

待機イベントの原因に応じたさまざまなアクションをお勧めします。

トピック

- アプリケーションロジックを調査する
- ブロッカーセッションを見つける
- 同時実行数が多いときにそれを減らす
- ボトルネックのトラブルシューティング

アプリケーションロジックを調査する

ブロッカーセッションが長い間idle in transactionステートメントにあったかどうかを確認します。その場合は、短期的な解決策としてブロッカーセッションの終了を検討してくださ

い。pg_terminate_backend 関数を使用することもできます。この関数の詳細については、「PostgreSQL のドキュメント」の「サーバーシグナリング関数」を参照してください。

長期的な解決策としては、以下を実行してください。

- アプリケーションロジックを調整します。
- idle_in_transaction_session_timeout パラメータを使用します。このパラメータは、指定された時間より長くアイドル状態であったオープントランザクションを持つセッションを終了させます。詳細については、PostgreSQL ドキュメントの 「Client Connection Defaults (クライアント接続のデフォルト)」 を参照してください。
- 可能な限りオートコミットを使用します。詳細については、PostgreSQL ドキュメントの 「Client authentication (クライアント認証)」 を参照してください。

ブロッカーセッションを見つける

Lock:tupleの待機イベントが発生している間に、どのロックが互いに依存しているかを探して、ブロッカーとブロックされたセッションを特定します。詳細については、PostgreSQL wiki にの「依存関係情報のロック」を参照してください。過去のLock:tupleイベントを分析するには、Aurora 関数aurora_stat_backend_waitsを使用します。

次の例では、tupleでフィルタリングしてwait_timeで順序付けし、すべてのセッションへのクエリを実行しています。

```
--AURORA_STAT_BACKEND_WAITS
      SELECT a.pid,
             a.usename,
             a.app_name,
             a.current_query,
             a.current_wait_type,
             a.current_wait_event,
             a.current_state,
             wt.type_name AS wait_type,
             we.event_name AS wait_event,
             a.waits,
             a.wait_time
        FROM (SELECT pid,
                     usename,
                     left(application_name,16) AS app_name,
                     coalesce(wait_event_type,'CPU') AS current_wait_type,
                     coalesce(wait_event,'CPU') AS current_wait_event,
```

```
                    state AS current_state,
                    left(query,80) as current_query,
                    (aurora_stat_backend_waits(pid)).*
                FROM pg_stat_activity
               WHERE pid <> pg_backend_pid()
                 AND usename<>'rdsadmin') a
NATURAL JOIN aurora_stat_wait_type() wt
NATURAL JOIN aurora_stat_wait_event() we
WHERE we.event_name = 'tuple'
    ORDER BY a.wait_time;

  pid  | usename | app_name |                 current_query                  |
 current_wait_type | current_wait_event | current_state | wait_type | wait_event |
 waits | wait_time
-------+---------+----------+------------------------------------------------
+------------------+--------------------+--------------+-----------+------------
+-------+-----------
 32136 | sys     | psql     | /*session3*/ update tab set col=1 where col=1; | Lock
           | tuple              | active        | Lock      | tuple      |     1 |
 1000018
 11999 | sys     | psql     | /*session4*/ update tab set col=1 where col=1; | Lock
           | tuple              | active        | Lock      | tuple      |     1 |
 1000024
```

同時実行数が多いときにそれを減らす

Lock:tupleイベントは、特にワークロードの多い時間帯に、コンスタントに発生する可能性があります。このような状況では、非常にビジーな行への同時実行数を減らすことを検討してください。多くの場合、キューまたはブールロジックを制御する行の数行だけで、これらの行は非常にビジーになります。

ビジネス要件、アプリケーションロジック、およびワークロードタイプに応じて、さまざまなアプローチを使用することで、競合を減らすことができます。例えば、次の操作を実行できます。

- テーブルとデータロジックを再設計し、競合を減らします。
- アプリケーションロジックを変更して、行レベルで高い競合を減らします。
- 行レベルのロックを活用してクエリを再設計します。
- NOWAIT節はリトライオペレーションで使用します。
- 楽観的かつハイブリッドロックロジックの同時実行制御の使用を検討します。
- データベースの隔離レベルの変更を検討してください。

ボトルネックのトラブルシューティング

Lock:tupleは、CPU の枯渇や Amazon EBS 帯域幅の最大使用率などのボトルネックを発生させることがあります。ボトルネックを減らすには、次のアプローチを検討します。

- インスタンスクラスタイプをスケールアップします。
- リソースを大量に消費するクエリを最適化します。
- アプリケーションロジックを変更します。
- ほとんどアクセスされないデータをアーカイブします。

LWLock:buffer_content (BufferContent)

このLWLock:buffer_content待機イベントは、セッションがデータページをメモリに読み取るまたは書き込むために待機中、他のセッションがそのページを書き込み用にロックしている場合に発生します。Aurora PostgreSQL 13 以降では、この待機イベントはBufferContentと呼ばれます。

トピック

サポート対象エンジンバージョン

この待機イベント情報は、Aurora PostgreSQL のすべてのバージョンでサポートされています。

Context

データの読み取りや操作のために、PostgreSQL は共有メモリバッファを介してデータにアクセスします。バッファから読み取るために、プロセスは共有モードでバッファコンテンツに対する軽量ロック (LwLock) を取得します。バッファに書き込むには、排他モードでその ロックを取得します。共有ロックを使用すると、他のプロセスがそのコンテンツの共有ロックを同時に取得できます。排他ロックは、他のプロセスによるいかなるタイプのロック取得も防ぎます。

LWLock:buffer_content(BufferContent) イベントは、複数のプロセスが特定のバッファの内容をロックしようとしていることを示します。

待機時間が増加する原因の可能性

LWLock:buffer_content (BufferContent) イベントが通常より頻繁に発生し、パフォーマンスの問題を示している可能性がある場合、代表的な原因として以下が挙げられます。

同一データに対する同時更新の増加

同じバッファコンテンツを更新するクエリによる同時実行セッションの数が増加する可能性があります。この競合は、インデックスの多いテーブルではより顕著になることがあります。

ワークロードデータがメモリ内に存在しない

アクティブなワークロードが処理しているデータがメモリ上にない場合、これらの待機イベントが増加する可能性があります。この効果は、ロックを保持しているプロセスが、ディスク I/O 操作の実行中にロックを長く維持できるためです。

外部キー制約の過度の使用

外部キー制約により、プロセスがバッファコンテンツロックを保持する時間を増やすことがあります。この効果は、読み取り操作では、そのキーが更新されている間、参照キーに対する共有バッファコンテンツのロックが必要になるためです。

アクション

待機イベントの原因に応じたさまざまなアクションをお勧めします。LWLock:buffer_content(BufferContent)イベントは、Amazon RDS Performance Insights を使用するか、ビューpg_stat_activityのクエリで特定することができます。

トピック

- インメモリ効率の向上
- 外部キー制約の使用を減らす
- 未使用インデックスの削除

インメモリ効率の向上

アクティブなワークロードデータがメモリ内に存在する可能性を高めるには、テーブルをパーティション化するか、インスタンスクラスをスケールアップします。DB インスタンスクラスの詳細については、「Aurora DB インスタンスクラス」を参照してください。

外部キー制約の使用を減らす

外部キー制約の使用で、LWLock:buffer_content(BufferContent)待機イベントが多発しているワークロードを調査します。不要な外部キーの制約を削除します。

未使用インデックスの削除

LWLock:buffer_content (BufferContent) 待機イベントが多いワークロードで、未使用のインデックスを特定して削除します。

LWLock:buffer_mapping

このイベントは、セッションがデータブロックを共有バッファプール内のバッファに関連付けるのを待っているときに発生します。

> **Note**
>
> この待機イベントは、Aurora PostgreSQL バージョン 12 以降ではLWLock:buffer_mapping、バージョン 13 以降ではLWLock:BufferMappingとして発生します。

トピック

- サポート対象エンジンバージョン
- Context
- 原因
- アクション

サポート対象エンジンバージョン

この待機イベント情報は、Aurora PostgreSQL バージョン 9.6 以降に関連します。

Context

共有バッファプールは、プロセスで使用されている、または使用されていたすべてのページを保持する Aurora PostgreSQL メモリ領域です。プロセスがページを必要とするとき、そのページを共有バッファプールに読み取ります。shared_buffersパラメータは、共有バッファサイズを設定し、

テーブルとインデックスページを格納するためのメモリ領域を予約します。このパラメータを変更する場合は、必ずデータベースを再起動してください。詳細については、「共有バッファ」を参照してください。

LWLock:buffer_mapping待機イベントは、以下の場合に発生します。

- プロセスは、バッファテーブルでページを検索し、共有バッファマッピングロックを取得します。
- プロセスは、ページをバッファプールにロードし、排他的バッファマッピングロックを取得します。
- プロセスは、プールからページを削除し、排他バッファーマッピングロックを取得します。

原因

このイベントが通常よりも頻繁に発生する場合、パフォーマンスの問題を示していることがあり、データベースは共有バッファプールにページインとアウトページングを行っています。代表的な原因としては、以下が挙げられます。

- 大きなクエリ
- 肥大化したインデックスとテーブル
- フルテーブルスキャン
- ワーキングセットより小さい共有プールサイズ

アクション

待機イベントの原因に応じたさまざまなアクションをお勧めします。

トピック

- バッファ関連のメトリクスをモニタリングする
- インデックス作成戦略を評価する
- 迅速に確保しなければならないバッファの数を減らす

バッファ関連のメトリクスをモニタリングする

LWLock:buffer_mappingがスパイクを待機したら、バッファヒット率を調べます。これらのメトリクスを使用すると、バッファキャッシュで何が起こっているかをより深く理解できます。次のメトリックを検証します。

BufferCacheHitRatio

この Amazon CloudWatch のメトリクスは、DB クラスター内の DB インスタンスのバッファキャッシュによって処理されるリクエストの割合を測定します。この指標は、LWLock:buffer_mapping待機イベントまでの間に減少することがあります。

blks_hit

この Performance Insights カウンターメトリクスは、共有バッファプールから取得されたブロックの数を示します。LWLock:buffer_mappingの待機イベントが表示された後、blks_hitのスパイクを観察することがあります。

blks_read

この Performance Insights カウンターメトリクスは、共有バッファプールへの読み取りのため、 I/O を必要とするブロックの数を示します。LWLock:buffer_mapping待機イベントまでにblks_readのスパイクが観察されることがあります。

インデックス作成戦略を評価する

インデックス作成戦略がパフォーマンスが低下させていないことを確認するには、次を確認してください。

インデックスの肥大化

インデックスとテーブルの肥大化によって、不要なページが共有バッファに読み込まれないようにします。テーブルに未使用の行がある場合は、データをアーカイブし、テーブルから行を削除することを検討してください。その後、サイズ変更されたテーブルのインデックスを再構築できます。

頻繁に使用するクエリのインデックス

最適なインデックスがあるかどうかを判断するには、Performance Insights で DB エンジンのメトリクスをモニタリングします。tup_returnedメトリクスは、読み込まれた行数を示します。tup_fetchedメトリクスは、クライアントに返される行数を示します。tup_returnedがtup_fetchedを大幅に超える場合、データが適切にインデックスされていない可能性があります。また、テーブルの統計が最新ではない可能性があります。

迅速に確保しなければならないバッファの数を減らす

LWLock:buffer_mapping待機イベントを減らすには、迅速に割り当てる必要があるバッファの数を減らしてください。1 つの戦略として、より小規模なバッチオペレーションを実行します。テーブルをパーティション化することで、より小さなバッチを実現できることがあります。

LWLock:BufferIO (IPC:BufferIO)

LWLock:BufferIOイベントは、Aurora PostgreSQL または RDS for PostgreSQL が同時にページにアクセスしようとしているときに、他のプロセスが入出力 (I/O) オペレーションの完了を待っているときに発生します。その目的は、同じページを共有バッファに読み込むことです。

トピック

- 関連するエンジンのバージョン
- Context
- 原因
- アクション

関連するエンジンのバージョン

この待機イベント情報は、Aurora PostgreSQL のすべてのバージョンに関連しています。Aurora PostgreSQL 12 以前のバージョンでは、この待機イベントは lwlock:buffer_io という名前でしたが、Aurora PostgreSQL 13 バージョンでは lwlock:bufferio という名前です。Aurora PostgreSQL 14 バージョンから、BufferIO 待機イベントは LWLock から IPC 待機イベントタイプ (IPC:BufferIO) に移動されました。

Context

各共有バッファは、ブロック (またはページ) が共有バッファプールの外部で取得される必要があるたびに、LWLock:BufferIO待機イベントに関連付けられた I/O ロックを持ちます。

このロックは、すべての同じブロックへのアクセスを必要とする複数のセッションを処理するために使用されます。このブロックは、shared_buffersパラメータで定義された共有バッファプールの外部から読み取る必要があります。

共有バッファプール内でページが読み込まれると、LWLock:BufferIOロックが解除されます。

> **Note**
>
> LWLock:BufferIO待機イベントはIO:DataFileRead待機イベントに先行します。IO:DataFileRead待機イベントは、データがストレージから読み込まれている間に発生します。

ライトウェイトロックの詳細については、「ロックの概要」を参照してください。

原因

LWLock:BufferIO上位待機中に表示されるイベントの一般的な原因には、次のものがあります。

- 複数のバックエンドまたは接続が I/O オペレーションを保留している同じページにアクセスしようとしている
- 共有バッファプール (shared_buffersパラメータで定義) のサイズと、現在のワークロードが必要とするバッファ数の比率
- 共有バッファプールのサイズが、他の操作によって削除されるページ数とのバランスが悪い
- エンジンが共有バッファプールに必要以上のページを読み込む必要がある大規模なインデックスまたは肥大化したインデックス
- DB エンジンが強制的に必要以上に多くのページをテーブルから読み取るインデックスの欠落
- 同じページで操作を実行しようとするデータベース接続が突然スパイクする

アクション

待機イベントの原因に応じたさまざまなアクションを実行することをお勧めします。

- BufferCacheHitRatioの急減とLWLock:BufferIO待機イベントの相関関係のため、Amazon CloudWatch メトリクスを観察します。この効果は、共有バッファの設定が小さいことを意味することがあります。増やすか、DB インスタンスクラスをスケールアップする必要がある場合があります。ワークロードをより多くのリーダーノードに分割できます。
- LWLock:BufferIOがBufferCacheHitRatioのメトリックと一致する場合は、ワークロードのピーク時間に基づいてmax_wal_sizeとcheckpoint_timeoutをチューニングしてください。次に、原因となっているクエリを特定します。
- 未使用のインデックスがあるかどうかを確認し、それらを削除します。

- パーティション化されたテーブルを使用します (パーティション化されたインデックスもあります)。これにより、インデックスの並べ替えを低く抑え、その影響を軽減することができます。
- 不必要に列のインデックスを作成しないようにします。
- 接続プールを使用して、突然のデータベース接続スパイクを防ぎます。
- ベストプラクティスとして、データベースへの最大接続数を制限します。

LWLock:lock_manager

このイベントは、Aurora PostgreSQL エンジンが、高速パスロックが不可能な場合に共有ロックのメモリ領域を維持し、ロックの割り当て、チェック、および解放を行うときに発生します。

トピック

サポート対象エンジンバージョン

この待機イベント情報は、Aurora PostgreSQL バージョン 9.6 以降に関連します。

Context

SQL ステートメントを発行すると、Aurora PostgreSQL は同時オペレーション中にデータベースの構造、データ、および整合性を保護するためにロックを記録します。エンジンは、高速パスロックまたは高速ではないパスロックを使用して、この目標を達成できます。高速ではないパスロックは、高速パスロックよりも高価で、オーバーヘッドも多く発生します。

高速パスロック

バックエンドプロセスでは、頻繁にロックされロック解除されるがめったに競合しないロックのオーバーヘッドを減らすために、高速パスロックを使用できます。データベースでは、以下の基準を満たすロックにこのメカニズムを使用します。

- DEFAULT ロック方式を使用します。
- これらは、共有関係ではなく、データベースリレーションに対するロックを表します。

- これらは競合する可能性が低い弱いロックです。
- エンジンは、競合するロックが存在する可能性がないことをすばやく確認できます。

エンジンは、以下のいずれかの条件が true の場合、高速パスロックを使用できません。

- ロックが上記の条件を満たしていません。
- バックエンドプロセスに使用できるスロットはこれ以上ありません。

高速パスロックの詳細については、「PostgreSQL ロックマネージャ README」 のfast pathおよび「PostgreSQL ドキュメント」のpg-locksを参照してください。

ロックマネージャのスケーリング問題の例

この例では、purchasesという名前のテーブルが 5 年分のデータを日ごとにパーティション化して保存しています。各パーティションには 2 つのインデックスがあります。次の一連のイベントが発生します。

1. 何日分ものデータをクエリすると、データベースは多くのパーティションを読み取る必要があります。
2. データベースは、各パーティションに対してロックエントリを作成します。パーティションインデックスがオプティマイザアクセスパスの一部である場合、データベースはそれらのロックエントリも作成します。
3. 同じバックエンドプロセスでリクエストされたロックエントリの数がFP_LOCK_SLOTS_PER_BACKENDの値である16より大きい場合、ロックマネージャは非高速パスロック方式を使用します。

最新のアプリケーションには何百ものセッションがあることがあります。同時セッションが適切なパーティションプルーニングを行わずに親を照会している場合、データベースは数百または数千の非高速パスロックを作成することがあります。通常、この同時実行が vCPUs の数よりも多いと、LWLock:lock_manager待機イベントが表示されます。

> **Note**
>
> LWLock:lock_manager待機イベントは、データベーススキーマのパーティションまたはインデックスの数とは関係ありません。代わりに、データベースが制御する必要がある非高速パスロックの数と関係しています。

待機時間が増加する原因の可能性

LWLock:lock_manager待機イベントが通常より頻繁に発生する場合は、おそらくパフォーマンスの問題を示しており、突然のスパイクが発生する原因として最も可能性が高いものは次のとおりです。

- 同時アクティブセッションは、高速パスロックを使用しないクエリを実行しています。また、これらのセッション数が最大 vCPU を超えています。
- 多数の同時アクティブセッションが、大きくパーティション化されたテーブルにアクセスしています。各パーティションには複数のインデックスがあります。
- データベースで接続ストームが発生しています。デフォルトでは、一部のアプリケーションと接続プールソフトウェアは、データベースが遅い場合により多くの接続を作成します。これは問題を悪化させます。接続ストームが発生しないように接続プールソフトウェアをチューニングします。
- 多数のセッションが、パーティションをプルーニングせずに親テーブルをクエリします。
- データ定義言語 (DDL)、データ操作言語 (DML)、またはメンテナンスコマンドは、頻繁にアクセスまたは変更されるビジーリレーションあるいはタプルのいずれかを排他的にロックします。

アクション

待機イベントの原因に応じたさまざまなアクションをお勧めします。

トピック

- パーティションプルーニングを使用する
- 不要なインデックスを削除する
- 高速パスロック用にクエリをチューニングする
- 他の待機イベントに合わせてチューニングする
- ハードウェアのボトルネックを減らす
- 接続プーラーを使用する
- Aurora PostgreSQL のバージョンをアップグレードする

パーティションプルーニングを使用する

パーティションプルーニングとは、不要なパーティションをテーブルスキャンから除外し、パフォーマンスを向上させるクエリ最適化戦略です。パーティションプルーニングは、デフォルトで有効になっています。オフになっている場合は、次のようにオンにします。

```
SET enable_partition_pruning = on;
```

クエリのWHERE節にパーティショニングに使用される列が含まれる場合は、パーティションのプルーニングを利用できます。詳細については、PostgreSQL のドキュメントの「パーティションプルーニング」を参照してください。を参照してください。

不要なインデックスを削除する

データベースには、未使用またはほとんど使用されないインデックスが含まれている可能性があります。その場合は、それらの削除を検討します。次のいずれかを実行します。

- 不要なインデックスを見つける方法については、PostgreSQL wikiの「未使用のインデックス」を参照してください。
- PG コレクタを実行します。この SQL スクリプトは、データベース情報を収集し、統合 HTML レポートに表示します。「未使用のインデックス」セクションをチェックします。詳細については、AWSLabs GitHub リポジトリの「PG コレクター」を参照してください。

高速パスロック用にクエリをチューニングする

クエリで高速パスロックが使用されているかどうかを調べるには、pg_locksテーブルのfastpath列にクエリを実行します。クエリで高速パスロックを使用していない場合は、クエリあたりのリレーション数を 16 未満に減らしてください。

他の待機イベントに合わせてチューニングする

LWLock:lock_managerが上位待機のリストの第1位または 2 番目である場合、次の待機イベントもリストに表示されるかどうかを確認します。

- Lock:Relation
- Lock:transactionid
- Lock:tuple

上記のイベントがリスト内で上位に表示される場合は、まずこれらの待機イベントのチューニングを検討してください。これらのイベントは、LWLock:lock_managerのドライバーになり得ます。

ハードウェアのボトルネックを減らす

CPU の枯渇や Amazon EBS 帯域幅の最大使用率など、ハードウェアのボトルネックが発生する可能性があります。このような場合は、ハードウェアのボトルネックを減らすことを検討してください。以下のアクションの場合を検討します。

- インスタンスクラスをスケールアップします。
- 大量の CPU とメモリを消費するクエリを最適化します。
- アプリケーションロジックを変更します。
- データをアーカイブします。

CPU、メモリ、および EBS ネットワーク帯域幅の詳細については、Amazon RDS インスタンスタイプを参照してください。

接続プーラーを使用する

アクティブな接続の総数が最大 vCPU を超えると、インスタンスタイプがサポートできるより多くの OS プロセスが CPU を必要とします。このような場合は、接続プールの使用またはチューニングを検討してください。インスタンスタイプの vCPUs の詳細については、「Amazon RDS インスタンスタイプ」を参照してください。

接続プールの詳細については、次のリソースを参照してください。

- Amazon RDS Proxy for Aurora の使用
- pgbouncer
- PostgreSQL ドキュメントの接続プールとデータソース

Aurora PostgreSQL のバージョンをアップグレードする

現在使用している Aurora PostgreSQL のバージョンが 12 より前のものであれば、バージョン 12 以上にアップグレードします。PostgreSQL バージョン 12 と 13 では、パーティションメカニズムが改良されています。バージョン12の詳細については、PostgreSQL 12.0 リリースノートを参照してください。Aurora PostgreSQL のアップグレードの詳細については、「Amazon Aurora PostgreSQL の更新」を参照してください。

LWLock:MultiXact

LWLock:MultiXactMemberBuffer、LWLock:MultiXactOffsetBuffer、LWLock:MultiXactMembe および LWLock:MultiXactOffsetSLRU の各待機イベントは、特定のテーブル内の同じ行を修正するトランザクションのリストをセッションが取得するのを待っていることを示します。

- LWLock:MultiXactMemberBuffer – プロセスは、multixact メンバーのシンプルな最も長い時間使われていない (SLRU) バッファの I/O を待っています。
- LWLock:MultiXactMemberSLRU – プロセスは、multixact メンバーのシンプルな最も長い時間使われていない (SLRU) キャッシュへのアクセスを待っています。
- LWLock:MultiXactOffsetBuffer – プロセスは、multixact オフセットのシンプルな最も長い時間使われていない (SLRU) バッファの I/O を待っています。
- LWLock:MultiXactOffsetSLRU – プロセスは、multixact オフセットのシンプルな最も長い時間使われていない (SLRU) キャッシュへのアクセスを待っています。

トピック

サポート対象エンジンバージョン

この待機イベント情報は、Aurora PostgreSQL のすべてのバージョンでサポートされています。

Context

multixact は、同じテーブル行を修正するトランザクション ID (XID) のリストを格納するデータ構造です。1 つのトランザクションでテーブル内の行が参照されると、トランザクション ID がテーブルヘッダー行に格納されます。複数のトランザクションでテーブル内の同じ行が参照されると、トランザクション ID のリストは multixact データ構造に格納されます。multixact 待機イベントは、テーブル内の特定の行を参照するトランザクションのリストをセッションがデータ構造から取得していることを示します。

待ち時間増加の考えられる原因

multixact が必要になる場合の一般的な原因は次のとおりです。

- 明示的なセーブポイントからのサブトランザクション - トランザクションにセーブポイントを明示的に作成すると、同じ行に新しいトランザクションが生成されます。例えば、SELECT FOR UPDATE、SAVEPOINT、UPDATE の順に使用した場合です。

  一部のドライバー、オブジェクトリレーショナルマッパー (ORM)、および抽象化レイヤーには、すべての操作をセーブポイントとともに自動的にラップするための設定オプションがあります。これにより、一部のワークロードで多数の multixact 待機イベントが生成される可能性があります。PostgreSQL JDBC ドライバーの autosave オプションはその一例です。詳細については、PostgreSQL JDBC ドキュメントの「pgJDBC」を参照してください。もう一つの例は、PostgreSQL ODBC ドライバーとその protocol オプションです。詳細については、PostgreSQL ODBC ドライバードキュメントの「psqlODBC Configuration Options」を参照してください。
- PL/pgSQL EXCEPTION 句からのサブトランザクション – PL/pgSQL 関数またはプロシージャに書き込む各 EXCEPTION 句は、内部で SAVEPOINT を作成します。
- 外部キー – 複数のトランザクションが親の行で共有ロックを取得する場合。

特定の行が複数のトランザクションオペレーションに含まれる場合、その行を処理するには、multixact リストからトランザクション ID を取得する必要があります。ルックアップによりメモリキャッシュから multixact を取得できない場合は、データ構造を Aurora ストレージレイヤーから読み取る必要があります。このストレージからの I/O により、SQL クエリに時間がかかることがあります。メモリキャッシュミスは、複数のトランザクションの数が多く、使用率が高くなると発生し始める可能性があります。これらのすべての要因が、この待機イベントの増加の原因となっています。

アクション

待機イベントの原因に応じて、異なるアクションをお勧めします。これらのアクションの一部は、待機イベントを即座に削減するのに役立ちます。ただし、アクションによっては、ワークロードをスケールするために調査と修正が必要な場合があります。

トピック

- この待機イベントがあるテーブルでバキュームフリーズを実行する
- この待機イベントがあるテーブルで autovacuum の頻度を上げる
- メモリパラメータを増加する
- 実行時間が長いトランザクションを削減する
- 長時間のアクション

この待機イベントがあるテーブルでバキュームフリーズを実行する

この待機イベントが突然急増して本番環境に影響する場合は、次のいずれかの一時的な方法を使用してその数を減らすことができます。

- 影響を受けるテーブルまたはテーブルパーティションで VACUUM FREEZE を使用して、問題を即座に解決します。詳細については、「VACUUM」を参照してください。
- VACUUM (FREEZE、INDEX_CLEANUP FALSE) 句を使用すると、インデックスをスキップしてクイックバキュームを実行できます。詳細については、「テーブルをできるだけ早くバキューム処理する」を参照してください。

この待機イベントがあるテーブルで autovacuum の頻度を上げる

すべてのデータベースのすべてのテーブルをスキャンした後、VACUUM は最終的に multixact を削除し、最も古い multixact 値が前に送られます。詳細については、「Multixacts と Wraparound」を参照してください。LWLock:MultiXact 待機イベントを最小限に抑えるには、必要な頻度で VACUUM を実行する必要があります。そのためには、Aurora PostgreSQL DB クラスターの VACUUM が最適に設定されていることを確認します。

影響を受けたテーブルまたはテーブルパーティションで VACUUM FREEZE を使用して待機イベントの問題を解決する場合は、インスタンスレベルで autovacuum を調整する代わりに、pg_cron などのスケジューラを使用して VACUUM を実行することをお勧めします。

autovacuum をより頻繁に実行するため、影響を受けるテーブルの `autovacuum_multixact_freeze_max_age` ストレージパラメータの値を減らすことができます。詳細については、「autovacuum_multixact_freeze_max_age」を参照してください。

メモリパラメータを増加する

以下のパラメータをクラスターレベルで設定することで、クラスター内のすべてのインスタンスの整合性を維持できます。これにより、ワークロードの待機イベントを減らすことができます。メモリを不足させないように、これらの値をあまり高く設定しないことをお勧めします。

- `multixact_offsets_cache_size ~ 128`
- `multixact_members_cache_size ~ 256`

パラエータのインスタンスを再起動して変更を有効にします。これらのパラメータを使うと、ディスクに書き込むことなく、より多くのインスタンス RAM を使用して multixact 構造をメモリに保存できます。

実行時間が長いトランザクションを削減する

実行時間が長いトランザクションでは、トランザクションがコミットされるか、読み取り専用トランザクションが閉じられるまで、バキュームにその情報が保持されます。長時間実行トランザクションを積極的にモニタリングし、管理することをお勧めします。詳細については、「データベースがトランザクション接続で長時間アイドル状態になっている」を参照してください。実行の長いトランザクションの使用を避けるか、最小限に抑えるようにアプリケーションを変更してみてください。

長時間のアクション

ワークロードを調べて、multixact スピルオーバーの原因を見つけます。ワークロードをスケーリングして待機イベントを減らすには、この問題を修正する必要があります。

- テーブルの作成に使用する DDL (データ定義言語) を分析する必要があります。テーブル構造とインデックスが適切に設計されていることを確認します。
- 影響を受けるテーブルに外部キーがある場合、それらが必要かどうか、または参照整合性を強制する別の方法があるかどうかを確認します。
- テーブルに未使用のインデックスが大量にある場合、autovacuum がワークロードに適合せず、実行がブロックされる可能性があります。これを回避するには、未使用のインデックスをチェックし、それらを完全に削除します。詳細については、「大量のインデックスでの autovacuum の管理」を参照してください。
- トランザクションでのセーブポイントの使用を減らします。

Timeout:PgSleep

`Timeout:PgSleep`イベントは、サーバプロセスが`pg_sleep`関数を呼び出し、スリープタイムアウトの期限切れを待っているときに発生します。

トピック

- サポート対象エンジンバージョン
- 待機時間が増加する原因の可能性
- アクション

サポート対象エンジンバージョン

この待機イベント情報は、Aurora PostgreSQL のすべてのバージョンでサポートされています。

待機時間が増加する原因の可能性

この待機イベントは、アプリケーション、ストアド関数、またはユーザーが次のいずれかの関数を呼び出す SQL ステートメントを発行したときに発生します。

- `pg_sleep`
- `pg_sleep_for`
- `pg_sleep_until`

前述の関数は、指定された秒数が経過するまで実行を遅らせます。例えば、`SELECT pg_sleep(1)` は 1 秒間一時停止します。詳細については、PostgreSQL のドキュメントの「実行の遅延」を参照してください。

アクション

`pg_sleep`関数を実行していたステートメントを特定します。関数の使用が適切かどうかを判断します。

Amazon DevOps Guru のプロアクティブインサイトによる Aurora PostgreSQL のチューニング

DevOps Guru のプロアクティブインサイトは、問題の原因となる可能性がある Aurora PostgreSQL DB クラスターの条件を検出して、問題が発生する前に通知します。DevOps Guru では、次のことができます。

- データベース構成を一般的な推奨設定と照合することで、データベースに関する多くの一般的な問題を防ぎます。
- 未チェックのままにしておくと、後で大きな問題につながる可能性があるフリート内の重大な問題について警告します。
- 新しく発見された問題について警告します。

すべてのプロアクティブインサイトには、問題の原因の分析と是正措置の推奨事項が含まれています。

トピック

- データベースがトランザクション接続で長時間アイドル状態になっている

データベースがトランザクション接続で長時間アイドル状態になっている

データベースへの接続が 1800 秒以上 `idle in transaction` 状態です。

トピック

サポート対象エンジンバージョン

このインサイト情報は、Aurora PostgreSQL のすべてのバージョンでサポートされています。

Context

`idle in transaction` 状態のトランザクションがロックを保持していて、他のクエリをブロックしている可能性があります。また、VACUUM (自動バキュームを含む) がデッド行をクリーンアップするのを妨げて、インデックスやテーブルが肥大化したり、トランザクション ID がラップアラウンドしたりします。

この問題の考えられる原因

インタラクティブセッションで BEGIN または START TRANSACTION を使用して開始されたトランザクションが、COMMIT、ROLLBACK、または END コマンドを使用しても終了していません。これにより、トランザクションは `idle in transaction` 状態に移行します。

アクション

`pg_stat_activity` クエリを実行すると、アイドル状態のトランザクションを見つけることができます。

SQL クライアントで、次のクエリを実行して、`idle in transaction` 状態にあるすべての接続を一覧表示し、継続時間順に並べ替えます。

```
SELECT now() - state_change as idle_in_transaction_duration, now() - xact_start as
 xact_duration,*
FROM  pg_stat_activity
```

```
WHERE state  = 'idle in transaction'
AND   xact_start is not null
ORDER BY 1 DESC;
```

インサイトの原因に応じて、異なるアクションをお勧めします。

トピック

- 接続を終了する
- 接続を作成する
- idle_in_session_timeout パラメータを設定する
- AUTOCOMMIT のステータスを確認する
- アプリケーションコード内のトランザクションロジックを確認する

接続を終了する

インタラクティブセッションで BEGIN または START TRANSACTION を使用してトランザクションを開始すると、トランザクションは `idle in transaction` 状態に移行します。COMMIT、ROLLBACK、END コマンドを実行してトランザクションを終了するか、接続を完全に切断してトランザクションをロールバックするまで、この状態のままになります。

接続を作成する

次のクエリを使用して、アイドル状態のトランザクションがある接続を終了します。

```
SELECT pg_terminate_backend(pid);
```

pid は接続のプロセス ID です。

idle_in_session_timeout パラメータを設定する

パラメータグループの `idle_in_transaction_session_timeout` パラメータを設定します。このパラメータを設定する利点は、手動操作を行わなくても、長時間アイドル状態になっているトランザクションを終了できることです。このパラメータの詳細については、「PostgreSQL documentation」(PostgreSQL ドキュメント) を参照してください。

接続が終了し、指定した時間を超えてトランザクションが idle_in_transaction 状態にあると、PostgreSQL ログファイルに次のメッセージが報告されます。

```
FATAL: terminating connection due to idle in transaction timeout
```

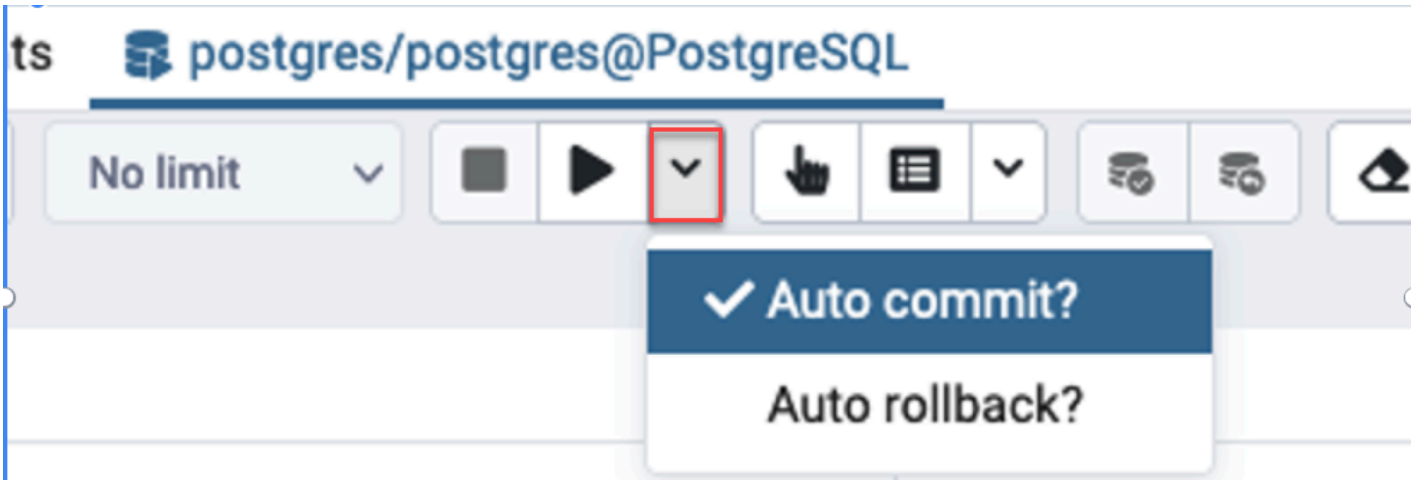
AUTOCOMMIT のステータスを確認する

AUTOCOMMIT は、デフォルトで有効になっています。ただし、クライアントで誤ってオフにした場合は、必ずオンに戻してください。

- psql クライアントで次のコマンドを実行します。

```
postgres=> \set AUTOCOMMIT on
```

- pgadmin で、下矢印から AUTOCOMMIT オプションを選択してオンにします。



アプリケーションコード内のトランザクションロジックを確認する

アプリケーションロジックを調べ手、問題がない確認します。以下のアクションの場合を検討します。

- アプリケーションで JDBC auto commit が true に設定されているかどうかを確認します。また、コード内で明示的な `COMMIT` コマンドを使用することも検討してください。
- エラー処理ロジックをチェックして、エラー後にトランザクションがクローズされるかどうかを確認します。
- トランザクションが開いているときに、アプリケーションがクエリによって返された行の処理に時間がかかるかどうかを確認します。その場合は、行を処理する前にトランザクションを閉じるようにアプリケーションをコーディングすることを検討してください。
- トランザクションに長時間実行される操作が多数含まれていないか確認します。その場合は、1 つのトランザクションを複数のトランザクションに分割します。

関連するメトリクス

以下の PI メトリクスがこのインサイトに関連しています。

- idle_in_transaction_count - `idle in transaction` 状態にあるセッション数。
- idle_in_transaction_max_time - `idle in transaction` 状態で最も長く実行されているトランザクションの継続時間。

Amazon Aurora PostgreSQL を使用する際のベストプラクティス

Amazon Aurora PostgreSQL DB クラスターを管理するためのベストプラクティス例を以下に示します。基本的なメンテナンスタスクも確認してください。詳細については、「Amazon Aurora PostgreSQL の管理」を参照してください。

トピック

- Aurora PostgreSQL DB インスタンスのパフォーマンスの低下、自動再起動、フェイルオーバーの回避
- 診断テーブルとインデックスの肥大化
- Aurora PostgreSQL のメモリ管理が改善されました
- Amazon Aurora PostgreSQL による高速フェイルオーバー
- Aurora PostgreSQL のクラスターキャッシュ管理によるフェイルオーバー後の高速リカバリ
- 管理する Aurora PostgreSQL 接続チャーンとプーリング
- Aurora PostgreSQL のメモリパラメータの調整
- Amazon CloudWatch メトリクスを使用して Aurora PostgreSQL のリソース使用状況を分析する
- 論理レプリケーションを使用して Aurora PostgreSQL のメジャーバージョンアップグレードを実行する
- ストレージ問題のトラブルシューティング

Aurora PostgreSQL DB インスタンスのパフォーマンスの低下、自動再起動、フェイルオーバーの回避

負荷の高いワークロードや、DB インスタンスに割り当てられたリソースを超えて急増するワークロードを実行している場合、アプリケーションと Aurora データベースを実行しているリソースを使い果たしてしまう可能性があります。CPU 使用率、メモリ使用量、使用されているデータベース接

続数など、データベースインスタンスに関するメトリクスを取得するには、Amazon CloudWatch、パフォーマンスインサイト、および拡張モニタリングが提供するメトリクスを参照できます。DB インスタンスのモニタリングについては、「Amazon Aurora クラスターでのメトリクスのモニタリング」を参照してください。

ワークロードが使用しているリソースを使い果たした場合、DB インスタンスは遅くなったり、再起動したり、別の DB インスタンスにフェイルオーバーしたりする可能性があります。これを避けるには、リソースの使用状況を監視し、DB インスタンスで実行されているワークロードを調べ、必要に応じて最適化を行います。最適化を行ってもインスタンスのメトリクスが改善されず、リソースの枯渇も緩和されない場合は、上限に達する前に DB インスタンスをスケールアップすることを検討してください。利用可能な DB インスタンスクラスとその仕様の詳細については、「Aurora DB インスタンスクラス」を参照してください。

診断テーブルとインデックスの肥大化

PostgreSQL のマルチバージョン同時実行制御 (MVCC) を使用すると、データの整合性を維持できます。PostgreSQL MVCC は、トランザクションがコミットまたはロールバックされるまで、更新または削除された行 (タプルとも呼ばれます) の内部コピーを保存することによって機能します。この保存された内部コピーはユーザーからは見えません。ただし、これらの非表示のコピーが VACUUM または AUTOVACUUM ユーティリティによって定期的にクリーンアップされない場合、テーブルが肥大化する可能性があります。テーブルの肥大化をチェックしないと、ストレージコストの増加や処理速度の低下につながる可能性があります。

多くの場合、Aurora の VACUUM または AUTOVACUUM のデフォルト設定は、不要なテーブルの肥大化を処理するのに十分なものです。ただし、次のような状況がアプリケーションで発生している場合は、肥大化がないかどうかを確認することをお勧めします。

- VACUUM プロセスの間で、比較的短時間で多数のトランザクションを処理している。
- パフォーマンスが低下し、ストレージが不足している。

はじめに、dead タプルでどの程度のスペースを使用しているか、テーブルとインデックスの肥大化をクリーンアップすることでどれだけ回復が見込めるか、最大限正確な情報を収集してください。そのためには、pgstattuple 拡張機能を使用して、Aurora クラスターの統計情報を収集します。詳細については、「pgstattuple」を参照してください。pgstattuple 拡張機能を使用する権限は、pg_stat_scan_tables ロールとデータベースのスーパーユーザーに限定されます。

Aurora で pgstattuple 拡張機能を作成するには、クライアントセッション (psql や pgAdmin など) をクラスターに接続し、次のコマンドを使用します。

```
CREATE EXTENSION pgstattuple;
```

プロファイルする各データベースで、拡張機能を作成します。拡張機能を作成したら、コマンドラインインターフェイス (CLI) を使用して、使用できないスペースをどの程度再利用できるかを測定します。統計を収集する前に、AUTOVACUUM を 0 に設定してクラスターパラメータグループを変更します。0 に設定すると、アプリケーションによって残されたデットタプルを Aurora が自動的にクリーンアップすることを防止しますが、結果の精度に影響を与える可能性があります。次のコマンドを入力して、単純なテーブルを作成します。

```
postgres=> CREATE TABLE lab AS SELECT generate_series (0,100000);
SELECT 100001
```

次の例では、DB クラスターの AUTOVACUUM をオンにしてクエリを実行しています。dead_tuple_count は 0 で、これは AUTOVACUUM が PostgreSQL データベースから古いデータやタプルを削除したことを示します。

pgstattuple を使用してテーブルに関する情報を収集するには、クエリにテーブルの名前またはオブジェクト識別子 (OID) を指定します。

```
postgres=> SELECT * FROM pgstattuple('lab');
```

```
table_len  | tuple_count | tuple_len | tuple_percent | dead_tuple_count |
 dead_tuple_len | dead_tuple_percent | free_space | free_percent
-----------+-------------+-----------+---------------+------------------
+----------------+--------------------+------------+--------------
3629056    | 100001      | 2800028   | 77.16         | 0                | 0
  | 0                  | 16616      | 0.46
(1 row)
```

次のクエリでは、AUTOVACUUM をオフにして、テーブルから 25,000 行を削除するコマンドを入力しています。その結果、dead_tuple_count は 25000 に増加しました。

```
postgres=> DELETE FROM lab WHERE generate_series < 25000;
```

```
DELETE 25000
```

```
SELECT * FROM pgstattuple('lab');
```

```
table_len | tuple_count | tuple_len | tuple_percent | dead_tuple_count | dead_tuple_len
 | dead_tuple_percent | free_space | free_percent
-----------+-------------+-----------+---------------+------------------
+----------------+--------------------+------------+--------------
3629056 | 75001 | 2100028 | 57.87 | 25000 | 700000 | 19.29 | 16616 | 0.46
(1 row)
```

これらのデットタプルを再利用するには、VACUUM プロセスを開始します。

アプリケーションを中断せずに肥大化をモニタリングする

Aurora クラスターの設定は、ほとんどのワークロードでベストプラクティスが実現できるように最適化されています。ただし、アプリケーションや使用パターンに合わせてクラスターを最適化する場合もあります。この場合、ビジー状態のアプリケーションを中断することなく pgstattuple 拡張機能を使用できます。そのためには、次のステップを実行します。

1. Aurora インスタンスのクローンを作成します。
2. パラメータファイルを変更して、クローンの AUTOVACUUM をオフにします。
3. サンプルワークロードまたは pgbench (PostgreSQL でベンチマークテストを実行するためのプログラム) を使用してクローンをテストしながら pgstattuple クエリを実行します。詳細については、「pgbench」を参照してください。

アプリケーションを実行して結果を確認したら、復元したコピーに pg_repack または VACUUM FULL を使用して違いを比較します。dead_tuple_count、dead_tuple_len、dead_tuple_percent のいずれかが大幅に減少している場合は、本稼働クラスターのバキュームスケジュールを調整して肥大化を最小限に抑えています。

テンポラリテーブルでの肥大化の回避

アプリケーションでテンポラリテーブルを作成する場合、そのテンポラリテーブルが不要になった場合はアプリケーションから削除してください。自動バキュームプロセスでは、テンポラリテーブルを検索しません。テンポラリテーブルをそのままにしておくと、データベースがすぐに肥大化してしまう可能性があります。さらに、この肥大化はシステムテーブルにまで及ぶ可能性があります。システムテーブルは、pg_attribute や pg_depend などの PostgreSQL オブジェクトや属性を追跡する内部テーブルです。

テンポラリテーブルが不要になったら、TRUNCATE ステートメントを使用してテーブルを空にして、スペースを空けることができます。次に、pg_attribute テーブルと pg_depend テーブルのバキューム処理を手動で実行します。これらのテーブルをバキューム処理することで、テンポラリテーブルを継続的に作成、切り捨て、削除しても、タプルが増えることでシステムが肥大化することがなくなります。

次の構文を追加して、コンテンツがコミットされた際に新しい行を削除することで、テンポラリテーブルを作成する際にこの問題を回避できます。

```
CREATE TEMP TABLE IF NOT EXISTS table_name(table_description) ON COMMIT DELETE ROWS;
```

ON COMMIT DELETE ROWS 句は、トランザクションがコミットされたときにテンポラリテーブルを切り捨てます。

インデックスの肥大化の回避

テーブルのインデックス付きフィールドを変更した場合、インデックスを更新すると、そのインデックスに 1 つまたは複数のデットタプルが生成されます。デフォルトでは、自動バキューム処理によってインデックスの肥大化をクリーンアップしますが、このクリーンアップにはかなりの時間とリソースが必要です。テーブルの作成時にインデックスクリーンアップ設定を指定するには、vacuum_index_cleanup 句を含めてください。デフォルトでは、テーブルの作成時にこの句は AUTO に設定されます。つまり、サーバーは、テーブルのバキューム処理時にインデックスのクリーンアップが必要かどうかを判断します。句を ON に設定すると、特定のテーブルのインデックスクリーンアップを有効になります。OFF に設定すると、そのテーブルのインデックスクリーンアップを無効にできます。インデックスのクリーンアップをオフにすると時間を節約できる可能性がありますが、インデックスが肥大化する可能性があることに注意してください。

コマンドラインでテーブルのバキューム処理を行うと、インデックスのクリーンアップを手動で制御できます。テーブルをバキュームしてデッドタプルをインデックスから削除するには、INDEX_CLEANUP 句を ON の値とテーブル名で記述します。

```
acctg=> VACUUM (INDEX_CLEANUP ON) receivables;

INFO: aggressively vacuuming "public.receivables"
VACUUM
```

インデックスを消去せずにテーブルをバキューム処理するには、OFF の値を指定します。

```
acctg=> VACUUM (INDEX_CLEANUP OFF) receivables;

INFO: aggressively vacuuming "public.receivables"
VACUUM
```

Aurora PostgreSQL のメモリ管理が改善されました

お客様のワークロードが DB インスタンスで使用可能な空きメモリを使い果たすと、オペレーティングシステムによってデータベースが再起動され、データベースが使用できなくなります。Aurora PostgreSQL では、メモリ管理機能が向上し、空きメモリ不足による安定性の問題やデータベースの再起動を未然に防ぐことができます。この改善は、以下のバージョンでデフォルトで利用できます。

- 15.3 以降の 15 バージョン
- 14.8 以降の 14 バージョン
- 13.11 以降の 13 バージョン
- 12.15 以降の 12 バージョン
- 11.20 以降の 11 バージョン

メモリ管理を改善するために、次の処理を行います。

- システムが重大なメモリ負荷に近づいているときに、より多くのメモリを要求するデータベーストランザクションをキャンセルします。
- すべての物理メモリを使い果たし、スワップを使い果たしそうになったとき、システムは重大なメモリ負荷にさらされていると言われます。このような状況では、DB インスタンスのメモリ負荷を直ちに軽減するために、メモリを要求するトランザクションはすべてキャンセルされます。
- 重要な PostgreSQL ランチャーと自動バキュームワーカーなどのバックグラウンドワーカーは常に保護されています。

メモリ管理パラメータの設定

メモリ管理を有効にするには

この機能は、デフォルトでオンになっています。次の例に示すように、メモリ不足のためにトランザクションがキャンセルされると、エラーメッセージが表示されます。

```
ERROR: out of memory Detail: Failed on request of size 16777216.
```

メモリ管理を無効にするには

この機能を無効にするには、psql を使用して Aurora PostgreSQL DB クラスターに接続し、以下で説明するようにパラメータ値に SET ステートメントを使用します。

Aurora PostgreSQL バージョン 11.21、12.16、13.12、14.9、15.4 以前のバージョンの場合：

```
postgres=>SET rds.memory_allocation_guard = true;
```

パラメータグループでは、`rds.memory_allocation_guard`パラメータのデフォルト値は `false` に設定されます。

Aurora PostgreSQL 12.17、13.13、14.10、15.5 以降のバージョンの場合：

```
postgres=>rds.enable_memory_management = false;
```

パラメータグループでは、`rds.enable_memory_management`パラメータのデフォルト値は `true` に設定されます。

DB クラスターパラメータグループでこれらのパラメータの値を設定すると、クエリはキャンセルされません。DB クラスターパラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。

これらの動的パラメータの値をセッションレベルで設定して、メモリ管理を改善するためにセッションを含めたり除外したりすることもできます。

> **Note**
>
> この機能をオフにするとメモリ不足エラーが発生し、システムのメモリ不足が原因でワークロードによってデータベースが再起動する可能性があるため、この機能をオフにすることはお勧めしません。

Amazon Aurora PostgreSQL による高速フェイルオーバー

以下では、フェイルオーバーが可能な限り速く実行する方法について説明します。フェイルオーバー後に迅速に回復するには、Aurora PostgreSQL DB クラスターのクラスターキャッシュ管理を使用できます。詳細については、「Aurora PostgreSQL のクラスターキャッシュ管理によるフェイルオーバー後の高速リカバリ」を参照してください。

フェイルオーバーを高速に実行するための手順には、次のようなものがあります。

- Transmission Control Protocol (TCP) のキープアライブを短い時間に設定することで、障害発生時、読み取りタイムアウトになる前に長時間実行中のクエリを停止します。
- Java ドメインネームシステム (DNS) キャッシュのタイムアウトを積極的に設定します。これにより、Aurora の読み取り専用エンドポイントでは、その後の接続試行で読み取り専用ノードを適切に循環させることができるようになります。
- JDBC 接続文字列で使用されるタイムアウト可変をできるだけ低く設定します。短期間の実行クエリと長期間の実行クエリに対して、別々の接続オブジェクトを使用します。
- 指定された Aurora エンドポイントの読み書きを使用してクラスターに接続します。
- RDS API オペレーションを使用して、サーバー側の障害に対するアプリケーションの応答をテストします。また、パケットドロップツールを使用して、クライアント側の障害に対するテストアプリケーションの応答をテストします。
- AWS JDBC ドライバーを使用して、Aurora PostgreSQL のフェイルオーバー機能を最大限に活用します。AWS JDBC ドライバーおよびその使用方法の詳細については、「Amazon Web Services (AWS) JDBC ドライバー GitHub リポジトリ」を参照してください。

これらについて、以下で詳しく説明します。

トピック

- TCP キープアライブパラメータの設定
- 高速フェイルオーバー用にアプリケーションを設定する
- フェイルオーバーのテスト
- 高速フェイルオーバーの例 (Java)

TCP キープアライブパラメータの設定

TCP 接続を設定すると、接続には一連のタイマーが関連付けられます。キープアライブタイマーがゼロになると、エンドポイントにキープアライブプローブパケットが送信されます。プローブが応答を受信した場合、接続は引き続き稼働中であると推測できます。

TCP キープアライブパラメータを有効化して積極的に設定することで、クライアントがデータベースに接続できない場合は、有効な接続が速やかに切断されます。その後、アプリケーションは新しいエンドポイントに接続できます。

以下の TCP キープアライブパラメータを設定する必要があります。

- `tcp_keepalive_time` は、ソケットからデータが送信されない場合に、キープアライブパケットが送信される時間を秒単位で制御します。ACK はデータとはみなされません。次の設定が推奨されます。

  ```
  tcp_keepalive_time = 1
  ```
- `tcp_keepalive_intvl` は、最初のパケットが送信されてから後続のパケットが送信されるまでの時間を秒単位で制御します。この時間は、`tcp_keepalive_time` パラメータで設定します。次の設定が推奨されます。

  ```
  tcp_keepalive_intvl = 1
  ```
- `tcp_keepalive_probes` は、アプリケーションに通知される前に発生する、認知されていないキープアライブプローブ数です。次の設定が推奨されます。

  ```
  tcp_keepalive_probes = 5
  ```

この設定で、データベースの応答が停止してから 5 秒後以内にアプリケーションに通知します。アプリケーションネットワークでキープアライブがドロップする頻度が高い場合には、`tcp_keepalive_probes` 値をより高く設定できます。これにより実際の障害を検出するまでの時間が増えますが、信頼性が低いネットワークでもバッファを確保できます。

Linux で TCP キープアライブパラメータを設定するには

1. TCP キープアライブパラメータの設定方法をテストします。

   そのためには、以下のコマンドをコマンドラインで使用することをお勧めします。この推奨設定はシステム全体に適用されます。つまり、`SO_KEEPALIVE` オプションをオンにしてソケットを作成する他のすべてのアプリケーションにも影響します。

```
sudo sysctl net.ipv4.tcp_keepalive_time=1
sudo sysctl net.ipv4.tcp_keepalive_intvl=1
sudo sysctl net.ipv4.tcp_keepalive_probes=5
```

2. アプリケーションに適した設定を見つけたら、以下の行を /etc/sysctl.conf に追加して、変更した内容を含めてこの設定を維持します。

```
tcp_keepalive_time = 1
tcp_keepalive_intvl = 1
tcp_keepalive_probes = 5
```

高速フェイルオーバー用にアプリケーションを設定する

以下では、Aurora PostgreSQL の高速フェイルオーバーに使える設定変更についていくつか説明します。PostgreSQL JDBC ドライバーのセットアップと設定の詳細については、PostgreSQL JDBC ドライバー のドキュメントを参照してください。

トピック

- DNS キャッシュタイムアウトの短縮
- Aurora PostgreSQL 接続文字列を高速フェイルオーバー用に設定する
- ホスト文字列を取得するためのその他のオプション

DNS キャッシュタイムアウトの短縮

フェイルオーバー後にアプリケーションが接続を確立しようとすると、新しい Aurora PostgreSQL ライターは前のリーダーになります。これは DNS の更新が完全に伝達される前に、Aurora の読み取り専用エンドポイントを使用して検索できます。Java DNS の有効期限 (TTL) を 30 秒以下のような低い値に設定すると、その後のリーダーノード間における接続試行のサイクルに役立ちます

```
// Sets internal TTL to match the Aurora RO Endpoint TTL
java.security.Security.setProperty("networkaddress.cache.ttl" , "1");
// If the lookup fails, default to something like small to retry
java.security.Security.setProperty("networkaddress.cache.negative.ttl" , "3");
```

Aurora PostgreSQL 接続文字列を高速フェイルオーバー用に設定する

Aurora PostgreSQL 高速フェイルオーバーを使用するには、アプリケーションの接続文字列が、単一のホストではなく、ホストの一覧になっている必要があります。Aurora PostgreSQL クラスターに接続するために使用できる接続文字列の例を次に示します。この例では、ホストは太字表示されています。

```
jdbc:postgresql://myauroracluster.cluster-c9bfei4hjlrd.us-east-1-
beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432
/postgres?user=<primaryuser>&password=<primarypw>&loginTimeout=2
&connectTimeout=2&cancelSignalTimeout=2&socketTimeout=60
&tcpKeepAlive=true&targetServerType=primary
```

最大限の可用性と RDS API への依存を回避するため、接続用のファイルを維持することをお勧めします。このファイルには、データベース接続の確立時にアプリケーションが読み込んだホスト文字列が含まれています。このホストの文字列には、クラスターで使用できるすべての Aurora エンドポイントがあります。Aurora エンドポイントの詳細については、「Amazon Aurora 接続管理」を参照してください。

例えば、以下のようにエンドポイントをローカルファイルに保存できます。

```
myauroracluster.cluster-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432
```

アプリケーションは、このファイルを読み込んで JDBC 接続文字列のホストセクションを入力します。DB クラスターの名前を変更すると、これらのエンドポイントが変更されます。このイベントが発生した場合は、アプリケーションによってこのイベントが処理されることを確認してください。

また、以下のように DB インスタンスノードの一覧を使用する方法もあります。

```
my-node1.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node2.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node3.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node4.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432
```

この方法の利点は、PostgreSQL JDBC 接続ドライバーがこの一覧のすべてのノードをループして有効な接続を検索することです。一方、Aurora エンドポイントを使用する場合、1 回の接続試行で 2 つのノードのみが試行されます。ただし、DB インスタンスノードを使用することにはデメリットが

あります。クラスターにノードを追加、または削除すると、インスタンスエンドポイントの一覧が古くなり、接続ドライバーが接続する接続ホストを見つけることができない場合があります。

以下のパラメータ設定を活用すると、アプリケーションがいずれか 1 つのホストに接続する際、必要以上に待機しないようにできます。

- targetServerType - ドライバーが書き込みノードと読み取りノードのどちらに接続するかを制御します。アプリケーションが書き込みノードにのみ再接続されるようにするには、targetServerType の値を primary に設定します。

  targetServerType パラメータの値は、primary、secondary、any、および preferSecondaryです。preferSecondary の値を指定すると、最初にリーダーへの接続の確立が試行されます。リーダーへの接続がない場合は、ライターに接続されます。
- loginTimeout - ソケット接続の確立後、アプリケーションがデータベースにログインするための待機時間を制御します。
- connectTimeout - ソケットがデータベースに接続を確立するまでの待機時間を制御します。

その他のアプリケーションパラメータを変更して、アプリケーションの希望する積極性に応じて、接続プロセスを高速化できます。

- cancelSignalTimeout - 一部のアプリケーションでは、タイムアウトがあるクエリで「ベストエフォート」型キャンセル信号を送信できます。このキャンセル信号がフェイルオーバーパスにある場合は、デッドホストにこの信号を送信しないように、この信号を積極的に設定することを検討してください。
- socketTimeout - このパラメータは、ソケットが読み取り操作で待機する時間を制御します。このパラメータは、グローバルな「クエリタイムアウト」として使用でき、すべてのクエリがこの値以上待機しないことを確保します。2 つの接続ハンドラーを用意することをお勧めします。1 つの接続ハンドラーが短期間のクエリを実行するため、この値を低く設定します。長時間クエリを実行する別の接続ハンドラーでは、この値を非常に高く設定します。この方法により、サーバーがダウンした場合に、TCP キープアライブパラメータによって長期間実行しているクエリを停止できます。
- tcpKeepAlive - このパラメータを有効にすると、設定した TCP キープアライブパラメータが優先されるようにします。
- loadBalanceHosts - true に設定すると、このパラメータは、選択可能なホストの一覧からランダムに選択されたホストにアプリケーションを接続します。

ホスト文字列を取得するためのその他のオプション

aurora_replica_status 関数を含めたいくつかの出典から、また Amazon RDS API を使用することで、ホスト文字列を取得できます。

多くの場合、クラスターのライターを特定するか、クラスター内の他のリーダーノードを検索する必要があります。そのために、アプリケーションでは、DB クラスター内の任意の DB インスタンスに接続し、aurora_replica_status 関数にクエリします。この関数を使用することで、接続先のホストを検索する時間を短縮できます。ただし、特定のネットワーク障害シナリオでは、aurora_replica_status 関数によって、最新ではない情報や不完全な情報を表示することがあります。

アプリケーションが接続先のノードを確実に見つけられるようにするために、クラスターライターエンドポイントへの接続を試行し、次にクラスターリーダーエンドポイントに接続するのが良い方法です。読み取り可能な接続を確立できるようになるまで、この操作を行います。これらのエンドポイントは、DB クラスターの名前を変更しない限り変更されません。したがって、一般的には、アプリケーションの静的メンバーとして残すか、アプリケーションが読み込むリソースファイルに保管することができます。

これらのエンドポイントの 1 つを使用して接続を確立したら、残りのクラスターの情報を取得できます。これを行うには、aurora_replica_status 関数を呼び出します。例えば、次のコマンドは aurora_replica_status で情報を取得します。

```
postgres=> SELECT server_id, session_id, highest_lsn_rcvd, cur_replay_latency_in_usec,
 now(), last_update_timestamp
FROM aurora_replica_status();

server_id | session_id | highest_lsn_rcvd | cur_replay_latency_in_usec | now |
 last_update_timestamp
-----------+--------------------------------------+------------------
+----------------------------+-------------------------------+-----------------------
mynode-1 | 3e3c5044-02e2-11e7-b70d-95172646d6ca | 594221001 | 201421 | 2017-03-07
 19:50:24.695322+00 | 2017-03-07 19:50:23+00
mynode-2 | 1efdd188-02e4-11e7-becd-f12d7c88a28a | 594221001 | 201350 | 2017-03-07
 19:50:24.695322+00 | 2017-03-07 19:50:23+00
mynode-3 | MASTER_SESSION_ID | | | 2017-03-07 19:50:24.695322+00 | 2017-03-07
 19:50:23+00
(3 rows)
```

例えば、接続文字列の [hosts] (ホスト) セクションは、ライタークラスターとリーダークラスターの両方のエンドポイントから始まる場合があります。

```
myauroracluster.cluster-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432
```

このシナリオでは、アプリケーションは、プライマリまたはセカンダリの任意のノードタイプに接続の確立を試行します。アプリケーションが接続されたら、最初にノードの読み書き状態を調べることをお勧めします。そのためには、コマンド SHOW transaction_read_only の結果をクエリします。

クエリの戻り値が OFF の場合、プライマリノードへの接続に成功しています。ただし、戻り値が ON で、アプリケーションによって読み書きの接続を要求しているとします。この場合、aurora_replica_status 関数を呼び出すことで、session_id='MASTER_SESSION_ID' を持つ server_id を判断できます。この関数は、プライマリノードの名前を表示します。後で説明する endpointPostfix と一緒に使用できます。

古いデータを持つレプリカに接続する場合は注意が必要です。この場合、aurora_replica_status 関数は、最新ではない情報を表示することがあります。古さのしきい値はアプリケーションレベルで設定できます。これを確認するには、サーバーの時間と last_update_timestamp の値との差で判断できます。一般に、アプリケーションでは、aurora_replica_status 関数から返される情報が矛盾することによる 2 つのホスト間での急変を避ける必要があります。アプリケーションでは aurora_replica_status から返されるデータに従うのではなく、最初にすべての既知のホストを試行する必要があります。

DescribeDBClusters API オペレーションを使用したインスタンスのリスト化 (Java の例)

インスタンスの一覧をプログラムで検索するには、AWS SDK for Java を使用し、特に DescribeDBClusters API オペレーションを使用します。

以下に、Java 8 でこれを行う方法の一例を紹介します。

```
AmazonRDS client = AmazonRDSClientBuilder.defaultClient();
DescribeDBClustersRequest request = new DescribeDBClustersRequest()
   .withDBClusterIdentifier(clusterName);
DescribeDBClustersResult result =
rdsClient.describeDBClusters(request);

DBCluster singleClusterResult = result.getDBClusters().get(0);

String pgJDBCEndpointStr =
singleClusterResult.getDBClusterMembers().stream()
   .sorted(Comparator.comparing(DBClusterMember::getIsClusterWriter)
   .reversed()) // This puts the writer at the front of the list
```

```
    .map(m -> m.getDBInstanceIdentifier() + endpointPostfix + ":" +
  singleClusterResult.getPort()))
    .collect(Collectors.joining(","));
```

ここで、以下に示すように pgJDBCEndpointStr にはエンドポイントの形式一覧が含まれています。

```
my-node1.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node2.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432
```

この変数 endpointPostfix には、アプリケーションが設定する定数を指定できます。または、アプリケーションがクラスター内の単一インスタンスの DescribeDBInstances API オペレーションをクエリすることで取得することもできます。この値は、AWS リージョン 内と個人のお客様では定数のままです。そのため、アプリケーションが読み取るリソースファイルにこの定数を保存するだけで、API 呼び出しを節約できます。前述の例では、以下のように設定されています。

```
.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com
```

API が応答しないか、応答するまでの時間が長すぎる場合、可用性を高めるために DB クラスターの Aurora エンドポイントをデフォルトで使用するように設定することをお勧めします。エンドポイントは、DNS レコードを更新するためにかかる時間内で最新状態に保たれることが保証されます。エンドポイントでの DNS レコードの更新は、通常 30 秒以内に完了します。エンドポイントは、アプリケーションが消費するリソースファイルに保存できます。

フェイルオーバーのテスト

いずれの場合も、2 つ以上の DB インスタンスを含む DB クラスターが必要です。

サーバー側では、特定の API オペレーションによって障害が発生する場合があり、アプリケーションがどのように応答するかテストするために使用できます。

- FailoverDBCluster - このオペレーションは、DB クラスターの新規の DB インスタンスをライターに昇格させようと試行します。

  次のコード例は、failoverDBCluster を使用して停止を引き起こす方法を示しています。Amazon RDS クライアントの設定の詳細については、AWS SDK for Java の使用を参照してください。

  ```
  public void causeFailover() {
  ```

```
    final AmazonRDS rdsClient = AmazonRDSClientBuilder.defaultClient();

    FailoverDBClusterRequest request = new FailoverDBClusterRequest();
    request.setDBClusterIdentifier("cluster-identifier");

    rdsClient.failoverDBCluster(request);
}
```

- RebootDBInstance - フェイルオーバーはこの API オペレーションでは保証されません。ただし、ライターのデータベースは停止します。これを使用すると、接続が切断された際にアプリケーションがどのように応答するかをテストできます。ForceFailover パラメータは Aurora エンジンには適用されません。代わりに、FailoverDBCluster API オペレーションを使用します。
- ModifyDBCluster - Port パラメータを変更すると、新しいポートでクラスター内のノードがリッスンを開始すると停止します。一般に、アプリケーションがポートの変更のみを制御することで、この障害に対応できます。また、依存するエンドポイントを適切に更新できることを確認します。これは、API レベルで変更が加えられた際に、手動でポートを更新することで実現できます。または、アプリケーションで RDS API を使用し、ポートが変更されたかどうかを確認できます。
- ModifyDBInstance— DBInstanceClass パラメータを変更すると停止します。
- DeleteDBInstance - プライマリ (ライター) を削除すると、新規の DB インスタンスが DB クラスターでライターに昇格します。

アプリケーション側またはクライアント側では、Linux を使用する場合、突然のパケットドロップに対してアプリケーションがどのように応答するかテストできます。これは、iptables コマンドを使用して、ポート、ホスト、または TCP キープアライブパケットの送受信の有無に基づき行うことができます。

高速フェイルオーバーの例 (Java)

以下のコード例では、アプリケーションがどのように Aurora PostgreSQL ドライバーマネージャーをセットアップできるかを示しています。

アプリケーションは、接続が必要になると getConnection 関数を呼び出します。getConnection への呼び出しでは、有効なホストが見つからないことがあります。一例として、ライターは見つかりませんが、targetServerType パラメータが primary に設定されています。この場合、呼び出し元のアプリケーションは、単に関数の呼び出しを再試行する必要があります。

この再試行動作を接続プーラーにまとめることで、再試行動作によってアプリケーションに与える影響を回避できます ほとんどの接続プーラーでは、JDBC 接続文字列を指定できます。したがって、

アプリケーションは getJdbcConnectionString を呼び出し、それを接続プーラーに渡すことができます。これにより、Aurora PostgreSQL でより高速なフェイルオーバーを使用できるようになります。

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

import org.joda.time.Duration;

public class FastFailoverDriverManager {
   private static Duration LOGIN_TIMEOUT = Duration.standardSeconds(2);
   private static Duration CONNECT_TIMEOUT = Duration.standardSeconds(2);
   private static Duration CANCEL_SIGNAL_TIMEOUT = Duration.standardSeconds(1);
   private static Duration DEFAULT_SOCKET_TIMEOUT = Duration.standardSeconds(5);

   public FastFailoverDriverManager() {
       try {
            Class.forName("org.postgresql.Driver");
       } catch (ClassNotFoundException e) {
            e.printStackTrace();
       }

       /*
         * RO endpoint has a TTL of 1s, we should honor that here. Setting this
 aggressively makes sure that when
         * the PG JDBC driver creates a new connection, it will resolve a new different
 RO endpoint on subsequent attempts
         * (assuming there is > 1 read node in your cluster)
         */
        java.security.Security.setProperty("networkaddress.cache.ttl" , "1");
       // If the lookup fails, default to something like small to retry
       java.security.Security.setProperty("networkaddress.cache.negative.ttl" , "3");
   }

   public Connection getConnection(String targetServerType) throws SQLException {
       return getConnection(targetServerType, DEFAULT_SOCKET_TIMEOUT);
   }
```

```
  public Connection getConnection(String targetServerType, Duration queryTimeout)
throws SQLException {
       Connection conn =
DriverManager.getConnection(getJdbcConnectionString(targetServerType, queryTimeout));

      /*
        * A good practice is to set socket and statement timeout to be the same thing
since both
        * the client AND server will stop the query at the same time, leaving no
running queries
        * on the backend
        */
       Statement st = conn.createStatement();
       st.execute("set statement_timeout to " + queryTimeout.getMillis());
       st.close();

      return conn;
  }

  private static String urlFormat = "jdbc:postgresql://%s"
          + "/postgres"
          + "?user=%s"
          + "&password=%s"
          + "&loginTimeout=%d"
          + "&connectTimeout=%d"
          + "&cancelSignalTimeout=%d"
          + "&socketTimeout=%d"
          + "&targetServerType=%s"
          + "&tcpKeepAlive=true"
          + "&ssl=true"
          + "&loadBalanceHosts=true";
  public String getJdbcConnectionString(String targetServerType, Duration
queryTimeout) {
      return String.format(urlFormat,
               getFormattedEndpointList(getLocalEndpointList()),
               CredentialManager.getUsername(),
               CredentialManager.getPassword(),
               LOGIN_TIMEOUT.getStandardSeconds(),
               CONNECT_TIMEOUT.getStandardSeconds(),
               CANCEL_SIGNAL_TIMEOUT.getStandardSeconds(),
               queryTimeout.getStandardSeconds(),
               targetServerType
      );
```

```
    }

    private List<String> getLocalEndpointList() {
        /*
          * As mentioned in the best practices doc, a good idea is to read a local
 resource file and parse the cluster endpoints.
          * For illustration purposes, the endpoint list is hardcoded here
          */
         List<String> newEndpointList = new ArrayList<>();
         newEndpointList.add("myauroracluster.cluster-c9bfei4hjlrd.us-east-1-
beta.rds.amazonaws.com:5432");
         newEndpointList.add("myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-
beta.rds.amazonaws.com:5432");

        return newEndpointList;
    }

    private static String getFormattedEndpointList(List<String> endpoints) {
        return IntStream.range(0, endpoints.size())
                .mapToObj(i -> endpoints.get(i).toString())
                .collect(Collectors.joining(","));
    }
}
```

Aurora PostgreSQL のクラスターキャッシュ管理によるフェイルオーバー後の高速リカバリ

フェイルオーバーが発生した場合に Aurora PostgreSQL クラスター内の書き込み DB インスタンスを迅速にリカバリするには、Amazon Aurora PostgreSQL のクラスターキャッシュ管理を使用します。クラスターキャッシュ管理を使用することで、フェイルオーバー発生時でもアプリケーションパフォーマンスを維持することができます。

一般的なフェイルオーバーが発生した場合、フェイルオーバー後に一時的だがパフォーマンスが大幅に低下することがあります。この低下は、フェイルオーバー DB インスタンスの起動時にバッファキャッシュが空になることが原因で発生します。空のキャッシュは、コールドキャッシュとも呼ばれます。コールドキャッシュでは、DB インスタンスは、低速のディスクから読み取る必要があるため、パフォーマンスが低下します。バッファキャッシュに格納されている値は使用されません。

クラスターキャッシュ管理では、特定の読み取り DB インスタンスをフェイルオーバーのターゲットとして設定します。クラスターキャッシュ管理により、指定された読み取りのキャッシュ内のデータは、書き込み DB インスタンスのキャッシュ内のデータと確実に同期されます。事前に入力された値

を持つ指定された読み取りのキャッシュは、ウォームキャッシュと呼ばれます。フェイルオーバーが発生すると、指定された読み取りでは、新しい書き込み DB インスタンスに昇格するとすぐにウォームキャッシュ内の値が使用されます。このアプローチでは、アプリケーションで優れたリカバリパフォーマンスを実現することができます。

クラスターキャッシュ管理では、指定されたリーダーインスタンスのインスタンスタイプとサイズ (`db.r5.2xlarge` または `db.r5.xlarge` など) が読み取りと同じである必要があります。Aurora PostgreSQL DB クラスターを作成すると、フェイルオーバー中のクラスターの回復が可能になることにご留意ください。インスタンスクラスのタイプとサイズの一覧については、Hardware specifications for DB instance classes for Aurora を参照してください。

> **Note**
>
> クラスターキャッシュ管理は、Aurora Global Database の一部である Aurora PostgreSQL DB クラスターではサポートされません。指定した Tier-0 リーダーではワークロードを実行しないことをお勧めします。

目次

クラスターキャッシュ管理の設定

クラスターキャッシュ管理を設定するには、以下のプロセスを順番に実行します。

トピック

> **Note**
>
> クラスターのキャッシュ管理が完全に機能するには、これらのステップを完了してから少なくとも 1 分以上かかります。

クラスターキャッシュ管理の有効化

クラスターキャッシュ管理を有効にするには、以下のステップを実行します。

コンソール

クラスターキャッシュ管理を有効にするには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、パラメータグループ を選択します。
3. リストで、Aurora PostgreSQL DB クラスターのパラメータグループを選択します。

   デフォルトのパラメータグループの値は変更することができないため、DB クラスターではデフォルトではないパラメータグループを使用する必要があります。
4. [Parameter group actions] (パラメータグループのアクション) で、編集 を選択します。
5. `apg_ccm_enabled` クラスターパラメータの値を [1] に設定します。
6. [Save changes] (変更を保存) をクリックします。

AWS CLI

Aurora PostgreSQL DB クラスターのクラスターキャッシュ管理を有効にするには、以下に示す必要なパラメータを指定して、AWS CLI の `modify-db-cluster-parameter-group` コマンドを使用します。

- `--db-cluster-parameter-group-name`
- `--parameters`

Example

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name my-db-cluster-parameter-group \
    --parameters "ParameterName=apg_ccm_enabled,ParameterValue=1,ApplyMethod=immediate"
```

Windows の場合:

```
aws rds modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name my-db-cluster-parameter-group ^
    --parameters "ParameterName=apg_ccm_enabled,ParameterValue=1,ApplyMethod=immediate"
```

書き込み DB インスタンスの昇格階層の優先度の設定

クラスターキャッシュ管理の場合、Aurora PostgreSQL DB クラスターの書き込み DB インスタンスの昇格階層の優先度が Tier 0 であることを確認します。昇格階層の優先度は、Aurora リーダーが失敗後に書き込み DB インスタンスに昇格する順序を指定する値です。有効な値は 0 ~ 15 です。ここで、0 は優先度が最も高く、15 が最も低いことを表します。昇格階層の詳細については、Aurora DB クラスターの耐障害性 を参照してください。

コンソール

書き込み DB インスタンスの昇格階層を Tier 0 に設定するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、データベースを選択します。
3. Aurora PostgreSQL DB クラスターの書き込み DB インスタンスを選択します。
4. Modify を選択します。Modify DB Instance ページが表示されます。
5. 追加設定パネルで、フェイルオーバー優先順位としてtier-0を選択します。
6. Continue を選択して、変更の概要を確認します。
7. 保存後に変更をすぐに反映させるには、すぐに適用を選択します。
8. DB インスタンスの変更 を選択して、変更を保存します。

AWS CLI

AWS CLI を使用して、書き込み DB インスタンスの昇格階層を 0 に設定するには、次の必要なパラメータを指定して、modify-db-instance コマンドを呼び出します。

- `--db-instance-identifier`
- `--promotion-tier`
- `--apply-immediately`

Example

Linux、macOS、Unix の場合:

```
aws rds modify-db-instance \
    --db-instance-identifier writer-db-instance \
    --promotion-tier 0 \
    --apply-immediately
```

Windows の場合:

```
aws rds modify-db-instance ^
    --db-instance-identifier writer-db-instance ^
    ---promotion-tier 0  ^
    --apply-immediately
```

読み取り DB インスタンスの昇格階層の優先度の設定

クラスターキャッシュ管理用にリーダー DB インスタンスを 1 つだけ設定する必要があります。そのためには、書き込み DB インスタンスと同じインスタンスクラスで同じサイズの Aurora PostgreSQL クラスターから、リーダーを選択します。例えば、読み取りが `db.r5.xlarge` を使用する場合、この同じインスタンスクラスターのタイプとサイズを使用するリーダーを選択します。その昇格階層の優先度を 0 に設定します。

昇格階層の優先度は、Aurora リーダーが失敗後に書き込み DB インスタンスに昇格する順序を指定する値です。有効な値は 0 ~ 15 です。ここで、0 は優先度が最も高く、15 が最も低いことを表します。

コンソール

読み取り DB インスタンスの昇格階層を Tier 0 に設定するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。

2. ナビゲーションペインで、[データベース] を選択します。
3. 書き込み DB インスタンスと同じインスタンスクラスである Aurora PostgreSQL DB クラスターの リーダー DB インスタンスを選択します。
4. Modify を選択します。Modify DB Instance ページが表示されます。
5. 追加設定パネルで、フェイルオーバー優先順位としてtier-0を選択します。
6. Continue を選択して、変更の概要を確認します。
7. 保存後に変更をすぐに反映させるには、すぐに適用を選択します。
8. DB インスタンスの変更 を選択して、変更を保存します。

AWS CLI

AWS CLI を使用して、読み取り DB インスタンスの昇格階層を 0 に設定するには、次の必要なパラメータを指定して、modify-db-instance コマンドを呼び出します。

- `--db-instance-identifier`
- `--promotion-tier`
- `--apply-immediately`

Example

Linux、macOS、Unix の場合:

```
aws rds modify-db-instance \
    --db-instance-identifier reader-db-instance \
    --promotion-tier 0 \
    --apply-immediately
```

Windows の場合:

```
aws rds modify-db-instance ^
    --db-instance-identifier reader-db-instance ^
    ---promotion-tier 0  ^
    --apply-immediately
```

バッファキャッシュのモニタリング

クラスターキャッシュ管理を設定したら、書き込み DB インスタンスのバッファキャッシュと指定された読み取りのウォームバッファキャッシュの間の同期の状態をモニタリングできます。書き込み DB インスタンスと、指定された読み取り DB インスタンスの両方のバッファキャッシュコンテンツを調べるには、PostgreSQL pg_buffercache モジュールを使用します。詳細については、pg_buffercache PostgreSQL ドキュメントを参照してください。

aurora_ccm_status 関数を使用する

また、クラスターキャッシュ管理では、aurora_ccm_status 関数も使用できます。指定された読み取りのキャッシュウォームの進行状況に関する次の情報を取得するには、書き込みの DB インスタンスの aurora_ccm_status 関数を使用します。

- buffers_sent_last_minute - 最後の 1 分間に指定された読み取りに送信されたバッファの数。
- buffers_found_last_minute – 過去 1 分間に識別されたアクセス頻度の高いバッファの数。
- buffers_sent_last_scan - バッファキャッシュの最後の完全スキャン中に指定された読み取りに送信されたバッファの数。
- buffers_found_last_scan - バッファキャッシュの完全な最終スキャン中に、頻繁にアクセスされ、送信する必要があると識別されたバッファの数。指定された読み取りに既にキャッシュされているバッファは送信されません。
- buffers_sent_current_scan - 現在のスキャン中にこれまでに送信されたバッファの数。
- buffers_found_current_scan - 現在のスキャンで頻繁にアクセスされたと識別されたバッファの数。
- current_scan_progress - 現在のスキャン中にこれまでにアクセスされたバッファの数。

次の例は、aurora_ccm_status 関数を使用してその出力の一部をウォームレートとウォームの割合 (%) に変換する方法を示しています。

```
SELECT buffers_sent_last_minute*8/60 AS warm_rate_kbps,
   100*(1.0-buffers_sent_last_scan::float/buffers_found_last_scan) AS warm_percent
   FROM aurora_ccm_status();
```

CCM 設定のトラブルシューティング

apg_ccm_enabled クラスターパラメータを有効にすると、ライター DB インスタンスのインスタンスレベルでクラスターキャッシュ管理が自動的にオンになり、Aurora PostgreSQL DB クラスターの 1 つのリーダー DB インスタンスで有効になります。ライターインスタンスとリーダーインスタンスは、同じインスタンスクラスタイプとサイズを使用する必要があります。昇格階層の優先度は 0 に設定されます。DB クラスター内の他のリーダーインスタンスにはゼロ以外の昇格階層が必要で、クラスターキャッシュ管理はそれらのインスタンスに対して無効になっています。

設定に問題が発生し、クラスターキャッシュ管理が無効になる理由は次のとおりです。

- 昇格階層 0 に設定されているリーダー DB インスタンスが 1 つもない場合。
- ライター DB インスタンスが昇格階層 0 に設定されていない場合。
- 複数のリーダー DB インスタンスが昇格階層 0 に設定されている場合。
- 昇格階層 0 のライター DB インスタンスと 1 つのリーダー DB インスタンスが同じインスタンスサイズでない場合。

管理する Aurora PostgreSQL 接続チャーンとプーリング

クライアントアプリケーションの接続と切断が頻繁に行われ、Aurora PostgreSQL DB クラスターの応答時間が遅くなると、クラスターに接続チャーンが発生しているとされます。Aurora PostgreSQL DB クラスターエンドポイントに新たに接続するたびにリソースが消費されるため、実際のワークロードの処理に使用できるリソースが減少します。接続チャーンについては、次のベストプラクティスに沿って対応することをお勧めします。

手始めに、接続チャーンが多い Aurora PostgreSQL DB クラスターの応答時間を改善できます。これを実行するには、RDS プロキシなどの接続プーラーを使用できます。接続プーラーによって、クライアントが接続のキャッシュをすぐに使用できます。Aurora PostgreSQL のほぼすべてのバージョンで RDS プロキシをサポートしています。詳細については、「Aurora PostgreSQL による Amazon RDS Proxy」を参照してください。

お使いの Aurora PostgreSQL の特定のバージョンが RDS プロキシをサポートしていない場合は、PgBouncer など、PostgreSQL 互換のある別の接続プーラーを使用できます。詳細については、PGBouncer のウェブサイトを参照してください。

Aurora PostgreSQL DB クラスターが接続プールのメリットがあるかどうかを確認するには、postgresql.log ファイルの接続および切断でチェックできます。また、Performance

Insights を使用して、Aurora PostgreSQL DB クラスターで接続チャーンがどの程度発生しているかを確認できます。以下は、この 2 つのトピックについての情報です。

接続と切断のログ

PostgreSQL `log_connections` および `log_disconnections` パラメータにより、Aurora PostgreSQL DB クラスターのライターインスタンス、への接続と切断をキャプチャできます。デフォルトでは、これらのパラメータはオフになっています。これらのパラメータをオンにするには、カスタムパラメータグループを使用し、値を 1 に変更することでオンになります。カスタムパラメータグループの詳細については、「DB クラスターパラメータグループを使用する」を参照してください。設定を確認するには、psql を使用して Aurora PostgreSQL の DB クラスターエンドポイントに接続し、次のようにクエリを実行します。

```
labdb=> SELECT setting FROM pg_settings
  WHERE name = 'log_connections';
 setting
---------
on
(1 row)
labdb=> SELECT setting FROM pg_settings
  WHERE name = 'log_disconnections';
setting
---------
on
(1 row)
```

この 2 つのパラメータを両方ともオンにすると、ログには新しい接続と切断がすべて記録されます。新しく許可された接続ごとに、ユーザーとデータベースが表示されます。次の例に示されているように、切断時にはセッションの継続時間もログに記録されます。

```
2022-03-07 21:44:53.978 UTC [16641] LOG: connection authorized: user=labtek
 database=labdb application_name=psql
2022-03-07 21:44:55.718 UTC [16641] LOG: disconnection: session time: 0:00:01.740
 user=labtek database=labdb host=[local]
```

アプリケーションの接続チャーンをチェックするため、これらのパラメータがまだオンになっていない場合はオンにします。次に、実際のワークロードと期間でアプリケーションを実行して、分析用に PostgreSQL ログにデータを収集します。ログファイルは、RDS コンソールで表示できます。Aurora PostgreSQL DB クラスターのライターインスタンスを選択し、[Logs & events] (ログと

イベント) タブをクリックします。詳細については、「データベースログファイルの表示とリスト化」を参照してください。

または、コンソールからログファイルをダウンロードし、次のコマンドシーケンスを使用することもできます。このシーケンスでは、1 分間に許可された接続と切断された接続の合計を求めます。

```
grep "connection authorized\|disconnection: session time:"
 postgresql.log.2022-03-21-16|\
awk {'print $1,$2}' |\
sort |\
uniq -c |\
sort -n -k1
```

出力例では、16:12:10 に認証済み接続が急増し、その後切断していることがわかります。

```
.....
,.......
.........
5 2022-03-21 16:11:55 connection authorized:
9 2022-03-21 16:11:55 disconnection: session
5 2022-03-21 16:11:56 connection authorized:
5 2022-03-21 16:11:57 connection authorized:
5 2022-03-21 16:11:57 disconnection: session
32 2022-03-21 16:12:10 connection authorized:
30 2022-03-21 16:12:10 disconnection: session
31 2022-03-21 16:12:11 connection authorized:
27 2022-03-21 16:12:11 disconnection: session
27 2022-03-21 16:12:12 connection authorized:
27 2022-03-21 16:12:12 disconnection: session
41 2022-03-21 16:12:13 connection authorized:
47 2022-03-21 16:12:13 disconnection: session
46 2022-03-21 16:12:14 connection authorized:
41 2022-03-21 16:12:14 disconnection: session
24 2022-03-21 16:12:15 connection authorized:
29 2022-03-21 16:12:15 disconnection: session
28 2022-03-21 16:12:16 connection authorized:
24 2022-03-21 16:12:16 disconnection: session
40 2022-03-21 16:12:17 connection authorized:
42 2022-03-21 16:12:17 disconnection: session
40 2022-03-21 16:12:18 connection authorized:
40 2022-03-21 16:12:18 disconnection: session
.....
,.......
```

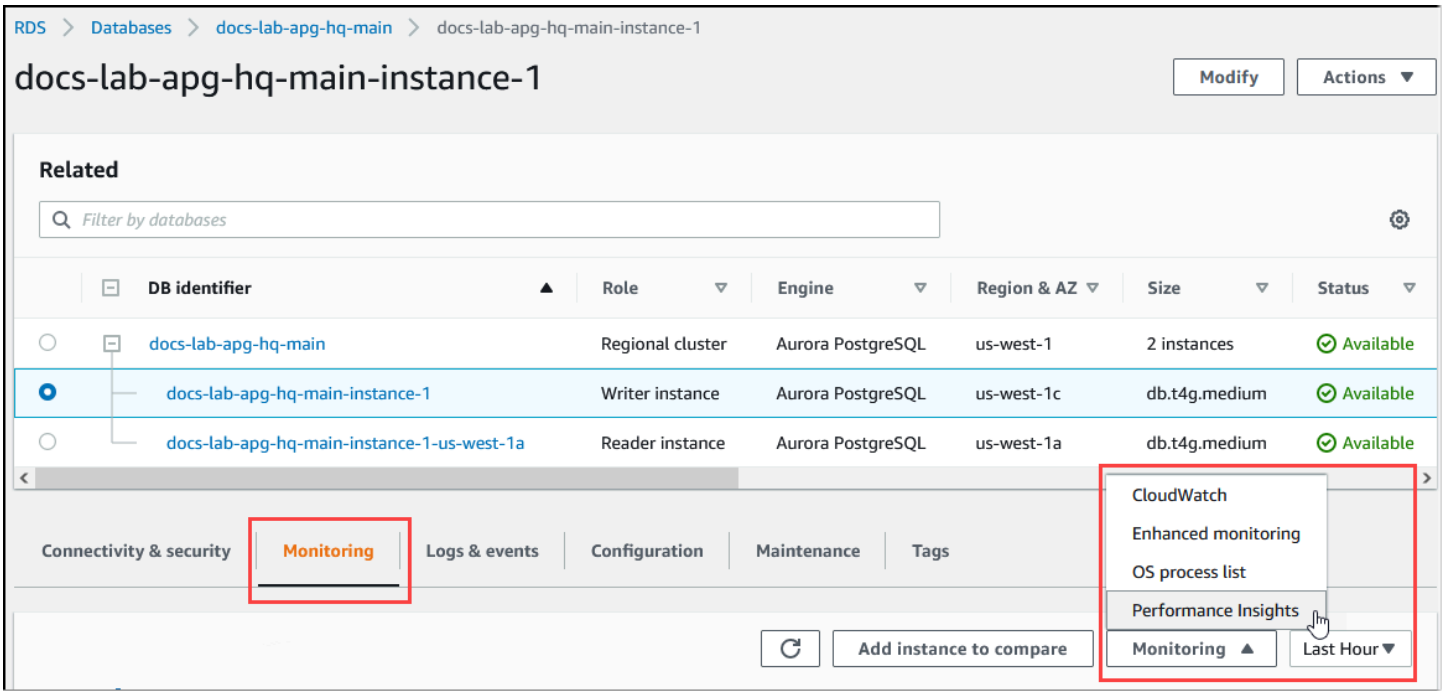
```
.........
1 2022-03-21 16:14:10 connection authorized:
1 2022-03-21 16:14:10 disconnection: session
1 2022-03-21 16:15:00 connection authorized:
1 2022-03-21 16:16:00 connection authorized:
```

この情報により、ワークロードに対して接続プーラーが有効かどうかを判断できます。より詳細な分析には、Performance Insights を使用できます。

Performance Insights による接続チャーンの検出

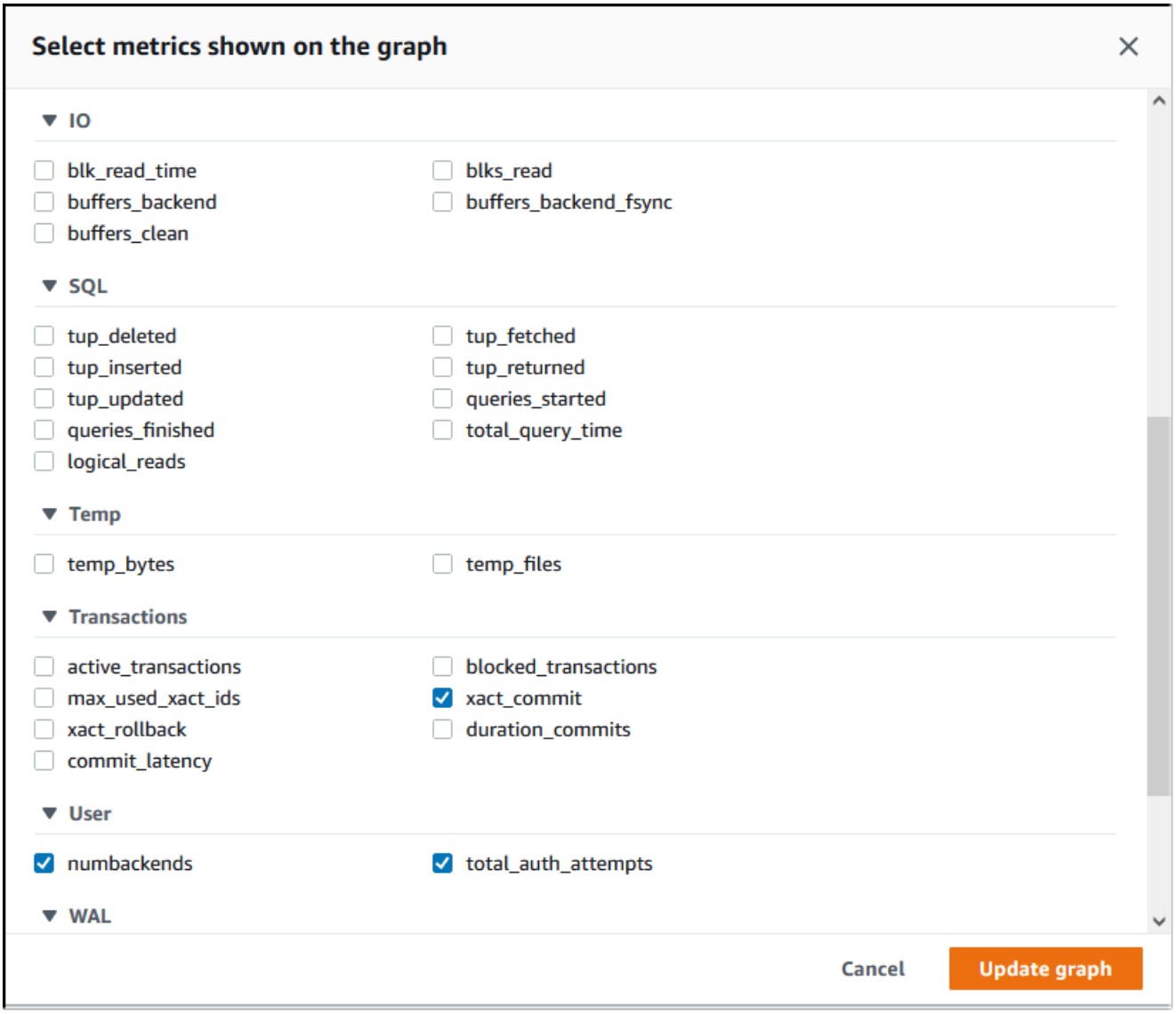
Performance Insights を使用して、Aurora PostgreSQL 互換エディション DB クラスター接続チャーンの量を評価できます。Aurora PostgreSQL DB クラスターを作成する際には、Performance Insights の設定はデフォルトでオンになります。DB クラスターの作成時にこの選択をオフした場合は、クラスターを変更して機能をオンにします。詳細については、「Amazon Aurora DB クラスターの変更」を参照してください。

Aurora PostgreSQL DB クラスターで Performance Insights を実行すると、モニタリングするメトリクスを選択できます。Performance Insights には、コンソールのナビゲーションペインからアクセスできます。また、次のイメージのように、Aurora PostgreSQL DB クラスターのライターインスタンスの [Monitoring] (モニタリング) タブから Performance Insights にアクセスできます。



Performance Insights コンソールから、[Manage metrics] (メトリクスの管理) を選択します。Aurora PostgreSQL DB クラスターの接続アクティビティと切断アクティビティを分析するには、次のメトリクスを選択します。これらはすべて PostgreSQL のメトリクスです。

- `xact_commit` — コミットされたトランザクション数。
- `total_auth_attempts` — 認証されたユーザーの 1 分あたりの接続試行数。
- `numbackends` — 現在データベースに接続されているバックエンド数。

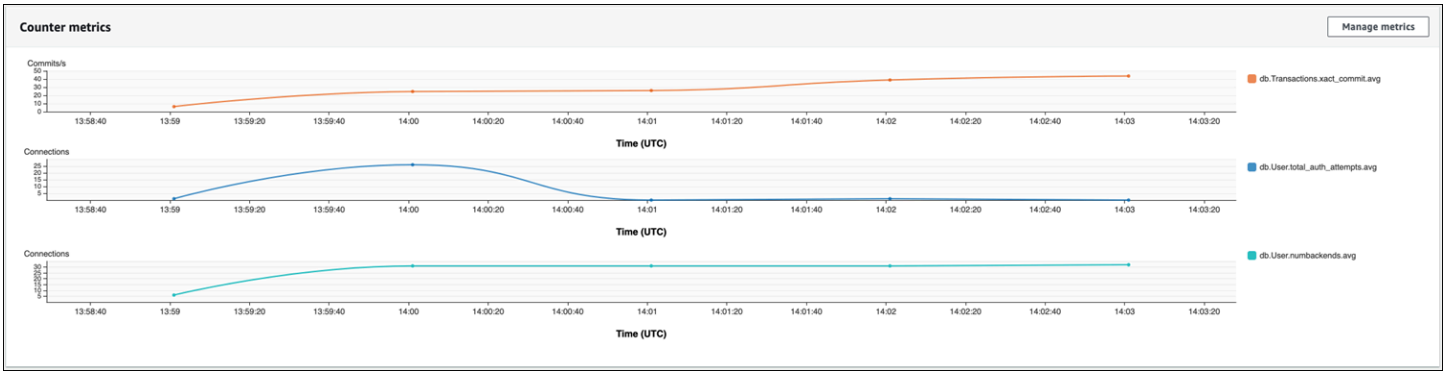


設定を保存し、接続アクティビティを表示するには、[Update graph] (グラフの更新) を選択します。

次のイメージでは、100 人のユーザーで pgbench を実行した場合の影響を確認できます。接続を示す線は、一貫して右肩上がりになっています。pgbench の詳細と使用方法については、PostgreSQL のドキュメントの「pgbench」を参照してください。

このイメージは、接続プーラーなしでわずか 100 人のユーザーでワークロードを実行した場合でも、ワークロードの処理期間中に `total_auth_attempts` の数が大幅に増加することを示しています。

RDS プロキシ接続プーリングでは、ワークロードの開始時に接続試行回数が増加します。接続プールを設定すると、平均値は低下します。トランザクションとバックエンドによって使用されるリソースは、ワークロードの処理中は常に一定に保たれます。



Aurora PostgreSQL DB クラスターでの Performance Insights 使用の詳細については、「Amazon Aurora での Performance Insights を使用したDB 負荷のモニタリング」を参照してください。メトリクスを分析するには、「Performance Insights ダッシュボードを使用してメトリクスを分析する」を参照してください。

接続プーリングの利点のデモンストレーション

前述のとおり、Aurora PostgreSQL DB クラスターに接続チャーンの問題があると判断した場合は、RDS プロキシを使用してパフォーマンスを向上させることができます。以下に、接続をプール

する場合としない場合のワークロード処理の違いを示す例を示します。この例では、pgbench を使用してトランザクションワークロードをモデル化しています。

psql と同様に、pgbench はローカルクライアントマシンからインストールして実行できる PostgreSQL クライアントアプリケーションです。また、Aurora PostgreSQL DB クラスターの管理に使用する Amazon EC2 インスタンスからインストールして実行できます。詳細については、PostgreSQL のドキュメントの「pgbench」を参照してください。

この例で順番に説明すると、まず、データベースに pgbench 環境を作成します。次のコマンドは、指定されたデータベースの pgbench テーブルを初期化するための基本的なテンプレートです。この例では、ログイン用にデフォルトのメインユーザーアカウントである `postgres` を使用しています。Aurora PostgreSQL DB クラスターに合わせ、必要に応じて変更します。pgbench 環境は、クラスターのライターインスタンス上のデータベースに作成します。

> **Note**
>
> pgbench の初期化プロセスでは、`pgbench_accounts`、`pgbench_branches`、`pgbench_history`、`pgbench_tellers` という名前のテーブルが削除され、再作成されます。pgbench を初期化する際には、これらの名前が *`dbname`* に選択したデータベースに使用されていないことを確認してください。

```
pgbench -U postgres -h db-cluster-instance-1.111122223333.aws-region.rds.amazonaws.com
 -p 5432 -d -i -s 50 dbname
```

pgbench では、次のパラメータを指定します。

-d

pgbench の実行時にデバッグレポートを出力します。

-h

Aurora PostgreSQL DB クラスターのライターインスタンスのエンドポイントを指定します。

-i

ベンチマークテスト用にデータベースの pgbench 環境を初期化します。

-p

データベース接続に使用されるポートを指定します。Aurora PostgreSQL のデフォルトは、通常 5432 または 5433 です。

-s

テーブルに行を入力する際に使用するスケーリング係数を指定します。デフォルトのスケーリング係数 1 で、pgbench_branches テーブルに 1 行、pgbench_tellers テーブルに 10 行、pgbench_accounts テーブルに 100,000 行が生成されます。

-U

Aurora PostgreSQL DB クラスターのライターインスタンスのユーザーアカウントを指定します。

pgbench 環境をセットアップしたら、接続プールの有無を問わずベンチマークテストを実行できます。デフォルトのテストは、トランザクションごとに 5 つの SELECT、UPDATE、INSERT コマンドが、指定された時間繰り返し実行されます。スケーリング係数、クライアント数などの詳細を指定して、独自のユースケースをモデル化できます。

例として、次のコマンドでは 20 の同時接続 (-c オプション) で 60 秒間 (-T オプション、時間) でベンチマークを実行します。-C オプションでは、クライアントセッションごとに 1 回ではなく、毎回新しい接続を使用してテストを実行します。この設定により、接続のオーバーヘッドがわかります。

```
pgbench -h docs-lab-apg-133-test-instance-1.c3zr2auzukpa.us-west-1.rds.amazonaws.com -U
 postgres -p 5432 -T 60 -c 20 -C labdb
Password:**********
pgbench (14.3, server 13.3)
  starting vacuum...end.
  transaction type: <builtin: TPC-B (sort of)>
    scaling factor: 50
    query mode: simple
    number of clients: 20
    number of threads: 1
    duration: 60 s
    number of transactions actually processed: 495
    latency average = 2430.798 ms
    average connection time = 120.330 ms
    tps = 8.227750 (including reconnection times)
```

接続を再利用せずに Aurora PostgreSQL DB クラスターのライターインスタンスで pgbench を実行した場合、1 秒間で約 8 件のトランザクションしか処理されていないことがわかります。これにより、1 分間のテストで合計 495 件のトランザクションが実行されます。

接続を再利用すると、ユーザー数に対する Aurora PostgreSQL DB クラスターからの応答は約 20 倍速くなります。再利用の場合、同じ時間、同じユーザー数の接続で 495 件のトランザクションが処理されるのに対し、合計 9,042 件のトランザクションが処理されます。次では、各接続が再利用される点が異なります。

```
pgbench -h docs-lab-apg-133-test-instance-1.c3zr2auzukpa.us-west-1.rds.amazonaws.com -U
 postgres -p 5432 -T 60 -c 20 labdb
Password:*********
pgbench (14.3, server 13.3)
      starting vacuum...end.
      transaction type: <builtin: TPC-B (sort of)>
      scaling factor: 50
      query mode: simple
      number of clients: 20
      number of threads: 1
      duration: 60 s
      number of transactions actually processed: 9042
      latency average = 127.880 ms
      initial connection time = 2311.188 ms
      tps = 156.396765 (without initial connection time)
```

この例では、接続をプールすることで応答時間を大幅に改善できることを示しています。Aurora PostgreSQL DB クラスターに RDS プロキシをセットアップする方法については、「Amazon RDS Proxy for Aurora の使用」を参照してください。

Aurora PostgreSQL のメモリパラメータの調整

Amazon Aurora PostgreSQL では、さまざまな処理タスクに使用されるメモリ量を制御する複数のパラメータを使用できます。タスクが特定のパラメータに設定されている量よりも多くのメモリを消費する場合、Aurora PostgreSQL ではディスクへの書き込みなどの処理に他のリソースを使用します。このため、Aurora PostgreSQL DB クラスターが遅くなったり、メモリ不足エラーが発生して停止したりする可能性があります。

各メモリパラメータのデフォルト設定によって、通常は目的の処理タスクを処理できます。ただし、Aurora PostgreSQL DB クラスターの のメモリ関連のパラメータをチューニングすることもできます。このチューニングを行うと、特定のワークロードの処理に十分なメモリが割り当てられるようになります。

メモリ管理を制御するパラメータについての情報は、以下を参照してください。メモリ使用率を評価する方法も説明しています。

パラメータ値の確認と設定

メモリを管理し、Aurora PostgreSQL DB クラスターのメモリ使用量を評価するために設定できるパラメータには以下のものがあります。

- work_mem — Aurora PostgreSQL DB クラスターが一時ディスクファイルに書き込む前に、内部ソート操作とハッシュテーブルで使用するメモリ量を指定します。
- log_temp_files — 一時ファイルの作成、ファイル名、サイズをログに記録します。このパラメータをオンにすると、作成される一時ファイルごとにログエントリが保存されます。これをオンにすると、Aurora PostgreSQL DB クラスターがディスクに書き込む必要がある頻度を確認できます。過剰なログの記録を避けるため、Aurora PostgreSQL DB クラスターの一時ファイル生成に関する情報を収集したら、再度オフにしてください。
- logical_decoding_work_mem — 論理デコードに使用するメモリ量 (メガバイト単位) を指定します。論理デコードは、レプリカの作成に使用するプロセスです。このプロセスは、ログ先行書き込み (WAL) ファイルのデータをターゲットが必要とする論理ストリーミング出力に変換することによって行われます。

  このパラメータ値により、各レプリケーション接続に指定されたサイズの単一のバッファが作成されます。デフォルトでは 65536 KB です。このバッファがいっぱいになると、超過分はファイルとしてディスクに書き込まれます。ディスクのアクティビティを最小化するため、このパラメータ値を work_mem より非常に高い値に設定ができます。

これらはすべて動的パラメータなので、現在のセッションで変更できます。これを実行するには、次に示すように psql で Aurora PostgreSQL DB クラスター に接続し、SET ステートメントを使用します。

```
SET parameter_name TO parameter_value;
```

セッション設定は、そのセッションの有効期間中のみ有効です。セッションが終了すると、パラメータは DB クラスターのパラメータグループの設定に戻ります。パラメータを変更する前に、次のように最初に pg_settings テーブルをクエリして現在の値を確認します。

```
SELECT unit, setting, max_val
   FROM pg_settings WHERE name='parameter_name';
```

例えば、work_mem パラメータの値を検索するには、Aurora PostgreSQL DB クラスターのライターインスタンスに接続し、次のクエリを実行します。

```
SELECT unit, setting, max_val, pg_size_pretty(max_val::numeric)
  FROM pg_settings WHERE name='work_mem';
unit  | setting  | max_val    | pg_size_pretty
------+----------+------------+----------------
 kB   | 1024     | 2147483647| 2048 MB
(1 row)
```

パラメータ設定を変更して設定を永続化させるには、カスタム DB クラスターのパラメータグループを使用する必要があります。SET ステートメントを使用して、これらのバラメータに異なる値を Aurora PostgreSQL DB クラスター に付与した後、カスタムパラメータグループを作成し、Aurora PostgreSQL DB クラスターに適用できます。詳細については、「「パラメータグループを使用する」」を参照してください。

ワーキングメモリパラメータの概要

ワーキングメモリパラメータ (work_mem) には、Aurora PostgreSQL が複雑なクエリを処理するために使用可能な最大メモリ量を指定します。複雑なクエリには、ソートやグループ化の操作、つまり次のような句を使用するものが含まれます。

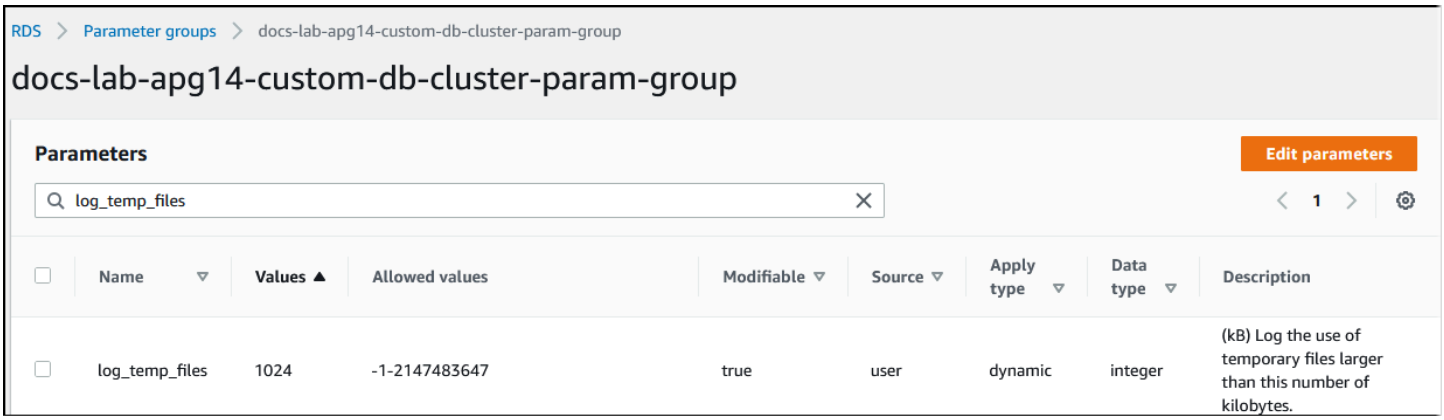
- ORDER BY
- DISTINCT
- GROUP BY
- JOIN (MERGE および HASH)

クエリプランナーは、Aurora PostgreSQL DB クラスターがどのようにワーキングメモリを使用するかについて間接的に影響を与えます。クエリプランナーは、SQL ステートメントを処理するための実行計画を生成します。特定の計画では、複雑なクエリを複数の作業単位に分割し、並行して実行できます。Aurora PostgreSQL では、各並列プロセスでディスクへの書き込み前に、各セッションの work_mem パラメータで指定されたメモリ量を使用します。

複数のデータベースユーザーが複数の操作を同時に実行し、複数の作業単位を並行して生成すると、Aurora PostgreSQL DB クラスターに割り当てられた作業メモリを使い果たしてしまう可能性があります。このため一時ファイルの作成やディスク I/O が過剰になり、最悪の場合はメモリ不足でエラーが発生する可能性があります。

一時ファイルの使用状況の確認

クエリの処理に必要なメモリが、work_mem で指定された値を超えた場合は、作業データは一時ファイルのディスクにオフロードされます。その発生頻度を確認するには、log_temp_files パラメータをオンにします。デフォルトでは、このパラメータはオフ (-1 に設定) に設定されています。すべての一時ファイル情報をキャプチャするには、このパラメータを 0 に設定します。log_temp_files に他の正の整数を設定すると、そのデータ量 (キロバイト単位) 以上のファイルの一時ファイル情報をキャプチャできます。次のイメージでは、AWS Management Console からの例を示します。



一時ファイルのログの記録を設定したら、自身のワークロードでテストして、ワーキングメモリの設定が十分かどうかを確認できます。また、PostgreSQL コミュニティのシンプルなベンチマークアプリケーションである pgbench を使用して、ワークロードをシミュレーションすることができます。

次の例では、テストを実行するために必要なテーブルと行を作成して、(-i) pgbench を初期化します。この例では、スケーリング係数 (-s 50) により、labdb データベースの pgbench_branches テーブルに 50行、pgbench_tellers テーブルに 500 行、pgbench_accounts テーブルに 5,000,000 行が作成されます。

```
pgbench -U postgres -h your-cluster-instance-1.111122223333.aws-regionrds.amazonaws.com
 -p 5432 -i -s 50 labdb
Password:
dropping old tables...
NOTICE:  table "pgbench_accounts" does not exist, skipping
NOTICE:  table "pgbench_branches" does not exist, skipping
NOTICE:  table "pgbench_history" does not exist, skipping
NOTICE:  table "pgbench_tellers" does not exist, skipping
creating tables...
generating data (client-side)...
5000000 of 5000000 tuples (100%) done (elapsed 15.46 s, remaining 0.00 s)
vacuuming...
```

```
creating primary keys...
done in 61.13 s (drop tables 0.08 s, create tables 0.39 s, client-side generate 54.85
 s, vacuum 2.30 s, primary keys 3.51 s)
```

環境を初期化したら、時間 (-T) とクライアント数 (-c) を指定してベンチマークを実行できます。この例では、-d オプションを使用して Aurora PostgreSQL DB クラスターによってトランザクションが処理される際にデバッグ情報も出力します。

```
pgbench -h -U postgres your-cluster-instance-1.111122223333.aws-regionrds.amazonaws.com
 -p 5432 -d -T 60 -c 10 labdb
Password:*******
pgbench (14.3)
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 50
query mode: simple
number of clients: 10
number of threads: 1
duration: 60 s
number of transactions actually processed: 1408
latency average = 398.467 ms
initial connection time = 4280.846 ms
tps = 25.096201 (without initial connection time)
```

pgbench の詳細については、PostgreSQL のドキュメントの「pgbench」を参照してください。

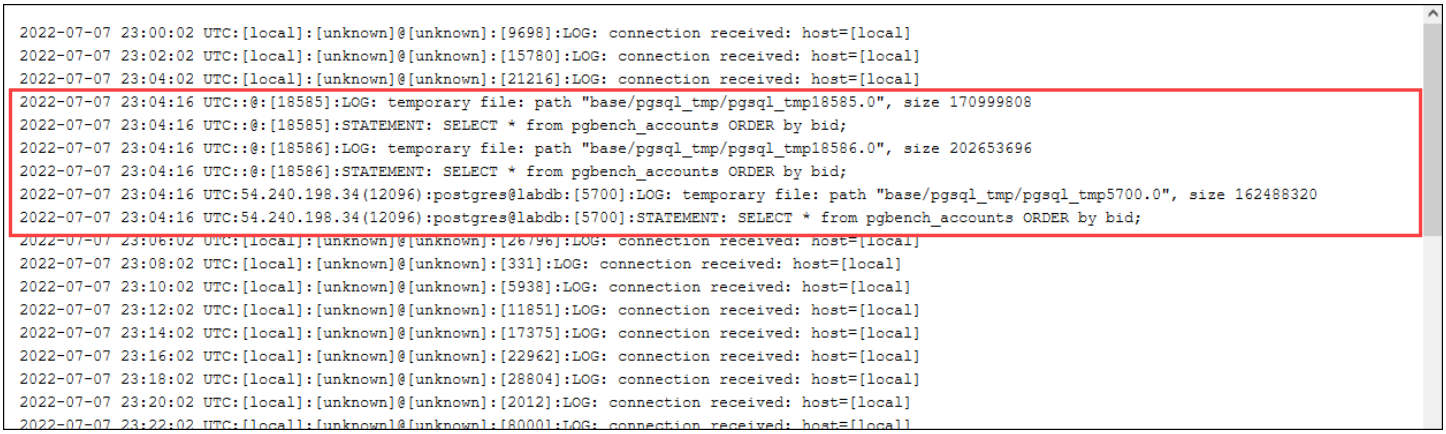
psql メタコマンド (\d) を使用して、pgbench が作成したテーブル、ビュー、インデックスなどのリレーションを一覧表示できます。

```
labdb=>  \d pgbench_accounts
 Table "public.pgbench_accounts"
  Column  |     Type      | Collation | Nullable | Default
----------+---------------+-----------+----------+---------
 aid      | integer       |           | not null |
 bid      | integer       |           |          |
 abalance | integer       |           |          |
 filler   | character(84) |           |          |
Indexes:
    "pgbench_accounts_pkey" PRIMARY KEY, btree (aid)
```

出力に示されているように、pgbench_accounts テーブルは aid 列にインデックス表示されています。この次のクエリがワーキングメモリを使用することを確認するには、次の例に示すように、インデックスが設定されていない列をすべてクエリします。

```
postgres=> SELECT * FROM pgbench_accounts ORDER BY bid;
```

一時ファイルのログを確認してください。これで実行するには、AWS Management Console を開き、Aurora PostgreSQL DB クラスターのライターインスタンスを選択し、[Logs & Events] (ログとイベント) タブをクリックします。コンソールでこのログを表示するか、ダウンロードして詳細を分析します。次のイメージに示すように、クエリの処理に必要な一時ファイルのサイズから、work_mem パラメータに指定する量を増やす検討も必要なことがわかります。



このパラメータは運用する上で、必要に応じて個人やグループごとに異なる設定をすることができます。例えば、dev_team という名前のロールに対し、work_mem パラメータを 8 GB に設定できます。

```
postgres=> ALTER ROLE dev_team SET work_mem='8GB';
```

work_mem に対するこの設定では、dev_team ロールのメンバーであるすべてのロールに最大 8 GB のワーキングメモリが割り当てられます。

インデックスの使用による応答時間の短縮

クエリの結果が返されるまでに時間がかかりすぎる場合は、インデックスが想定どおりに使用されているかどうかを確認できます。まず、次のように psql メタコマンドの \timing をオンにします。

```
postgres=>  \timing on
```

timing をオンにしたら、シンプルな SELECT ステートメントを使用します。

```
postgres=> SELECT COUNT(*) FROM
  (SELECT * FROM pgbench_accounts
  ORDER BY bid)
  AS accounts;
count
-------
5000000
(1 row)
Time: 3119.049 ms (00:03.119)
```

出力に示されているように、このクエリは完了するのに 3 秒以上かかりました。応答時間を改善するには、次のように pgbench_accounts にインデックスを作成します。

```
postgres=> CREATE INDEX ON pgbench_accounts(bid);
CREATE INDEX
```

クエリを再実行すると、応答時間が短くなったことがわかります。この例では、クエリは約 5 倍速く、約 0.5 秒で完了しました。

```
postgres=>  SELECT COUNT(*) FROM (SELECT * FROM pgbench_accounts ORDER BY bid) AS
 accounts;
 count
-------
 5000000
(1 row)
Time: 567.095 ms
```

論理デコード用のワーキングメモリの調整

論理レプリケーションは、PostgreSQL バージョン 10 で導入された以降、Aurora PostgreSQL のすべてのバージョンで使用できます。また、論理レプリケーションを設定する場合、logical_decoding_work_mem パラメータを使用して、論理デコードプロセスがデコードおよびストリーミングプロセスに使用できるメモリ量を指定できます。

論理デコード中に、ログ先行書き込み (WAL) レコードは SQL ステートメントに変換され、論理レプリケーションまたは別のタスクのために別のターゲットに送信されます。トランザクションを WAL に書き込んで変換する場合、トランザクション全体が logical_decoding_work_mem で指定された値の範囲に収まる必要があります。デフォルトでは、このパラメータは 65536 KB に設定されています。オーバーフローはすべてディスクに書き込まれます。つまり、送信先に送信する前にディスクからの再読み取りが必要なため、プロセス全体が遅くなります。

次の例に示すように、aurora_stat_file 関数を使用することで、特定の時点における現在のワークロードのトランザクションオーバーフローの量を評価できます。

```
SELECT split_part (filename, '/', 2)
   AS slot_name, count(1) AS num_spill_files,
   sum(used_bytes) AS slot_total_bytes,
   pg_size_pretty(sum(used_bytes)) AS slot_total_size
   FROM aurora_stat_file()
   WHERE filename like '%spill%'
   GROUP BY 1;
  slot_name | num_spill_files | slot_total_bytes | slot_total_size
------------+-----------------+------------------+-----------------
 slot_name  |       590       |      411600000   | 393 MB
(1 row)
```

このクエリは、クエリが呼び出された際に Aurora PostgreSQL DB クラスター上のスピルファイルの数とサイズを返します。長時間実行されるワークロードでは、ディスク上にスピルファイルがまだ存在しない可能性があります。長時間実行されるワークロードをプロファイリングするには、ワークロードの実行中にオーバーフローファイル情報をキャプチャするテーブルを作成することをお勧めします。次に示すようにテーブルを作成できます。

```
CREATE TABLE spill_file_tracking AS
    SELECT now() AS spill_time,*
    FROM aurora_stat_file()
    WHERE filename LIKE '%spill%';
```

論理レプリケーション中にスピルファイルがどのように使用されるかを確認するには、パブリッシャーとサブスクライバを設定し、シンプルなレプリケーションを開始します。詳細については、「Aurora PostgreSQL DB クラスターの論理レプリケーションの設定」を参照してください。レプリケーションが進行中であれば、次のように aurora_stat_file() スピルファイル関数から結果セットをキャプチャするジョブを作成できます。

```
INSERT INTO spill_file_tracking
  SELECT now(),*
  FROM aurora_stat_file()
  WHERE filename LIKE '%spill%';
```

次の psql コマンドを使用して、ジョブを 1 秒に 1 回実行します。

```
\watch 0.5
```

ジョブの実行中に、別の psql セッションからライターインスタンスに接続します。次の一連のステートメントを使用すると、メモリ設定を超えるワークロードを実行し、Aurora PostgreSQL ではスピルファイルを作成します。

```
labdb=> CREATE TABLE my_table (a int PRIMARY KEY, b int);
CREATE TABLE
labdb=> INSERT INTO my_table SELECT x,x FROM generate_series(0,10000000) x;
INSERT 0 10000001
labdb=> UPDATE my_table SET b=b+1;
UPDATE 10000001
```

このステートメントには完了するのに数分かかります。終了したら、Ctrl キーと C キーを同時に押して、モニタリング機能を停止します。次に、次のコマンドを使用して、Aurora PostgreSQL DB クラスターのスピルファイルの使用状況に関する情報を保存するテーブルを作成します。

```
SELECT spill_time, split_part (filename, '/', 2)
    AS slot_name, count(1)
    AS spills, sum(used_bytes)
    AS slot_total_bytes, pg_size_pretty(sum(used_bytes))
    AS slot_total_size FROM spill_file_tracking
  GROUP BY 1,2 ORDER BY 1;
                  spill_time | slot_name             | spills | slot_total_bytes |
 slot_total_size
-------------------------------+-----------------------+--------+------------------
+-----------------
2022-04-15 13:42:52.528272+00 | replication_slot_name | 1      | 142352280        | 136
 MB
2022-04-15 14:11:33.962216+00 | replication_slot_name | 4      | 467637996        | 446
 MB
2022-04-15 14:12:00.997636+00 | replication_slot_name | 4      | 569409176        | 543
 MB
2022-04-15 14:12:03.030245+00 | replication_slot_name | 4      | 569409176        | 543
 MB
2022-04-15 14:12:05.059761+00 | replication_slot_name | 5      | 618410996        | 590
 MB
2022-04-15 14:12:07.22905+00  | replication_slot_name | 5      | 640585316        | 611
 MB
(6 rows)
```

この例を実行すると、出力には 5 つのスピルファイルが作成され、611 MB のメモリを使用したことが示されています。ディスクへの書き込みを避けるには、`logical_decoding_work_mem` パラメータを次の最大メモリサイズの 1,024 に設定することをお勧めします。

Amazon CloudWatch メトリクスを使用して Aurora PostgreSQL のリソース使用状況を分析する

Aurora はメトリクスデータを 1 分間隔で CloudWatch に自動的に送信します。CloudWatch メトリクスを使用して Aurora PostgreSQL のリソース使用状況を分析することができます。メトリクスを使用して、ネットワークのスループットとネットワーク使用量を評価できます。

CloudWatch によるネットワークスループットの評価

システム使用量がインスタンスタイプのリソース限界に近づくと、処理が遅くなることがあります。CloudWatch の [Logs Insights] (ログのインサイト) を使用して、ストレージリソースの使用状況をモニタリングし、十分なリソースが使用可能であることを確認できます。必要に応じて、DB インスタンスをより大きなインスタンスクラスに変更できます。

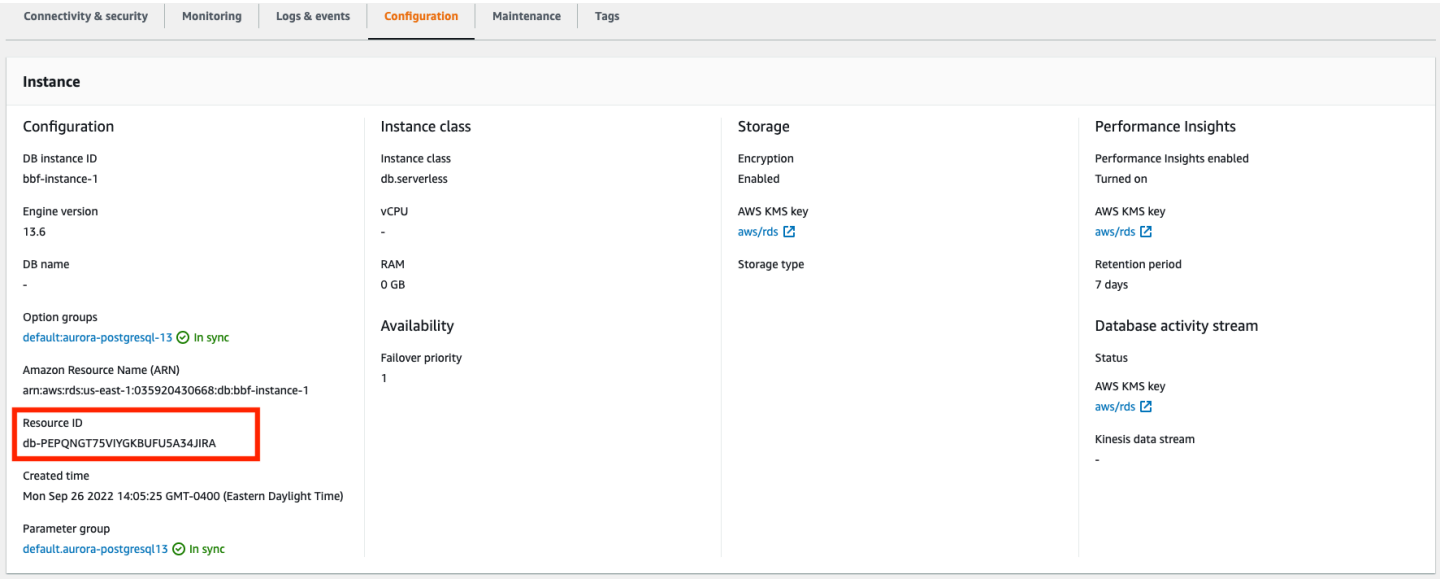
Aurora ストレージの処理は、次の理由で遅くなることがあります。

- クライアントと DB インスタンス間のネットワーク帯域幅が不十分です。
- ストレージサブシステムへのネットワーク帯域幅が不十分です。
- ご使用のインスタンスタイプでは大きいワークロード。

CloudWatch の [Logs Insights] (ログのインサイト) にクエリを実行して、Aurora ストレージリソースの使用状況のグラフを生成し、リソースを監視できます。グラフには CPU 使用率とメトリクスが表示され、より大きなインスタンスサイズにスケールアップするかどうかの判断に役立ちます。CloudWatch [Logs Insights] (ログのインサイト) のクエリ構文の詳細については、「CloudWatch Logs Insights クエリ構文」を参照してください。

CloudWatch を使用するには、Aurora PostgreSQL ログファイルを CloudWatch にエクスポートする必要があります。既存のクラスターを変更して、ログを CloudWatch にエクスポートすることもできます。 ログを CloudWatch にエクスポートする方法については、「Amazon CloudWatch にログを発行するオプションをオンにする」を参照してください。

CloudWatch [Logs Insights] (ログのインサイト) にクエリを実行するには、DB インスタンスの [Resource ID] (リソース ID) が必要です。[Resource ID] (リソース ID) は、コンソールの [Configuration] (設定) タブで確認できます。



ログファイルにリソースストレージメトリクスを照会するには:

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。

   CloudWatch 概要のホームページが表示されます。

2. 必要に応じて AWS リージョン を変更します。ナビゲーションバーで、AWS リソースがある AWS リージョン を選択します。詳細については、「リージョンとエンドポイント」を参照してください。

3. ナビゲーションペインで、[Logs] (ログ)、[Logs Insights] (ログのインサイト) の順に選択します。

   [Logs Insights] (ログのインサイト) ページが表示されます。

4. ドロップダウンリストから、分析するログファイルを選択します。

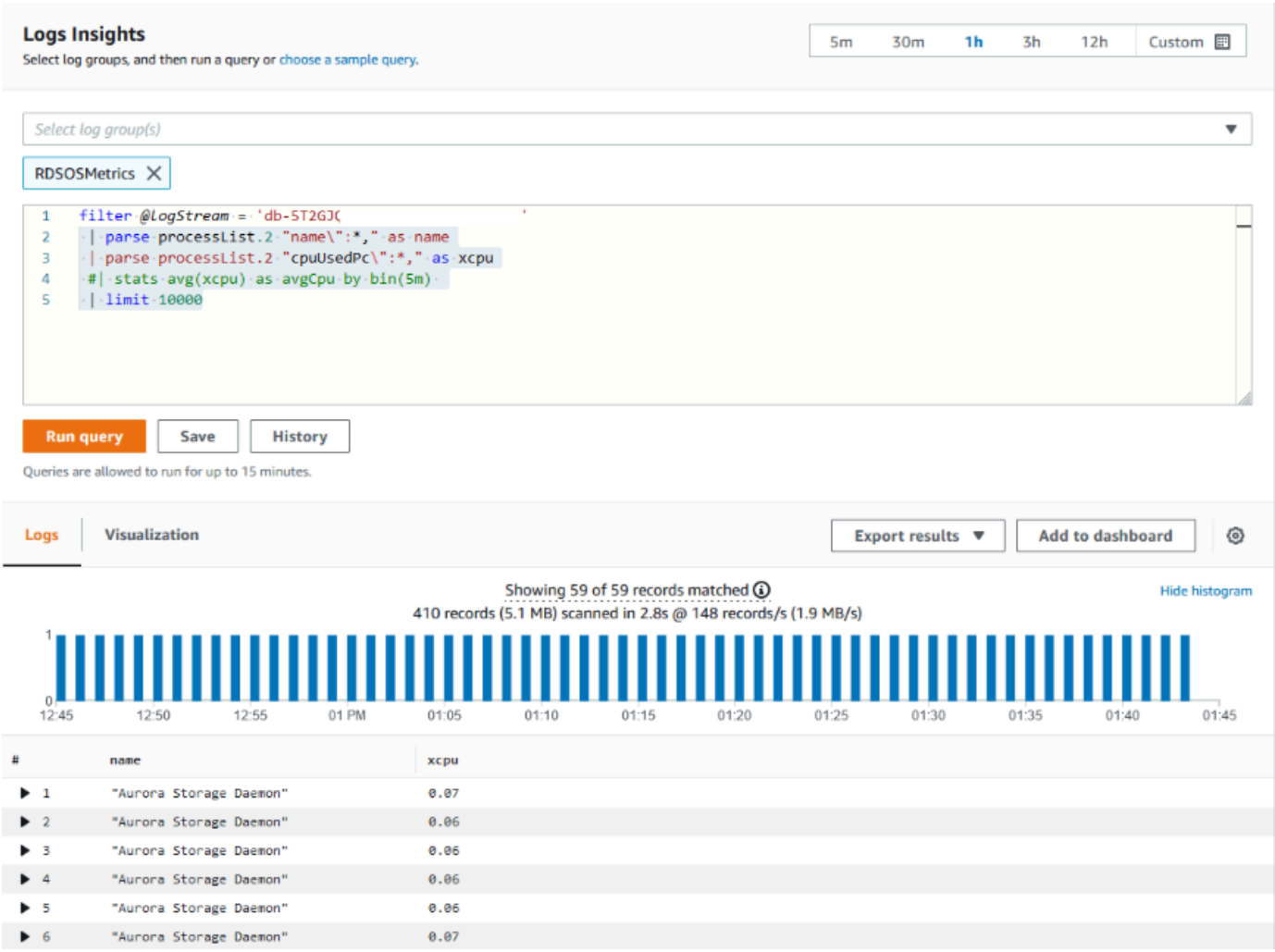
5. フィールドに次のクエリを入力します。<resource ID> を DB クラスターのリソース ID に置き換えてください。

   ```
   filter @logStream = <resource ID> | parse @message "\"Aurora Storage Daemon\"*memoryUsedPc\":*,\"cpuUsedPc\":*," as a,memoryUsedPc,cpuUsedPc | display memoryUsedPc,cpuUsedPc #| stats avg(xcpu) as avgCpu by bin(5m) | limit 10000
   ```

6. [Run query] (クエリを実行) をクリックします。

   ストレージ使用率グラフが表示されます。

次の画像は、[Logs Insights] (ログのインサイト) ページとグラフ表示を示しています。



CloudWatch メトリクスで DB インスタンスの使用状況を評価する

CloudWatch メトリクスを使用して、インスタンスのスループットを監視し、インスタンスクラスがアプリケーションに十分なリソースを提供しているかどうかを確認できます。DB インスタンスクラスの制限については、Aurora 用の DB インスタンスクラスのハードウェア仕様 に移動し、DB インスタンスクラスの仕様を確認して、ネットワークパフォーマンスを確認してください。

DB インスタンスの使用量がインスタンスクラスの限界に近い場合、パフォーマンスが低下し始める可能性があります。CloudWatch メトリクスによってこの状況を確認できるので、より大きなインスタンスクラスへの手動スケールアップを計画できます。

以下の CloudWatch メトリクス値を組み合わせて、インスタンスクラスの限界に近づいているかどうかを確認します。

- NetworkThroughput - Aurora DB クラスター内の各インスタンスについて、各クライアントによって送受信されたネットワークスループットの量。このスループット値には、DB クラスターとクラスターボリューム内のインスタンス間のネットワークトラフィックは含まれません。
- StorageNetworkThroughput – Aurora DB クラスター内の各インスタンスによって Aurora ストレージサブシステムとの間で送受信されたネットワークスループットの量。

NetworkThroughput を StorageNetworkThroughput に加えると、Aurora DB クラスター内の各インスタンスによって Aurora ストレージサブシステムとの間で送受信されたネットワークスループットになります。インスタンスのインスタンスクラス限界は、この 2 つのメトリクスの合計よりも大きい必要があります。

以下のメトリクスを使用して、送受信時のクライアントアプリケーションからのネットワークトラフィックの詳細を確認できます。

- NetworkReceiveThroughput – Aurora PostgreSQL DB クラスター内の各インスタンスがクライアントから受信したネットワークスループットの量。 DB クラスターとクラスターボリューム内のインスタンス間のネットワークトラフィックは、このスループットに含まれません。
- NetworkTransmitThroughput – Aurora DB クラスター内の各インスタンスが各クライアントに送信したネットワークスループットの量。 DB クラスターとクラスターボリューム内のインスタンス間のネットワークトラフィックは、このスループットに含まれません。
- StorageNetworkReceiveThroughput – DB クラスター内の各インスタンスが Aurora ストレージサブシステムから受信したネットワークスループットの量。
- StorageNetworkTransmitThroughput – DB クラスター内の各インスタンスが Aurora ストレージサブシステムに送信したネットワークスループットの量。

これらすべてのメトリクスを合計して、ネットワーク使用量とインスタンスクラスの限界との比較を評価します。インスタンスクラス限界は、これらのメトリクスの合計よりも大きい必要があります。

ネットワーク限界とストレージの CPU 使用率は相互に関係しています。ネットワークのスループットが増加すると、CPU 使用率も増加します。CPU とネットワークの使用状況を監視すると、リソースがどのくらい枯渇しているのか、またなぜ枯渇しているのかについての情報が得られます。

ネットワークの使用量を最小限に抑えるには、次のことを検討してください。

- より大きなインスタンスクラスを使用します。
- `pg_partman` パーティショニング戦略を使用します。

- 書き込みリクエストをバッチに分割して、トランザクション全体を減らします。
- 読み取り専用ワークロードを読み取り専用インスタンスに転送します。
- 未使用のインデックスをすべて削除します。
- 膨張したオブジェクトと VACUUM をチェックします。膨張がひどい場合は、PostgreSQL 拡張機能 pg_repack を使用します。pg_repack の詳細については、「最小限のロックで PostgreSQL データベース内のテーブルを再編成する」を参照してください。

論理レプリケーションを使用して Aurora PostgreSQL のメジャーバージョンアップグレードを実行する

論理レプリケーションと Aurora 高速クローニングを使用すると、変更するデータを新しいメジャーバージョンデータベースに徐々に移行しながら、Aurora PostgreSQL データベースの現行バージョンを使用するメジャーバージョンアップグレードを実行できます。このダウンタイムの少ないアップグレードプロセスは、ブルー/グリーンアップグレードと呼ばれます。データベースの現在のバージョンは「ブルー」環境、新しいデータベースバージョンは「グリーン」環境と呼ばれます。

Aurora 高速クローニングでは、ソースデータベースのスナップショットを取得して既存のデータをすべてロードします。高速クローニングでは、Aurora ストレージレイヤー上に構築された Copy-on-Write を使用します。これにより、データベースのクローンを短時間で作成できます。この方法は、大規模なデータベースにアップグレードする場合に非常に効果的です。

PostgreSQL の論理レプリケーションでは、新しいバージョンの PostgreSQL に移行するまで、最初のインスタンスから同時実行される新しいインスタンスへのデータ変更が追跡され、転送されます。論理的なレプリケーションは、発行およびサブスクライブモデルを使用します。Aurora PostgreSQL の論理レプリケーションの詳細については、「Amazon Aurora PostgreSQL でのレプリケーション」を参照してください。

> **Tip**
>
> マネージド Amazon RDS ブルー/グリーンデプロイ機能を使用することで、メジャーバージョンのアップグレードに必要なダウンタイムを最小限に抑えることができます。詳細については、「データベース更新のために Amazon RDS ブルー/グリーンデプロイを使用する」を参照してください。

トピック

- 要件
- 制限事項
- パラメータ値の設定と確認
- Aurora PostgreSQL を新しいメジャーバージョンにアップグレードする
- アップグレード後のタスクの実行

要件

このダウンタイムの少ないアップグレードプロセスを実行するには、次の要件を満たす必要があります。

- rds_superuser 権限が付与されている必要があります。
- アップグレードする Aurora PostgreSQL DB クラスターは、論理レプリケーションを使用してメジャーバージョンアップグレードを実行できるサポートバージョンを実行している必要があります。マイナーバージョンの更新とパッチを DB クラスターに適用してください。この方法で使用される `aurora_volume_logical_start_lsn` 関数は、以下のバージョンの Aurora PostgreSQL でサポートされています。
  - バージョン 15 の 15.2 以降
  - バージョン 14 の中の 14.3 以降
  - バージョン 13 の 13.6 以降
  - バージョン 12 の 12.10 以降
  - 11.15 以上の 11 バージョン
  - 10.20 以上の 10 バージョン

  `aurora_volume_logical_start_lsn` 関数の詳細については、「aurora_volume_logical_start_lsn」を参照してください。
- すべてのテーブルにプライマリキーがあるか、PostgreSQL の IDENTITY 列が含まれている必要があります。
- 新旧両方の 2 つの Aurora PostgreSQL DB クラスター間でインバウンドおよびアウトバウンドのアクセスを許可するように VPC のセキュリティグループを設定します。特定の Classless Inter-Domain Routing (CIDR) 範囲、または VPC あるいはピア VPC (VPC ピアリング接続が必要) の別のセキュリティグループにアクセス権を付与できます。

> **Note**
>
> 実行中の論理レプリケーションシナリオの設定と管理に必要な権限の詳細については、「PostgreSQL core documentation」を参照してください。

制限事項

Aurora PostgreSQL DB クラスターでダウンタイムの少ないアップグレードを実行して新しいメジャーバージョンにアップグレードする場合、ネイティブの PostgreSQL 論理レプリケーション機能を使用します。機能と制限は PostgreSQL 論理的レプリケーションと同じです。詳細については、「PostgreSQL 論理的レプリケーション」を参照してください。

- データ定義言語 (DDL) コマンドはレプリケートされません。
- レプリケーションは、ライブデータベースのスキーマの変更をサポートしていません。スキーマは、クローニング処理中に元の形式で再作成されます。クローニング後にスキーマを変更しても、アップグレードを完了する前は、アップグレードされたインスタンスに反映されません。
- ラージオブジェクトはレプリケートされませんが、通常のテーブルにデータを保存できます。
- レプリケーションは、パーティションテーブルなどのテーブルでのみサポートされます。ビュー、マテリアライズドビュー、外部テーブルなどの他の種類のリレーションへのレプリケーションは、サポートされていません。
- シーケンスデータはレプリケートされないため、フェイルオーバー後に手動で更新する必要があります。

> **Note**
>
> このアップグレードでは、自動スクリプトはサポートされていません。すべての手順は手動で実行する必要があります。

パラメータ値の設定と確認

アップグレード前に、Aurora PostgreSQL DB クラスターのライターインスタンスをパブリケーションサーバーとして機能するように設定します。インスタンスは、次の設定のカスタム DB クラスターパラメータグループを使用する必要があります。

- rds.logical_replication – このパラメータは 1 に設定します。rds.logical_replication パラメータは、スタンドアロンの PostgreSQL サーバーの wal_level パラメータや、ログ先行書き込みファイル管理を制御するその他のパラメータと同じ目的を果たします。
- max_replication_slots – このパラメータの値を、作成する予定のサブスクリプションの総数に設定します。AWS DMS を使用している場合、このパラメータを、この DB クラスターからの変更データキャプチャに使用する予定の AWS DMS タスクの数に設定します。
- max_wal_senders— 同時接続数に加えて、いくつかの追加接続数を設定して、管理タスクや新しいセッションに使用できるようにします。AWS DMS を使用する場合、max_wal_senders の数は、同時セッション数と、任意の時点で動作している可能性のある AWS DMS に タスクの数の合計に等しい必要があります。
- max_logical_replication_workers — 予想される論理レプリケーションワーカーとテーブル同期ワーカーの数に設定します。通常、レプリケーションワーカーの数は max_wal_senders に使用するのと同じ値に設定すると、安全です。ワーカーは、サーバーに割り当てられたバックグラウンドプロセス (max_worker_processes) のプールから取得されます。
- max_worker_processes — サーバーのバックグラウンドプロセス数に設定します。この数は、レプリケーション、自動バキュームプロセス、および同時に行われる可能性があるその他のメンテナンスプロセスにワーカーを割り当てるのに十分な大きさである必要があります。

Aurora PostgreSQL の新しいバージョンにアップグレードする場合、以前のバージョンのパラメータグループで変更したパラメータを複製する必要があります。このようなパラメータは、アップグレードされたバージョンに適用されます。pg_settings テーブルをクエリしてパラメータ設定のリストを取得し、新しい Aurora PostgreSQL DB クラスターで再作成できます。

例えば、レプリケーションパラメータの設定を取得するには、次のクエリを実行します。

```
SELECT name, setting FROM pg_settings WHERE name in
('rds.logical_replication', 'max_replication_slots',
'max_wal_senders', 'max_logical_replication_workers',
'max_worker_processes');
```

Aurora PostgreSQL を新しいメジャーバージョンにアップグレードする

パブリッシャーを準備するには (ブルー)

1. 以下の例では、ソースライターインスタンス (ブルー) は、PostgreSQL バージョン 11.15 が実行中の Aurora PostgreSQL DB クラスターです。これは、レプリケーションシナリオのパブリケー

ションノードです。このデモンストレーションでは、一連の値を保持しているサンプルテーブルをソースライターインスタンスがホストします。

```
CREATE TABLE my_table (a int PRIMARY KEY);
INSERT INTO my_table VALUES (generate_series(1,100));
```

2. ソースインスタンス上にパブリケーションを作成するには、psql (PostgreSQL の CLI) または任意のクライアントを使用してインスタンスのライターノードに接続します。各データベースに次のコマンドを入力します。

```
CREATE PUBLICATION publication_name FOR ALL TABLES;
```

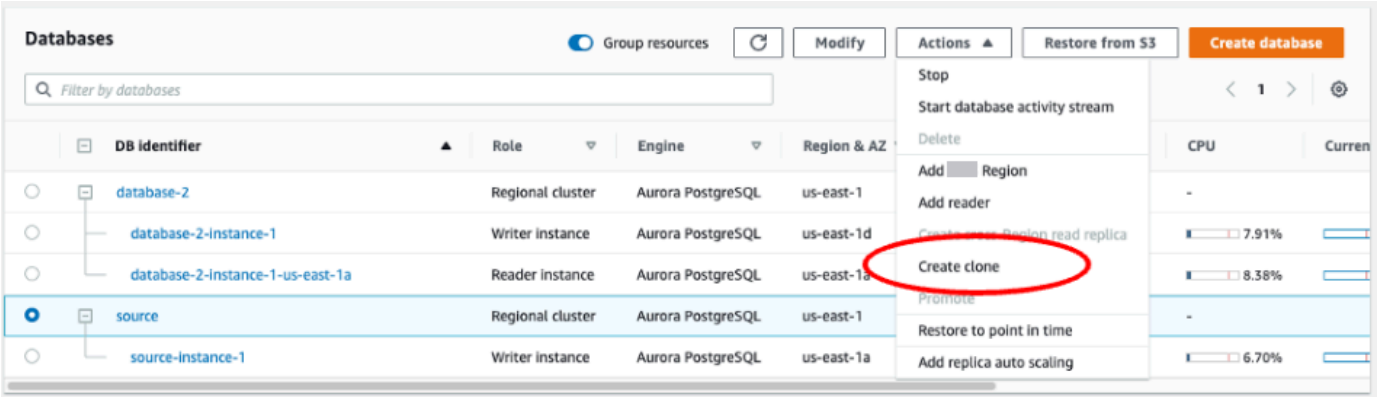
publication_name には、パブリケーションの名前を指定します。

3. また、インスタンス上にレプリケーションスロットも作成する必要があります。次のコマンドを実行すると、レプリケーションスロットが作成され、pgoutput 論理デコーディングプラグインがロードされます。プラグインがロードされると、先行書き込みログ (WAL) から読み込まれたコンテンツが論理レプリケーションプロトコルに変更され、パブリケーション仕様に従ってデータがフィルタリングされます。

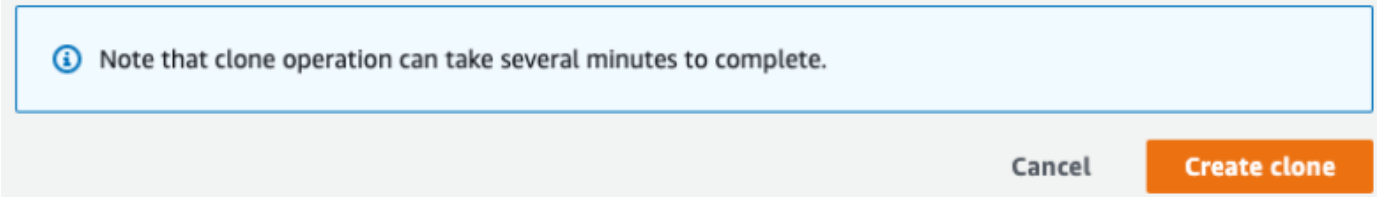
```
SELECT pg_create_logical_replication_slot('replication_slot_name', 'pgoutput');
```

パブリッシャーのクローンを作成するには

1. Amazon RDS コンソールを使用して、ソースインスタンスのクローンを作成します。Amazon RDS コンソールでインスタンス名をハイライトし、[Actions] (アクション) メニューで [Create clone] (クローンの作成) を選択します。



2. インスタンスの一意の名前を指定します。設定のほとんどはソースインスタンスのデフォルトです。新しいインスタンスに必要な変更を加えたら、[Create clone] (クローンの作成) を選択します。



3. ターゲットインスタンスの起動中、ライターノードの [Status] (ステータス) 列には、[Creating] (作成中) と表示されます。インスタンスが使用可能になると、ステータスは [Available] (使用可能) に変わります。

クローンをアップグレード用に準備するには

1. クローンはデプロイモデルの「グリーン」インスタンスです。このクローンが、レプリケーションサブスクリプションノードのホストになります。ノードが使用可能になったら、psql に接続し、新しいライターノードにクエリを実行してログシーケンス番号 (LSN) を取得します。LSN は WAL ストリーム内のレコードの先頭を識別します。

```
SELECT aurora_volume_logical_start_lsn();
```

2. クエリからの応答には、LSN 番号が含まれています。この番号はプロセスの後半で必要となるため、書き留めておいてください。

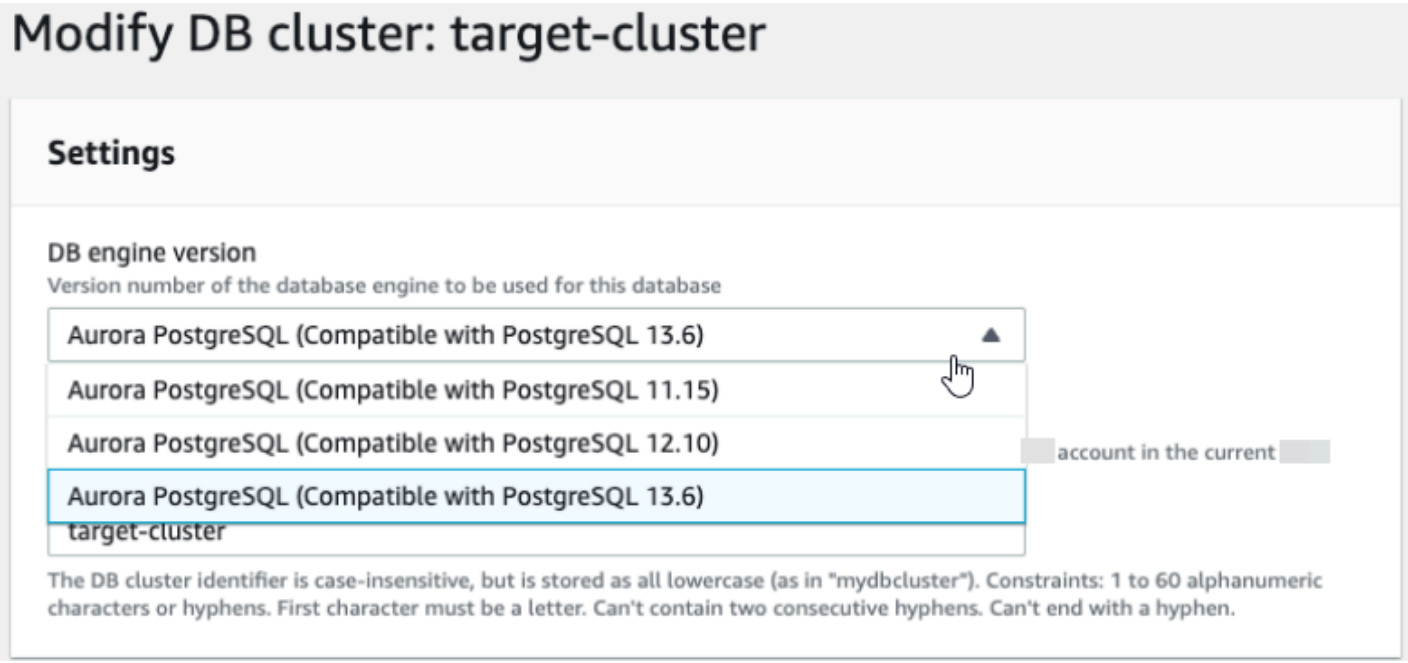
```
postgres=> SELECT aurora_volume_logical_start_lsn();
aurora_volume_logical_start_lsn
---------------
0/402E2F0
(1 row)
```

3. クローンをアップグレードする前に、クローンのレプリケーションスロットを削除してください。

```
SELECT pg_drop_replication_slot('replication_slot_name');
```

クラスターを新しいメジャーバージョンにアップグレードするには

- プロバイダーノードのクローンを作成した後、Amazon RDS コンソールを使用して、サブスクリプションノードのメジャーバージョンアップグレードを開始します。RDS コンソールでインスタンス名をハイライトし、[Modify] (変更) ボタンを選択します。更新されたバージョンと更新されたパラメータグループを選択し、すぐに設定を適用してターゲットインスタンスをアップグレードします。



- CLI を使用してアップグレードを実行することもできます。

```
aws rds modify-db-cluster –db-cluster-identifier $TARGET_Aurora_ID –engine-version
 13.6 –allow-major-version-upgrade –apply-immediately
```

サブスクライバーを準備するには (グリーン)

1. アップグレード後にクローンが使用可能になったら、psql に接続して、サブスクリプションを定義します。これを行うには、CREATE SUBSCRIPTION コマンドで次のオプションを指定する必要があります。

   - subscription_name – サブスクリプションの名前。
   - admin_user_name — rds_superuser 権限を持つ管理ユーザーの名前。
   - admin_user_password – 管理ユーザーに関連付けられているパスワード。
   - source_instance_URL — パブリケーションサーバーインスタンスの URL。

- database — サブスクリプションサーバーが接続するデータベース。
- publication_name — パブリケーションサーバーの名前。
- replication_slot_name – レプリケーションスロットの名前。

```
CREATE SUBSCRIPTION subscription_name
CONNECTION 'postgres://admin_user_name:admin_user_password@source_instance_URL/
database' PUBLICATION publication_name
WITH (copy_data = false, create_slot = false, enabled = false, connect = true,
 slot_name = 'replication_slot_name');
```

2. サブスクリプションを作成したら、pg_replication_origin ビューにクエリして、レプリケーションオリジンの識別子である roname 値を取得します。各インスタンスには、roname が 1 つあります。

```
SELECT * FROM pg_replication_origin;
```

例:

```
postgres=>
SELECT * FROM pg_replication_origin;

roident | roname
---------+----------
1 | pg_24586
```

3. パブリケーションノードの以前のクエリで保存した LSN と、サブスクリプションノード [INSTANCE] から返された roname をコマンドに入力します。このコマンドは、pg_replication_origin_advance 関数を使用してレプリケーションのログシーケンスの開始点を指定します。

```
SELECT pg_replication_origin_advance('roname', 'log_sequence_number');
```

roname は、pg_replication_origin ビューによって返される識別子です。

log_sequence_number は、aurora_volume_logical_start_lsn 関数の以前のクエリによって返された値です。

4. 次に、ALTER SUBSCRIPTION... ENABLE 句を使用して論理レプリケーションをオンにします。

```
ALTER SUBSCRIPTION subscription_name ENABLE;
```

5. この時点で、レプリケーションが機能していることを確認できます。パブリケーションインスタンスに値を追加して、その値がサブスクリプションノードにレプリケートされることを確認します。

   次に、次のコマンドを使用して、パブリケーションノードのレプリケーションラグをモニタリングします。

```
SELECT now() AS CURRENT_TIME, slot_name, active, active_pid,
 pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_lsn(),
confirmed_flush_lsn)) AS diff_size, pg_wal_lsn_diff(pg_current_wal_lsn(),
confirmed_flush_lsn) AS diff_bytes FROM pg_replication_slots WHERE slot_type =
 'logical';
```

   例:

```
postgres=> SELECT now() AS CURRENT_TIME, slot_name, active, active_pid,
 pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_lsn(),
confirmed_flush_lsn)) AS diff_size, pg_wal_lsn_diff(pg_current_wal_lsn(),
 confirmed_flush_lsn) AS diff_bytes FROM pg_replication_slots WHERE slot_type =
 'logical';

current_time                   | slot_name             | active | active_pid |
 diff_size | diff_bytes
-------------------------------+-----------------------+--------+------------
+-----------+------------
2022-04-13 15:11:00.243401+00  | replication_slot_name | t      | 21854      | 136
 bytes | 136
(1 row)
```

   diff_size および diff_bytes 値を使用して、レプリケーションラグをモニタリングできます。これらの値が 0 に達すると、レプリカがソース DB インスタンスに追いついています。

アップグレード後のタスクの実行

アップグレードが完了すると、コンソールのダッシュボードの [Status] (ステータス) 列にインスタンスのステータスが [Available] (使用可能) と表示されます。新しいインスタンスで、次の操作を実行することをお勧めします。

- ライターノードを指すようにアプリケーションをリダイレクトします。
- リーダーノードを追加してケースロードを管理し、ライターノードで問題が発生した場合に備えて高可用性を持たせます。
- Aurora PostgreSQL DB クラスターでは、オペレーティングシステムの更新が必要になる場合があります。これらのアップデートには glibc ライブラリの新しいバージョンが含まれることがあります。このような更新の際は、「Aurora PostgreSQL でサポートされる照合」で説明されているガイドラインに従うことをお勧めします。
- 新しいインスタンスのユーザー権限を更新して、アクセスを確実にします。

新しいインスタンスでアプリケーションとデータをテストしたら、最初のインスタンスを削除する前に最終バックアップを作成することをお勧めします。Aurora ホストにおける論理レプリケーションの使用の詳細については、「Aurora PostgreSQL DB クラスターの論理レプリケーションの設定」を参照してください。

ストレージ問題のトラブルシューティング

ソートまたはインデックス作成操作に必要な作業メモリ量が `work_mem` パラメータによって割り当てられた量を超える場合、Aurora PostgreSQL はこの超過したデータを一時ディスクファイルに書き込みます。Aurora PostgreSQL は、データを書き込む際、エラーやメッセージログを保管する場合に使用するのと同じストレージ領域、つまりローカルストレージを使用します。Aurora PostgreSQL DB クラスターの各インスタンスには、使用できるローカルストレージの量が指定されています。ストレージの量は DB インスタンスクラスに基づいています。ローカルストレージの量を増やすには、より大きな DB インスタンスクラスを使用するようにインスタンスを変更する必要があります。DB インスタンスクラスの仕様については、「Aurora 用の DB インスタンスクラスのハードウェア仕様」を参照してください。

Aurora PostgreSQL DB クラスターのローカルストレージスペースをモニタリングするには、Amazon CloudWatch メトリックスで `FreeLocalStorage` を監視します。このメトリクスでは、 Aurora DB クラスターの各 DB インスタンスの一時テーブルとログで使用できるストレージの量が報告されます。詳細については、「Amazon CloudWatch を使用した Amazon Aurora メトリクスのモニタリング」を参照してください。

ソート、インデックス作成、およびグループ化の操作は作業メモリで開始されますが、多くの場合、ローカルストレージにオフロードされる必要があります。このような操作が原因で Aurora PostgreSQL DB クラスターのローカルストレージが不足した場合、次のいずれかのアクションを実行することで問題を解決できます。

- 作業メモリの量を増やします。これにより、ローカルストレージを使用する必要が少なくなります。デフォルトでは、PostgreSQL はソート、グループ化、インデックス作成の各操作に 4 MB を割り当てます。Aurora PostgreSQL DB クラスターのライターインスタンスの現在の作業メモリを確認するには、psql を使用してインスタンスに接続し、次のコマンドを実行します。

  ```
  postgres=> SHOW work_mem;
  work_mem
  ----------
   4MB
  (1 row)
  ```

  次のように、ソート、グループ化、その他の操作の前のセッションレベルの作業メモリを増やすことができます。

  ```
  SET work_mem TO '1 GB';
  ```

  作業メモリの詳細については、PostgreSQL ドキュメントの「リソース消費」を参照してください。
- ログの保存期間を変更して、ログの保存期間を短くします。この方法の詳細は、「Aurora PostgreSQL データベースログファイル」を参照してください。

Aurora PostgreSQL DB クラスターが 40 TB より大きい場合は、db.t2、db.t3、または db.t4g インスタンスクラスを使用しないでください。T DB インスタンスクラスは、開発サーバーおよびテストサーバー、またはその他の本稼働以外のサーバーにのみ使用することをお勧めします。詳細については、「DB インスタンスクラスタイプ」を参照してください。

Amazon Aurora PostgreSQL でのレプリケーション

以下では Amazon Aurora PostgreSQL でのレプリケーションと、レプリケーションのモニタリングについて説明します。

トピック

- Aurora レプリカの使用
- Aurora レプリカの読み取り可用性の向上
- Aurora PostgreSQL レプリケーションのモニタリング
- Aurora での PostgreSQL 論理的なレプリケーションの使用

Aurora レプリカの使用

Aurora レプリカは Aurora DB クラスター内の独立したエンドポイントで、読み取りオペレーションのスケーリングと可用性向上に最適です。Aurora DB クラスターには、Aurora DB クラスターがある AWS リージョンのアベイラビリティーゾーン全体に配置された、最大 15 の Aurora レプリカを含めることができます。

DB クラスターボリュームは、DB クラスターのデータの複数のコピーから構成されます。ただし、クラスターボリューム内のデータは、DB クラスターのプライマリ書き込み DB インスタンスおよび Aurora レプリカに対して単一の論理的なボリュームとして表されます。Aurora レプリカの詳細については、「Aurora レプリカ」を参照してください。

Aurora レプリカは、クラスターボリュームに対する読み取りオペレーション専用であるため、読み取りのスケーリングに適しています。書き込み DB インスタンスは、書き込みオペレーションを管理します。クラスターボリュームは、Aurora PostgreSQL DB クラスター内のすべてのインスタンス間で共有されます。したがって、各 Aurora レプリカでデータのコピーをレプリケートする追加作業は必要ありません。

Aurora PostgreSQL では、Aurora レプリカが削除されるとそのインスタンスエンドポイントは直ちに削除され、Aurora レプリカもリーダーエンドポイントから削除されます。削除中の Aurora レプリカで実行されているステートメントがある場合は、削除までに 3 分の猶予期間があります。既存のステートメントは、猶予期間中に適切に終了する場合があります。猶予期間が終了すると、Aurora レプリカはシャットダウンし、削除されます。

Aurora PostgreSQL DB クラスターでは、Aurora グローバルデータベースを使用し、異なる AWS リージョンの Aurora レプリカがサポートしています。詳細については、「Amazon Aurora Global Database の使用」を参照してください。

> **Note**
>
> 読み取り可用性機能が向上した DB クラスターの Aurora レプリカを再起動するには、手動で実行する必要があります。この機能の前に作成された DB クラスターでは、ライター DB インスタンスを再起動すると、Aurora レプリカが自動的に再起動されます。自動再起動で再確立されるエントリポイントにより、DB クラスター全体での読み取り/書き込みの一貫性が保証されます。

Aurora レプリカの読み取り可用性の向上

Aurora PostgreSQL は、ライター DB インスタンスが再起動したときや Aurora レプリカが書き込みトラフィックに追いつけなくなったときに読み取りリクエストを継続的に処理することで、DB クラスターの読み取り可用性を向上させます。

読み取り可用性機能は、Aurora PostgreSQL の次のバージョンでデフォルトで使用できます。

- バージョン 15 の 15.2 以降
- バージョン 14 の 14.7 以降
- バージョン 13 の 13.10 以降
- バージョン 12 の 12.14 以降

この起動前にこれらのいずれかのバージョンで作成された DB クラスターの読み取り可用性機能を使用するには、DB クラスターのライターインスタンスを再起動します。

Aurora PostgreSQL DB クラスターの静的パラメーターを変更する場合、パラメーターの変更を有効にするにはライターインスタンスを再起動する必要があります。たとえば、`shared_buffers` の値を設定するときは、ライターインスタンスを再起動する必要があります。Aurora レプリカの可用性が向上したことで、DB クラスターは再起動中も読み取りの可用性を維持し、ライターインスタンスへの変更による影響が軽減されます。リーダーインスタンスは再起動せず、読み取りリクエストに応答し続けます。静的パラメーターの変更を適用するには、個々のリーダーインスタンスを再起動します。

Aurora PostgreSQL DB クラスターの Aurora レプリカは、ライターと再接続した後すぐにインメモリデータベースの状態に回復することで、ライターの再起動、フェイルオーバー、低速レプリケーション、ネットワークの問題などのレプリケーションエラーから回復できます。このアプローチにより、Aurora レプリカインスタンスは、クライアントデータベースを引き続き使用しながら、最新のストレージアップデートとの整合性を保つことができます。

レプリケーション復旧と競合する進行中のトランザクションはエラーを受け取る可能性がありますが、リーダーがライターに追いついたら、クライアントはこれらのトランザクションを再試行できます。

Aurora レプリカのモニタリング

ライターの接続解除からのリカバリ時に Aurora レプリカをモニタリングできます。以下のメトリクスを使用して、リーダーインスタンスに関する最新情報を確認したり、処理中の読み取り専用トランザクションを追跡したりできます。

- aurora_replica_status 関数は、リーダーインスタンスがまだ接続状態で最新の情報を返すように更新されています。クエリが実行された DB インスタンスに対応する行に対しては、aurora_replica_status の最終更新タイムスタンプは常に空となります。これは、リーダーインスタンスに最新のデータがあることを示します。
- Aurora レプリカがライターインスタンスとの接続を切断して再接続すると、次のデータベースイベントが発生します。

  ```
  Read replica has been disconnected from the writer instance and
  reconnected.
  ```

- リカバリの競合が原因で読み取り専用クエリがキャンセルされると、データベースエラーログに次のエラーメッセージが表示される可能性があります。

  ```
  Canceling statement due to conflict with recovery.
  ```

制限事項

可用性が改善された Aurora レプリカには、次の制限が適用されます。

- セカンダリ AWS リージョン の Global DB Aurora レプリカはサポートされていません。
- Aurora レプリカは、オンラインレプリケーションリカバリが既に進行中の状態で再起動される場合は、オンラインレプリケーションリカバリをサポートしません。
- Aurora レプリカは、DB インスタンスがトランザクション ID のラップアラウンドに近づくと再起動します。トランザクション ID 循環の詳細については、「トランザクション ID 循環失敗の防止」を参照してください。
- Aurora レプリカは、特定の状況でレプリケーションプロセスがブロックされたときに再起動できます。

Aurora PostgreSQL レプリケーションのモニタリング

読み取りのスケーリングと高可用性は最短遅延時間に左右されます。Amazon CloudWatch ReplicaLag メトリクスをモニタリングすることにより、Aurora レプリカが Aurora PostgreSQL DB クラスターの書き込み DB インスタンスからどれくらい遅延しているかをモニタリングできます。Aurora レプリカは、書き込み DB インスタンスと同じクラスターボリュームから読み取るため、ReplicaLag メトリクスの意味は Aurora PostgreSQL DB クラスターの場合とは異なります。Aurora レプリカの ReplicaLag メトリクスは、書き込み DB インスタンスのページキャッシュと比較した場合の Aurora レプリカのページキャッシュの遅延を示します。

RDS インスタンスと CloudWatch メトリックスのモニタリングの詳細については、「Amazon Aurora クラスターでのメトリクスのモニタリング」を参照してください。

Aurora での PostgreSQL 論理的なレプリケーションの使用

Aurora PostgreSQL DB クラスターで PostgreSQL の論理レプリケーション機能を使用することで、データベースインスタンス全体ではなく、個々のテーブルをレプリケートおよび同期できます。論理レプリケーションでは、パブリケーションおよびサブスクリプションモデルを使用して、ソースからの変更を 1 人または複数の受信者にレプリケートします。これは、PostgreSQL のログ先行書き込み (WAL) の変更レコードを使用することで動作します。送信元 (パブリッシャー) は、指定されたテーブルの WAL データを 1 人または複数の受信者 (サブスクライバー) に送信します。これによって変更がレプリケートされ、サブスクライバーのテーブルとパブリッシャーのテーブルの同期を維持できます。パブリッシャーによる一連の変更は、パブリケーションを使用して識別します。サブスクライバーは、パブリッシャーのデータベースとそのパブリケーションへの接続を定義するサブスクリプションを作成することによって変更を取得できます。レプリケーションスロットは、この方式によってサブスクリプションの進行状況を追跡するために使用されるメカニズムです。

Aurora PostgreSQL DB クラスターでは、WAL レコードは Aurora ストレージに保存されます。論理レプリケーションシナリオでパブリッシャーとして機能する Aurora PostgreSQL DB クラスターは、Aurora ストレージから WAL データを読み込み、デコードしてサブスクライバーに送信し、そのインスタンスのテーブルに変更を適用できるようにします。パブリッシャーでは、論理デコーダーを使用してデータをデコードすることでサブスクライバーが使用できるようにします。デフォルトでは、Aurora PostgreSQL DB クラスターはデータを送信する際にネイティブ PostgreSQL `pgoutput` プラグインを使用します。他の論理デコーダーも使用できます。例えば、Aurora PostgreSQL は WAL データを JSON に変換する `wal2json` プラグインもサポートしています。

Aurora PostgreSQL バージョン 14.5、13.8、12.12、11.17 以降、Aurora PostgreSQL は PostgreSQL の論理レプリケーションプロセスをライトスルーキャッシュで強化してパフォーマンスを向上させています。WAL トランザクションログは、ディスク I/O の量、つまり論理デコード中に Aurora ストレージから読み取る量を減らすために、ローカルでバッファにキャッシュされます。Aurora PostgreSQL DB クラスターの論理レプリケーションを使用する場合は常に、ライトスルーキャッシュがデフォルトで使用されます。Aurora には、キャッシュの管理に使用できる機能がいくつか用意されています。詳細については、「Aurora PostgreSQL 論理レプリケーションライトスルーキャッシュの管理」を参照してください。

論理レプリケーションは、現在の Aurora PostgreSQL のすべてのバージョンでサポートされています。詳細については、「Aurora PostgreSQL リリースノート」の「Amazon Aurora PostgreSQL の更新」を参照してください。

> **Note**
>
> PostgreSQL 10 で導入されたネイティブの PostgreSQL 論理レプリケーション機能に加えて、Aurora PostgreSQL は pglogical エクステンションもサポートしています。詳細については、「pglogical を使用してインスタンス間でデータを同期する」を参照してください。

PostgreSQL 論理レプリケーションの詳細については、PostgreSQL ドキュメントの「論理レプリケーション」と「論理デコーディングのコンセプト」を参照してください。

次のトピックでは、Aurora PostgreSQL DB クラスター間の論理レプリケーションの設定方法について説明します。

トピック

- Aurora PostgreSQL DB クラスターの論理レプリケーションの設定
- 論理レプリケーションをオフにする
- Aurora PostgreSQL 論理レプリケーションライトスルーキャッシュの管理
- Aurora PostgreSQL の論理スロットの管理
- 例: Aurora PostgreSQL DB クラスターにおける論理レプリケーションの使用
- 例: Aurora PostgreSQL と AWS Database Migration Service を使用した論理レプリケーション

Aurora PostgreSQL DB クラスターの論理レプリケーションの設定

論理レプリケーションの設定には rds_superuser 権限が必要です。以下で説明する手順のように、Aurora PostgreSQL DB クラスターは、必要なパラメータを設定できるように、カスタム DB クラスターパラメータグループを使用するように設定する必要があります。詳細については、「DB クラスターパラメータグループを使用する」を参照してください。

Aurora PostgreSQL DB クラスターに PostgreSQL 論理レプリケーションを設定するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[Aurora PostgreSQL DB cluster] (Aurora PostgreSQL DB クラスター) を選択します。
3. [Configuration] (設定) タブを開きます。インスタンスの詳細の中から、タイプの DB クラスターパラメーターグループとのパラメーターグループのリンクを見つけます。

4. リンクを選択して、Aurora PostgreSQL DB クラスターに関連するカスタムパラメータを開きます。
5. [Parameters] (パラメータ) 検索フィールドに `rds` と入力し、`rds.logical_replication` パラメータを検索します。このパラメータのデフォルト値は `0` です。つまり、デフォルトではオフになっています。
6. [Edit parameters] (パラメータの編集) を選択してプロパティ値にアクセスし、次にセレクタから `1` を選択して機能をオンにします。予想される使用状況に応じて、次のパラメータ設定の変更が必要になる場合があります。ただし、多くの場合はデフォルト値で十分です。

   - `max_replication_slots` — このパラメータには、最低でも論理レプリケーションのパブリケーションとサブスクリプションの合計予定数以上の値を設定します。AWS DMS を使用している場合、このパラメータには、クラスターからの変更データ取得タスクの予定数に、論理レプリケーションのパブリケーションとサブスクリプションを加えたものと最低でも同じ値を設定する必要があります。
   - `max_wal_senders` および `max_logical_replication_workers` - これらのパラメータには、アクティブにする予定の論理レプリケーションスロットの最小数、または変更データ取得用のアクティブな AWS DMS タスクの数以上の値を設定します。論理レプリケーションスロットを非アクティブにしておくと、バキュームによってテーブルから古いタプルを削除できないため、レプリケーションスロットをモニタリングして、必要に応じて非アクティブなスロットを削除することをお勧めします。
   - `max_worker_processes` – このパラメータには、最低でも `max_logical_replication_workers`、`autovacuum_max_workers`、`max_parallel_worker` の合計と同じ値を設定してください。小規模な DB インスタンスクラスでは、バックグラウンドのワーカープロセスがアプリケーションのワークロードに影響を与える可能性があるため、`max_worker_processes` をデフォルト値よりも高い値を設定する場合はデータベースのパフォーマンスをモニタリングしてください。(デフォルト値は `GREATEST(${DBInstanceVCPU*2},8}` の結果であり、つまり、デフォルトでは 8 または DB インスタンスクラスの CPU 相当量の 2 倍のどちらか大きい方になります)。

   > **Note**
   >
   > ユーザー定義の DB パラメータグループのパラメータ値は変更できますが、デフォルトの DB パラメータグループのパラメータ値を変更することはできません。

7. [Save changes] (変更の保存) をクリックします。

8. Aurora PostgreSQL DB クラスターのライターインスタンスを再起動して、変更を有効にします。Amazon RDS コンソールで、クラスターのプライマリ DB インスタンスを選択し、[Actions] (アクション) メニューから [Reboot] (再起動) を選択します。
9. インスタンスが使用可能になると、次のように論理レプリケーションがオンになっていること確認できます。

    a. `psql` を使用して、Aurora PostgreSQL DB クラスターのライターインスタンスに接続します。

    ```
    psql --host=your-db-cluster-instance-1.aws-region.rds.amazonaws.com --port=5432
     --username=postgres --password --dbname=labdb
    ```

    b. 次のコマンドを使用して、論理レプリケーションが有効になっていることを確認します。

    ```
    labdb=> SHOW rds.logical_replication;
     rds.logical_replication
    -------------------------
     on
    (1 row)
    ```

    c. `wal_level` が `logical` に設定されていることを確認してください。

    ```
    labdb=> SHOW wal_level;
      wal_level
    -----------
     logical
    (1 row)
    ```

論理レプリケーションを使用して、ソースの Aurora PostgreSQL DB クラスターからの変更とデータベーステーブルの同期を維持させる例については、「例: Aurora PostgreSQL DB クラスターにおける論理レプリケーションの使用」を参照してください。

論理レプリケーションをオフにする

レプリケーションタスクが完了したら、レプリケーションプロセスを停止し、レプリケーションスロットを削除して論理レプリケーションをオフにする必要があります。スロットを削除する前に、そのスロットが不要になったことを確認します。アクティブなレプリケーションスロットは削除できません。

論理レプリケーションをオフにするには

1. すべてのレプリケーションスロットを削除します。

   すべてのレプリケーションスロットを削除するには、パブリッシャーに接続して以下の SQL コマンドを実行します。

   ```
   SELECT pg_drop_replication_slot(slot_name)
     FROM pg_replication_slots
    WHERE slot_name IN (SELECT slot_name FROM pg_replication_slots);
   ```

   そのコマンドを実行する際、レプリケーションスロットをアクティブにすることはできません。
2. 「Aurora PostgreSQL DB クラスターの論理レプリケーションの設定」で説明したように、パブリッシャーに関連するカスタム DB クラスターのパラメータグループを変更し、`rds.logical_replication` パラメータを 0 に設定します。

   カスタムパラメータグループの詳細については、「DB クラスターパラメータグループのパラメータの変更」を参照してください。
3. `rds.logical_replication` パラメータの変更を有効にするには、パブリッシャーの Aurora PostgreSQL DB クラスターを再起動します。

Aurora PostgreSQL 論理レプリケーションライトスルーキャッシュの管理

デフォルトでは、Aurora PostgreSQL バージョン 14.5、13.8、12.12、11.17 以降は、ライトスルーキャッシュを使用して論理レプリケーションのパフォーマンスを向上させます。ライトスルーキャッシュがない場合、Aurora PostgreSQL はネイティブ PostgreSQL 論理レプリケーションプロセスの実装に Aurora ストレージレイヤーを使用します。そのためには、WAL データをストレージに書き込み、ストレージからデータを読み戻してデコードし、ターゲット (サブスクライバー) に送信 (複製) します。このため、Aurora PostgreSQL DB クラスターの論理レプリケーション中にボトルネックが発生する可能性があります。

ライトスルーキャッシュにより、Aurora ストレージレイヤーを使用する必要性が減ります。Aurora PostgreSQL は、常に Aurora ストレージレイヤーからの書き込みと読み取りを行う代わりに、バッファを使用して論理 WAL ストリームをキャッシュするため、レプリケーションプロセス中に常にディスクからプルする必要がありません。このバッファは、論理レプリケーションで使用される PostgreSQL ネイティブキャッシュであり、Aurora PostgreSQL DB クラスターのパラメータで `rds.logical_wal_cache` として識別されます。デフォルトでは、このキャッシュは Aurora PostgreSQL DB クラスターのバッファキャッシュ設定 (`shared_buffers`) の 32 分の 1 を使用しま

すが、64 KB より少ないことはなく、または 1 つの WAL セグメントのサイズ (通常は 16 MB) を超えません。

Aurora PostgreSQL DB クラスターで論理レプリケーションを使用する場合 (ライトスルーキャッシュをサポートするバージョンの場合)、キャッシュヒット率をモニタリングして、ユースケースでの機能を確認できます。そのためには、次の例に示すように、psql を使用して Aurora PostgreSQL DB クラスターの書き込みインスタンスに接続し、次に Aurora 関数 aurora_stat_logical_wal_cache を使用します。

```
SELECT * FROM aurora_stat_logical_wal_cache();
```

関数は、次のような出力を返します。

```
name       | active_pid | cache_hit | cache_miss | blks_read | hit_rate |
 last_reset_timestamp
-----------+------------+-----------+------------+-----------+----------+--------------
test_slot1 | 79183      | 24        | 0          | 24        | 100.00%  | 2022-08-05
 17:39...
test_slot2 |            | 1         | 0          |  1        | 100.00%  | 2022-08-05
 17:34...
(2 rows)
```

last_reset_timestamp 値は読みやすいように短くされています。この関数の詳細については、「aurora_stat_logical_wal_cache」を参照してください。

Aurora PostgreSQL には、ライトスルーキャッシュをモニタリングするための次の 2 つの関数が用意されています。

- aurora_stat_logical_wal_cache 関数 - リファレンスドキュメントについては、「aurora_stat_logical_wal_cache」を参照してください。
- aurora_stat_reset_wal_cache 関数 - リファレンスドキュメントについては、「aurora_stat_reset_wal_cache」を参照してください。

自動調整された WAL キャッシュサイズがワークロードに十分でない場合は、カスタム DB クラスターパラメータグループのパラメータを変更することによって、rds.logical_wal_cache の値を手動で変更できます。32kB 未満の正の値は 32kB として扱われることに注意してください。wal_buffers の詳細については、PostgreSQL ドキュメントの「ログ先行書き込み」を参照してください。

Aurora PostgreSQL の論理スロットの管理

ストリーミングアクティビティは、pg_replication_origin_status ビューにキャプチャされます。このビューの内容を確認するには、次に示す pg_show_replication_origin_status() 関数が使用できます。

```
SELECT * FROM pg_show_replication_origin_status();
```

次の SQL クエリを使用すると、論理スロットのリストを取得できます。

```
SELECT * FROM pg_replication_slots;
```

論理スロットを削除するには、次のコマンドに示すように、スロットの名前を指定して pg_drop_replication_slot を使用します。

```
SELECT pg_drop_replication_slot('test_slot');
```

例: Aurora PostgreSQL DB クラスターにおける論理レプリケーションの使用

以下の手順では、2 つの Aurora PostgreSQL DB クラスター間で論理レプリケーションを開始する方法を示しています。「Aurora PostgreSQL DB クラスターの論理レプリケーションの設定」で説明したように、パブリッシャーとサブスクライバーの両方が、論理レプリケーション用に設定されている必要があります。

パブリッシャーとして指定されている Aurora PostgreSQL DB クラスターでも、レプリケーションスロットへのアクセスを許可する必要があります。そのためには、Amazon VPC サービスに基づいて Aurora PostgreSQL DB クラスターの仮想パブリッククラウド (VPC) に関連付けられているセキュリティグループを変更します。サブスクライバーの VPC に関連付けられているセキュリティグループをパブリッシャーのセキュリティグループに追加することで、インバウンドアクセスを許可します。詳細については、「Amazon VPC ユーザーガイド」の「セキュリティグループを使用してリソースへのトラフィックを制御する」を参照してください。

これらの準備手順が完了したら、次の手順で説明されているように、パブリッシャーには PostgreSQL の CREATE PUBLICATION コマンドを、サブスクライバーには CREATE SUBSCRIPTION コマンドを使用できます。

2 つの Aurora PostgreSQL DB クラスター間で論理レプリケーションを開始するには

これらの手順では、Aurora PostgreSQL DB クラスターに、サンプルテーブルを作成するデータベースを含むライターインスタンスがあることを前提としています。

1. パブリッシャーとしての Aurora PostgreSQL DB クラスター

   a. 次の SQL ステートメントを使用してテーブルを作成します。

   ```
   CREATE TABLE LogicalReplicationTest (a int PRIMARY KEY);
   ```

   b. 次の SQL ステートメントを使用して、パブリッシャーデータベース内にデータを挿入します。

   ```
   INSERT INTO LogicalReplicationTest VALUES (generate_series(1,10000));
   ```

   c. 次の SQL ステートメントを使用して、テーブルにデータが存在することを確認します。

   ```
   SELECT count(*) FROM LogicalReplicationTest;
   ```

   d. 次のように、CREATE PUBLICATION ステートメントを使用してこのテーブルのパブリケーションを作成します。

   ```
   CREATE PUBLICATION testpub FOR TABLE LogicalReplicationTest;
   ```

2. サブスクライバーとしての Aurora PostgreSQL DB クラスター

   a. 次のように、パブリッシャーで作成したものと同じ LogicalReplicationTest テーブルをサブスクライバーに作成します。

   ```
   CREATE TABLE LogicalReplicationTest (a int PRIMARY KEY);
   ```

   b. このテーブルが空であることを確認します。

   ```
   SELECT count(*) FROM LogicalReplicationTest;
   ```

   c. サブスクリプションを作成して、パブリッシャーから変更を取得します。パブリッシャーの Aurora PostgreSQL DB クラスターについて、次の詳細を使用する必要があります。

   - host (ホスト) - パブリッシャーである Aurora PostgreSQL DB クラスターのライター DB インスタンス。
   - ポート - 書き込み DB インスタンスがリッスンするポート。PostgreSQL のデフォルト値は 5432 です。
   - dbname (データベース名) – データベースの名前。

```
CREATE SUBSCRIPTION testsub CONNECTION
   'host=publisher-cluster-writer-endpoint port=5432 dbname=db-name user=user
 password=password'
   PUBLICATION testpub;
```

> **Note**
>
> セキュリティ上のベストプラクティスとして、ここに示されているプロンプト以外のパスワードを指定してください。

サブスクリプションを作成すると、論理的なレプリケーションスロットがパブリッシャーで作成されます。

d. この例で、初期のデータがサブスクライバーにレプリケートされていることを確認するには、サブスクライバーデータベースで次の SQL ステートメントを使用します。

```
SELECT count(*) FROM LogicalReplicationTest;
```

パブリッシャーの以降のすべての変更がサブスクライバーにレプリケートされます。

論理レプリケーションはパフォーマンスに影響を与えます。レプリケーションタスクが完了したら、論理レプリケーションをオフにすることをお勧めします。

例: Aurora PostgreSQL と AWS Database Migration Service を使用した論理レプリケーション

AWS Database Migration Service 「AWS DMS」 を使用してデータベースまたはその一部をレプリケートできます。AWS DMS を使用して、Aurora PostgreSQL データベースから、別のオープンソースデータベースまたは商用データベースにデータを移行させます。AWS DMS の詳細については、『AWS Database Migration Service ユーザーガイド』を参照してください。

次の例では、Aurora PostgreSQL データベースからの (パブリッシャーとしての) 論理的なレプリケーションを設定し、次に移行のために AWS DMS を使用する方法を示します。この例では、「例: Aurora PostgreSQL DB クラスターにおける論理レプリケーションの使用」で作成したのと同じパブリッシャーおよびサブスクライバーを使用しています。

AWS DMS で論理的なレプリケーションを設定するには、Amazon RDS のパブリッシャーとサブスクライバーに関する詳細が必要です。特に、パブリッシャーの書き込み DB インスタンスとサブスクライバーの DB インスタンスに関する詳細が必要です。

パブリッシャーの書き込み DB インスタンスについては以下の情報を取得します。

- 仮想プライベートクラウド (VPC) の識別子
- サブネットグループ
- アベイラビリティーゾーン (AZ)
- VPC セキュリティグループ
- DB インスタンス ID

サブスクライバーの DB インスタンスについては以下の情報を取得します。

- DB インスタンス ID
- 出典エンジン

Aurora PostgreSQL での論理的なレプリケーションに AWS DMS を使用するには

1. AWS DMS を使用するようにパブリッシャーデータベースを準備します。

   これを行うには、PostgreSQL 10.x 以降のデータベースで、AWS DMS ラッパー関数をパブリッシャーデータベースに適用する必要があります。このステップと以降のステップの詳細については、AWS Database Migration Service ユーザーガイドの「AWS DMS のソースとして PostgreSQL バージョン 10.x 以降を使用する」を参照してください。
2. AWS Management Console にサインインして、AWS DMS で https://console.aws.amazon.com/dms/v2 コンソールを開きます。右上で、パブリッシャーとサブスクライバーがある同じ AWS リージョンを選択します。
3. AWS DMS レプリケーションインスタンスを作成します。

   パブリッシャーの書き込み DB インスタンスと同じ値を選択します。これらには、以下の設定が含まれます。

   - 「VPC」 で、書き込み DB インスタンスと同じ VPC を選択します。
   - 「Replication Subnet Group (レプリケーションのサブネットグループ)」 で、書き込み DB インスタンスと同じ値のサブネットグループを選択します。必要に応じて新規に作成してください。

- 「アベイラビリティーゾーン」 で、書き込み DB インスタンスと同じゾーンを選択します。
- 「VPC セキュリティグループ」 で、書き込み DB インスタンスと同じグループを選択します。

4. 出典の AWS DMS エンドポイントを作成します。

   次の設定を使用して、パブリッシャーを出典エンドポイントとして指定します。

   - 「Endpoint type (エンドポイントタイプ)」 で 「Source endpoint (出典エンドポイント)」を選択します。
   - 「Select RDS DB Instance (RDS DB インスタンスの選択)」 を選択します。
   - 「RDS Instance (RDS インスタンス)」で、パブリッシャーの書き込み DB インスタンスの DB 識別子を選択します。
   - 「Source engine (出典エンジン)」 で 「postgres」 を選択します。

5. ターゲットの AWS DMS エンドポイントを作成します。

   次の設定を使用して、サブスクライバーをターゲットエンドポイントとして指定します。

   - 「Endpoint type (エンドポイントタイプ)」 で 「Target endpoint (ターゲットエンドポイント)」 を選択します。
   - [Select RDS DB Instance (RDS DB インスタンスの選択)] を選択します。
   - [RDS Instance (RDS インスタンス)] で、サブスクライバーの DB インスタンスの DB 識別子を選択します。
   - [Source engine (出典エンジン)] の値を選択します。例えば、サブスクライバーが RDS PostgreSQL データベースの場合は、[postgres] を選択します。サブスクライバーが Aurora PostgreSQL データベースである場合は、[aurora-postgresql] を選択します。

6. AWS DMS データベース移行タスクを作成します。

   データベース移行タスクを使用して、移行するデータベーステーブルを指定し、ターゲットスキーマを使用してデータをマッピングして、ターゲットデータベースで新しいテーブルを作成します。少なくとも、[Task configuration (タスクの設定)] で以下の設定を使用します。

   - [Replication instance (レプリケーションインスタンス)] で、前のステップで作成したレプリケーションインスタンスを選択します。
   - [Source database endpoint (出典データベースエンドポイント)] で、前のステップで作成したパブリッシャー出典を選択します。

- [Target database endpoint (ターゲットデータベースエンドポイント)] で、前のステップで作成したサブスクライバーターゲットを選択します。

タスクの残りの詳細は、移行プロジェクトに応じて異なります。DMS タスクのすべての詳細を指定する方法については、AWS Database Migration Service ユーザーガイドの「AWS DMS タスクの使用」を参照してください。

AWS DMS は、タスクを作成した後で、パブリッシャーからサブスクライバーへのデータの移行を開始します。

Amazon Bedrock のナレッジベースとしての Aurora PostgreSQL の使用

Aurora PostgreSQL 15.4、14.9、13.12、12.16 バージョンから、Aurora PostgreSQL DB クラスターを Amazon Bedrock のナレッジベースとして使用できます。詳細については、「Amazon Aurora でベクトルストアを作成する」を参照してください。ナレッジベースは、Amazon S3 バケットに保存されている非構造化テキストデータを自動的に取得し、テキストチャンクとベクトルに変換して、PostgreSQL データベースに保存します。生成 AI アプリケーションでは、Agents for Amazon Bedrock を使用してナレッジベースに保存されているデータをクエリし、それらのクエリの結果を使用して基本的なモデルが提供する回答を補強できます。このワークフローは、取得拡張生成 (RAG) と呼ばれます。RAG の詳細については、「取得拡張生成 (RAG)」を参照してください。

トピック

前提条件

Amazon Bedrock のナレッジベースとして Aurora PostgreSQL クラスターを使用するための以下の前提条件を理解してください。大まかに言うと、Bedrock で使用する以下のサービスを設定する必要があります。

- 以下のバージョンで作成された Amazon Aurora PostgreSQL DB クラスター：
  - 15.4 以降のバージョン

- 14.9 以降のバージョン
- 13.12 以降のバージョン
- 12.16 以降のバージョン

> **Note**
>
> ターゲットデータベースで pgvector 拡張機能を有効にし、バージョン 0.5.0 以降を使用する必要があります。詳細については、「HNSW インデックス作成による pgvector v0.5.0」を参照してください。

- Data API
- Secrets Manager で管理されているユーザー。詳細については、「Amazon Aurora および AWS Secrets Manager によるパスワード管理」を参照してください。

Amazon Bedrock のナレッジベースとして使用する Aurora PostgreSQL の準備

Aurora PostgreSQL DB クラスターを Amazon Bedrock のナレッジベースとして使用するには、以下のステップに従って作成および設定する必要があります。

1. Aurora PostgreSQL DB クラスターを作成します。詳細については、「Aurora PostgreSQL DB クラスターの作成と接続」を参照してください。
2. Aurora PostgreSQL DB クラスターの作成時に Data API を有効にします。サポートされているバージョンの詳細については、「RDS Data API の使用」を参照してください。
3. Amazon Bedrock で使用する Aurora PostgreSQL DB クラスターの Amazon リソースネーム (ARN) を書き留めておきます。詳細については、「Amazon リソースネーム」を参照してください。
4. マスターユーザーを使用してデータベースにログインし、pgvector をセットアップします。拡張機能がインストールされていない場合は、次のコマンドを使用します。

   ```
   CREATE EXTENSION IF NOT EXISTS vector;
   ```

   HNSW インデックス作成をサポートする pgvector 0.5.0 以降のバージョンを使用します。詳細については、「HNSW インデックス作成による pgvector v0.5.0」を参照してください。

pg_vector がインストールされているバージョンを確認するには、以下のコマンドを使用します。

```
postgres=>SELECT extversion FROM pg_extension WHERE extname='vector';
```

5. Bedrock がデータのクエリに使用できる特定のスキーマを作成します。スキーマを作成するには以下のコマンドを使用します。

```
CREATE SCHEMA bedrock_integration;
```

6. Bedrock がデータベースのクエリに使用できる新しいロールを作成します。次のコマンドを使用して、新しいロールを作成します。

```
CREATE ROLE bedrock_user WITH PASSWORD password LOGIN;
```

> **Note**
>
> Secrets Manager のパスワードを作成する場合と同じものを使用するため、このパスワードを書き留めておきます。

7. bedrock_integration スキーマにテーブルまたはインデックスを作成できるように、スキーマを管理する bedrock_user アクセス許可を付与します。

```
GRANT ALL ON SCHEMA bedrock_integration to bedrock_user;
```

8. bedrock_user として ログインし、bedrock_integration schema にテーブルを作成します。

```
CREATE TABLE bedrock_integration.bedrock_kb (id uuid PRIMARY KEY, embedding
 vector(1536), chunks text, metadata json);
```

9. bedrock がデータのクエリに使用できるコサイン演算子を使用してインデックスを作成することをお勧めします。

```
CREATE INDEX on bedrock_integration.bedrock_kb USING hnsw (embedding
 vector_cosine_ops);
```

10. AWS Secrets Manager データベースシークレットを作成します。詳細については、「AWSSecrets Manager データベースのシークレット」を参照してください。

Bedrock コンソールでのナレッジベースの作成

ナレッジベースのベクトルストアとして使用するように Aurora PostgreSQL を準備するときは、Amazon Bedrock コンソールに提供する必要がある以下の詳細情報を収集します。

- Amazon Aurora DB クラスターの ARN
- シークレット ARN
- データベース名 (postgres など)
- テーブル名 - スキーマ修飾名、つまり CREATE TABLE bedrock_integration.bedrock_kb を指定するように通知します。これは、bedrock_integration スキーマに bedrock_kb テーブルを作成します。
- テーブルフィールド:

  ID: (id)

  テキストチャンク (チャンク)

  ベクトル埋め込み (埋め込み)

  メタデータ (メタデータ)

これらの詳細を使用して、Bedrock コンソールでナレッジベースを作成できます。詳細については、「Amazon Aurora でベクトルストアを作成する」を参照してください。

Aurora がナレッジベースとして追加されたら、データソースをそこに取り込みます。詳細については、「データソースをナレッジベースに取り込む」を参照してください。

Amazon Aurora PostgreSQL を他の AWS のサービスと統合する

Amazon Aurora を AWS の他のサービスと統合することで、Aurora PostgreSQL DB クラスターを拡張し、使用できる AWS クラウドの機能を追加できます。Aurora PostgreSQL DB クラスターでは、AWS のサービスを使用して以下のことができます。

- Amazon RDS Performance Insights を使用して Aurora PostgreSQL DB インスタンスのパフォーマンスを迅速に収集、表示、および評価します。Performance Insights は、既存の Amazon RDS モニタリング機能を拡張して、データベースのパフォーマンスを明確にし、これに影響を与えるあらゆる問題を分析しやすくします。Performance Insights ダッシュボードを使用してデータベースロードを視覚化したり、ロードを待機、SQL ステートメント、ホスト、ユーザー別にフィ

ルタリングしたりできます。Performance Insights の詳細については、「Amazon Aurora での Performance Insights を使用したDB 負荷のモニタリング」を参照してください。

- ログデータを Amazon CloudWatch Logs に発行するために、Aurora PostgreSQL DB クラスターを設定します。CloudWatch Logs により、高い耐久性の高いを備えたストレージがログレコード用に提供されます。CloudWatch Logs を使用すると、ログデータのリアルタイム分析や、CloudWatch を使用したアラームの作成、メトリクスの表示を行うことができます。詳細については、「Amazon CloudWatch Logs への Aurora PostgreSQL ログの発行」を参照してください。
- Amazon S3 バケットから Aurora PostgreSQL DB クラスターにデータをインポートするか、Aurora PostgreSQL DB クラスターから Amazon S3 バケットにデータをエクスポートします。詳細については、Amazon S3 から Aurora PostgreSQL DB クラスターにデータをインポートする および Aurora PostgreSQL DB クラスターから Amazon S3 へのデータのエクスポート を参照してください。
- SQL 言語を使用して、データベースアプリケーションに機械学習ベースの予測を追加します。Aurora 機械学習では、Aurora データベースと、SageMaker および Amazon Comprehend といった AWS の機械学習 (ML) サービスの、高度に最適化した統合を使用します。詳細については、「Aurora PostgreSQL で Amazon Aurora 機械学習を使用する」を参照してください。
- Aurora PostgreSQL DB クラスターから AWS Lambda 関数を呼び出します。呼び出すには、Aurora PostgreSQL で提供される `aws_lambda` PostgreSQL エクステンションを使用します。詳細については、「Aurora PostgreSQL DB クラスターから AWS Lambda 関数を呼び出す」を参照してください。
- Amazon Redshift と Aurora PostgreSQL からのクエリを統合します。詳細については、Amazon Redshift データベースデベロッパーガイドの「PostgreSQL への横串検索を使用したスタート方法」を参照してください。

Amazon S3 から Aurora PostgreSQL DB クラスターにデータをインポートする

Amazon Simple Storage Service を使用して保存されたデータを、Aurora PostgreSQL DB クラスターインスタンス 上のテーブルにインポートできます。これを行うには、Aurora PostgreSQL `aws_s3`拡張機能を最初にインストールします。この拡張機能には、Amazon S3 バケットからのデータのインポートに使用する関数が含まれます。バケットとは、Amazon S3 のオブジェクトおよびファイルのコンテナです。データは、カンマ区切り値 (CSV) ファイル、テキストファイル、または圧縮 (gzip) ファイルでインポートできます。次に、拡張機能のインストール方法と、Amazon S3 からテーブルにデータをインポートする方法について説明します。

Amazon S3 から にインポートするには、データベースで PostgreSQL バージョン 10.7 以降を実行している必要があります。Aurora PostgreSQL

Amazon S3 にデータが保存されていない場合は、まずバケットを作成し、データを保存する必要があります。詳細については、Amazon Simple Storage Service コンソールユーザーガイドの以下のトピックを参照してください。

- バケットの作成
- バケットにオブジェクトを追加する

Amazon S3 からのクロスアカウントインポートがサポートされています。詳細については、「Amazon Simple Storage Service ユーザーガイド」の「クロスアカウントアクセス許可の付与」を参照してください。

S3 からデータをインポートする際は、カスタマーマネージドキーを暗号化に使用できます。詳細については、Amazon Simple Storage Service ユーザーガイドの「AWS KMS に保存される KMS キー」を参照してください。

> **Note**
>
> Amazon S3 からデータをインポートすることは、Aurora Serverless v1 でサポートされていません。Aurora Serverless v2 に対してサポートされています。

トピック

- aws_s3 拡張機能のインストール
- Amazon S3 データからのデータのインポートの概要
- Amazon S3 バケットへのアクセスを設定する
- Amazon S3 から Aurora PostgreSQL DB クラスターにデータをインポートする
- 関数リファレンス

aws_s3 拡張機能のインストール

Aurora PostgreSQL DB クラスター で Amazon S3 を使用する前に、aws_s3 拡張機能をインストールする必要があります。この拡張機能には、Amazon S3 からデータをインポートするための関数が含まれます。また、Aurora PostgreSQL DB クラスターのインスタンス から Amazon S3 バケットへデータを エクスポートするための関数も含まれています。詳しくは、「Aurora PostgreSQL DB ク

ラスターから Amazon S3 へのデータのエクスポート」を参照してください。aws_s3 拡張機能は aws_commons 拡張機能の一部のヘルパー関数に依存しており、必要に応じて自動的にインストールされます。

aws_s3 拡張機能をインストールするには

1. rds_superuser 権限があるユーザーとして、psql (または pgAdmin) を使用して Aurora PostgreSQL DB クラスターのライターインスタンス に接続します。設定プロセス中にデフォルトの名前を保持している場合は、postgres として接続します。

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password
```

2. 拡張機能をインストールするには、次のコマンドを実行します。

```
postgres=> CREATE EXTENSION aws_s3 CASCADE;
NOTICE: installing required extension "aws_commons"
CREATE EXTENSION
```

3. 拡張機能がインストールされていることを確認するには、psql \dx メタコマンドを使用します。

```
postgres=> \dx
       List of installed extensions
    Name     | Version |   Schema   |                 Description
-------------+---------+------------+---------------------------------------------
 aws_commons | 1.2     | public     | Common data types across AWS services
 aws_s3      | 1.1     | public     | AWS S3 extension for importing data from S3
 plpgsql     | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

Amazon S3 からデータをインポートし、データを Amazon S3 にエクスポートするための関数が使用できるようになりました。

Amazon S3 データからのデータのインポートの概要

S3 データを Aurora PostgreSQL にインポートするには

まず、関数で指定する必要がある詳細情報を収集します。この情報には、Aurora PostgreSQL DB クラスターのインスタンス のテーブルの名前、バケット名、ファイルパス、ファイルタイプ、Amazon

S3 データが保存される AWS リージョンが含まれます。詳細については、Amazon Simple Storage Service ユーザーガイドの「オブジェクトの表示」を参照してください。

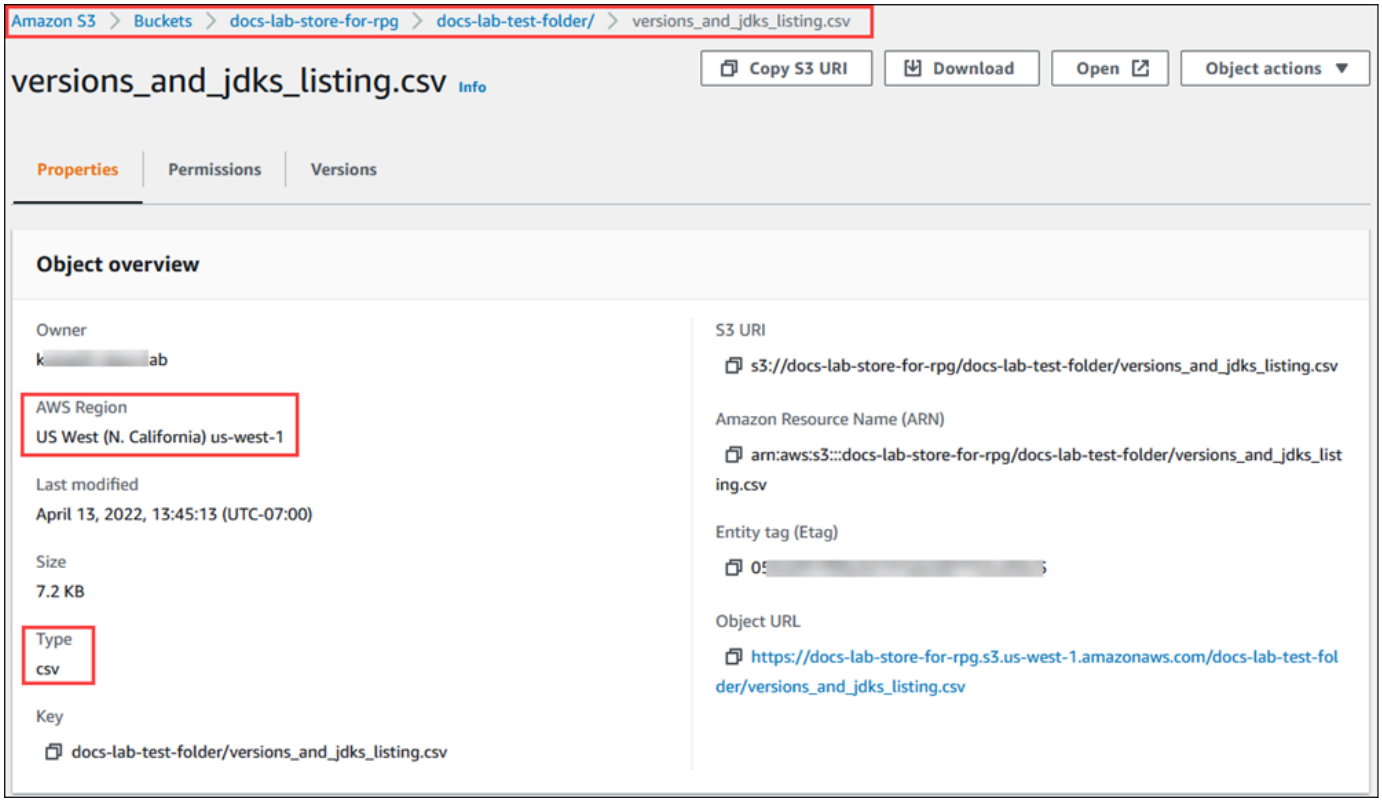
> **Note**
>
> Amazon S3 からのマルチパートデータインポートは現在サポートされていません。

1. `aws_s3.table_import_from_s3` 関数によってデータがインポートされるテーブルの名前を取得します。例えば、次のコマンドにより、後の手順で使用されるテーブル `t1` が作成されます。

   ```
   postgres=> CREATE TABLE t1
       (col1 varchar(80),
       col2 varchar(80),
       col3 varchar(80));
   ```

2. Amazon S3 バケットの詳細とインポートするデータを取得します。これを実行するには、Amazon S3 コンソール (https://console.aws.amazon.com/s3/) を開き、[Bucket] (バケット) を選択します。リストで、データを含むバケットを探します。バケットを選択し、オブジェクト概要ページを開き、[Properties] (プロパティ) を選択します。

   バケット名、パス、AWS リージョン、およびファイルタイプを書き留めておきます。IAM ロールによる Amazon S3 へのアクセスを設定するには、後で Amazon リソースネーム (ARN) が必要になります。詳細については、「Amazon S3 バケットへのアクセスを設定する」を参照してください。次のイメージは例を示しています。



3. AWS CLI コマンド `aws s3 cp` を使用して、Amazon S3 バケットのデータへのパスを確認できます。情報が正しい場合、このコマンドは Amazon S3 ファイルのコピーをダウンロードします。

```
aws s3 cp s3://sample_s3_bucket/sample_file_path ./
```

4. Aurora PostgreSQL DB クラスター に対するアクセス許可を設定して、Amazon S3 バケット上のファイルへのアクセスを許可します。これを行うには、AWS Identity and Access Management (IAM) ロールまたはセキュリティ認証情報を使用します。詳しくは、「Amazon S3 バケットへのアクセスを設定する」を参照してください。

5. 収集したパスと他の Amazon S3 オブジェクトの詳細 (ステップ 2 を参照) を `create_s3_uri` 関数で指定し、Amazon S3 URI オブジェクトを構成します。この関数の詳細については、「aws_commons.create_s3_uri」を参照してください。psql セッション中にこのオブジェクトを構成する例は次のとおりです。

```
postgres=> SELECT aws_commons.create_s3_uri(
   'docs-lab-store-for-rpg',
   'versions_and_jdks_listing.csv',
   'us-west-1'
) AS s3_uri \gset
```

次のステップでは、このオブジェクト (aws_commons._s3_uri_1) を aws_s3.table_import_from_s3 関数に渡して、データをテーブルにインポートします。

6. aws_s3.table_import_from_s3 関数を呼び出して、Amazon S3 からテーブルにデータをインポートします。参考情報については、「aws_s3.table_import_from_s3」を参照してください。例については、「Amazon S3 から Aurora PostgreSQL DB クラスターにデータをインポートする」を参照してください。

Amazon S3 バケットへのアクセスを設定する

Amazon S3 ファイルからデータをインポートするには、Aurora PostgreSQL DB クラスターに、ファイルが含まれている Amazon S3 バケットへのアクセス許可を与える必要があります。次のトピックで説明する 2 つの方法のいずれかで、Amazon S3 バケットへのアクセスを提供します。

トピック

- IAM ロールを使用した Amazon S3 バケットへのアクセス
- セキュリティ認証情報を使用して Amazon S3 バケットにアクセスする
- Amazon S3 へのアクセスのトラブルシューティング

IAM ロールを使用した Amazon S3 バケットへのアクセス

Amazon S3 ファイルからデータをロードするには、ファイルが含まれる Amazon S3 バケットへのアクセス許可を Aurora PostgreSQL DB クラスターに与えます。こうすれば、追加の認証情報を管理したり、aws_s3.table_import_from_s3 関数呼び出しで提供したりする必要はありません。

これを行うには、Amazon S3 バケットへのアクセスを提供する IAM ポリシーを作成します。IAM ロールを作成して、ポリシーをロールにアタッチします。次に、IAM ロールを DB クラスターに割り当てます。

> **Note**
>
> IAM ロールを Aurora Serverless v1 DB クラスターに関連付けることができないため、次の手順は適用されません。

IAM ロール経由で、Amazon S3 へのアクセス権を Aurora PostgreSQL DB クラスターに付与するには

1. IAM ポリシーを作成します。

   ポリシーは、Aurora PostgreSQL DB クラスターに Amazon S3 へのアクセスを許可するバケットとオブジェクトのアクセス許可を付与します。

   ポリシーに、Amazon S3 バケットから Aurora PostgreSQL へのファイル転送を許可ための次の必須アクションを含めます。

   - `s3:GetObject`
   - `s3:ListBucket`

   ポリシーに次のリソースを含めて、Amazon S3 バケットとバケット内のオブジェクトを識別します。これは、Amazon S3 にアクセスするための Amazon リソースネーム (ARN) 形式を示しています。

   - arn:aws:s3:::*your-s3-bucket*
   - arn:aws:s3:::*your-s3-bucket*/*

   Aurora PostgreSQL の IAM ポリシーの作成の詳細については、「IAM データベースアクセス用の IAM ポリシーの作成と使用」を参照してください。IAM ユーザーガイドの「チュートリアル: はじめてのカスタマー管理ポリシーの作成とアタッチ」も参照してください。

   以下の AWS CLI コマンドでは、これらのオプションを指定して、`rds-s3-import-policy` という名前の IAM ポリシーを作成します。このポリシーでは、`your-s3-bucket` という名前のバケットへのアクセス権が付与されます。

   > **Note**
   >
   > このコマンドによって返されるポリシー のAmazon リソースネーム (ARN) をメモしておきます。ポリシーを IAM ロールにアタッチする場合、後続のステップで ARN が必要です。

Example

Linux、macOS、Unix の場合:

```
aws iam create-policy \
   --policy-name rds-s3-import-policy \
   --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3import",
         "Action": [
           "s3:GetObject",
           "s3:ListBucket"
         ],
         "Effect": "Allow",
         "Resource": [
           "arn:aws:s3:::your-s3-bucket",
           "arn:aws:s3:::your-s3-bucket/*"
         ]
       }
     ]
   }'
```

Windows の場合:

```
aws iam create-policy ^
   --policy-name rds-s3-import-policy ^
   --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3import",
         "Action": [
           "s3:GetObject",
           "s3:ListBucket"
         ],
         "Effect": "Allow",
         "Resource": [
           "arn:aws:s3:::your-s3-bucket",
           "arn:aws:s3:::your-s3-bucket/*"
```

```
        ]
      }
    ]
  }'
```

2. IAM ロールを作成します。

これを行うと、Aurora PostgreSQL がユーザーに代わってこの IAM ロールを引き受け、Amazon S3 バケットにアクセスできます。詳細については、IAM ユーザーガイドの「IAM ユーザーにアクセス許可を委任するロールの作成」を参照してください。

リソースポリシー内では aws:SourceArn および aws:SourceAccount のグローバル条件コンテキストキーを使用して、サービスに付与するリソースへのアクセス許可を制限することをお勧めします。これは、混乱した使節の問題に対する最も効果的な保護方法です。

グローバル条件コンテキストキーの両方を使用し、aws:SourceArn の値にアカウント ID が含まれている場合、同じポリシーステートメントで使用する場合は、aws:SourceArn の値と aws:SourceAccount の値のアカウントでは同じアカウント ID を使用する必要があります。

- 単一リソースに対するクロスサービスアクセスが必要な場合は aws:SourceArn を使用します。
- そのアカウント内の任意のリソースをクロスサービス使用に関連付けることを許可する場合、aws:SourceAccountを使用します。

ポリシーでは、必ずリソースの完全な ARN を持つ aws:SourceArn グローバル条件コンテキストキーを使用してください。以下の例は、AWS CLI コマンドを使用して、rds-s3-import-role という名前のロールを作成する方法を示しています。

Example

Linux、macOS、Unix の場合:

```
aws iam create-role \
   --role-name rds-s3-import-role \
   --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
```

```
            "Service": "rds.amazonaws.com"
          },
          "Action": "sts:AssumeRole",
          "Condition": {
              "StringEquals": {
                 "aws:SourceAccount": "111122223333",
                 "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:clustername"
                 }
              }
       }
     ]
   }'
```

Windows の場合:

```
aws iam create-role ^
   --role-name rds-s3-import-role ^
   --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:clustername"
                }
             }
       }
     ]
   }'
```

3. 作成した IAM ポリシーを、作成した IAM ロールにアタッチします。

   次の AWS CLI コマンドは、先ほどのステップで作成したポリシーを `rds-s3-import-role` という名前のロールに添付し、*`your-policy-arn`* を前のステップでメモしたポリシー ARN に置き換えます。

Example

Linux、macOS、Unix の場合:

```
aws iam attach-role-policy \
   --policy-arn your-policy-arn \
   --role-name rds-s3-import-role
```

Windows の場合:

```
aws iam attach-role-policy ^
   --policy-arn your-policy-arn ^
   --role-name rds-s3-import-role
```

4. DB クラスターに IAM ロールを追加します。

   これを行うには、以下で説明するように、AWS Management Console または AWS CLI を使用します。

コンソール

コンソールを使用して PostgreSQL DB クラスターの IAM ロールを追加するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. 詳細を表示するには、PostgreSQL DB クラスターの名前を選択します。
3. [接続とセキュリティ] タブの [IAM ロールの管理] セクションで、このクラスターに [IAM ロールを追加] で追加するロールを選択します。
4. [Feature] で、[s3Import] を選択します。
5. [Add role] を選択します。

AWS CLI

CLI を使用して PostgreSQL DB クラスターの IAM ロールを追加するには

- 次のコマンドを使用して、`my-db-cluster` という名前の PostgreSQL DB クラスターにロールを追加します。*`your-role-arn`* を、以前のステップで書き留めたロール ARN に置き換えます。`s3Import` オプションの値に `--feature-name` を使用します。

  Example

  Linux、macOS、Unix の場合:

  ```
  aws rds add-role-to-db-cluster \
     --db-cluster-identifier my-db-cluster \
     --feature-name s3Import \
     --role-arn your-role-arn   \
     --region your-region
  ```

  Windows の場合:

  ```
  aws rds add-role-to-db-cluster ^
     --db-cluster-identifier my-db-cluster ^
     --feature-name s3Import ^
     --role-arn your-role-arn ^
     --region your-region
  ```

RDS API

Amazon RDS API を使用して PostgreSQL DB クラスターに IAM ロールを追加するには、AddRoleToDBCluster オペレーションを呼び出します。

セキュリティ認証情報を使用して Amazon S3 バケットにアクセスする

必要に応じて、IAM ロールでアクセスを提供する代わりに、セキュリティ認証情報を使用して Amazon S3 バケットへのアクセスを提供できます このためには、aws_s3.table_import_from_s3 関数呼び出しで `credentials` パラメータを指定します。

`credentials` パラメータは、型の構造体 `aws_commons._aws_credentials_1` で、AWS 認証情報を含みます。aws_commons.create_aws_credentials 関数を使用し

て、aws_commons._aws_credentials_1 構造でアクセスキーおよびシークレットキーを設定します。以下に例を示します。

```
postgres=> SELECT aws_commons.create_aws_credentials(
   'sample_access_key', 'sample_secret_key', '')
AS creds \gset
```

aws_commons._aws_credentials_1 構造を作成したら、以下に示すように、aws_s3.table_import_from_s3 関数を credentials パラメータと共に使用してデータをインポートします。

```
postgres=> SELECT aws_s3.table_import_from_s3(
   't', '', '(format csv)',
   :'s3_uri',
   :'creds'
);
```

または、aws_commons.create_aws_credentials 関数の呼び出しのインラインを aws_s3.table_import_from_s3 関数の呼び出し内に含めることもできます。

```
postgres=> SELECT aws_s3.table_import_from_s3(
   't', '', '(format csv)',
   :'s3_uri',
   aws_commons.create_aws_credentials('sample_access_key', 'sample_secret_key', '')
);
```

Amazon S3 へのアクセスのトラブルシューティング

Amazon S3 からデータをインポートしようとしたときに接続の問題が発生した場合は、次の推奨事項を参照してください。

- Amazon Aurora のアイデンティティおよびアクセスのトラブルシューティング
- Amazon Simple Storage Service ユーザーガイド の「Troubleshooting Amazon S3」
- IAM ユーザーガイド の Amazon S3 のトラブルシューティングと IAM

Amazon S3 から Aurora PostgreSQL DB クラスターにデータをインポートする

aws_s3 拡張機能の table_import_from_s3 関数を使用して Amazon S3 バケットからデータをインポートします。参考情報については、「aws_s3.table_import_from_s3」を参照してください。

> **Note**
>
> 以下の例では、IAM ロールメソッドを使用して、Amazon S3 バケットへのアクセスを許可します。したがって、aws_s3.table_import_from_s3 関数呼び出しには認証情報パラメータは含まれません。

次の例は、代表的な例を示しています。

```
postgres=> SELECT aws_s3.table_import_from_s3(
   't1',
   '',
   '(format csv)',
   :'s3_uri'
);
```

パラメータは次のとおりです。

- t1 - データのコピー先となる PostgreSQL DB クラスター内のテーブルの名前。
- '' - データベーステーブル内の列のオプションのリスト。S3 データをコピーする列とテーブル列を指定するには、このパラメータを使用します。列を指定しない場合は、すべての列がテーブルにコピーされます。列のリストの使用例については、カスタム区切り文字を使用する Amazon S3 ファイルをインポートする を参照してください。
- (format csv) - PostgreSQL COPY 引数。このコピープロセスでは、PostgreSQL COPY コマンドの引数と形式を使用してデータをインポートします。フォーマットとしては、この例のようなカンマ区切り値 (CSV)、テキスト、およびバイナリを指定できます。デフォルトではテキストに設定されています。
- s3_uri - Amazon S3 ファイルを識別する情報を含む構造。aws_commons.create_s3_uri 関数を使用して s3_uri 構造を作成する例については、「Amazon S3 データからのデータのインポートの概要」を参照してください。

この関数の詳細については、「aws_s3.table_import_from_s3」を参照してください。

このaws_s3.table_import_from_s3関数はテキストを返します。Amazon S3 バケットからインポートする他の種類のファイルを指定するには、次の例のいずれかを参照してください。

> **Note**
>
> 0 バイトファイルをインポートすると、エラーが発生します。

トピック

- カスタム区切り文字を使用する Amazon S3 ファイルをインポートする
- Amazon S3 圧縮 (gzip) ファイルをインポートする
- エンコードされた Amazon S3 ファイルをインポートする

カスタム区切り文字を使用する Amazon S3 ファイルをインポートする

以下の例では、カスタム区切り文字を使用するファイルのインポート方法を示します。また、column_list 関数の aws_s3.table_import_from_s3 パラメータを使用して、データベースのデータを置く場所を制御する方法を示します。

この例では、次の情報が Amazon S3 ファイル内のパイプ区切りの列に編成されているとします。

```
1|foo1|bar1|elephant1
2|foo2|bar2|elephant2
3|foo3|bar3|elephant3
4|foo4|bar4|elephant4
...
```

カスタム区切り文字を使用するファイルをインポートするには

1. インポートされたデータのテーブルをデータベースに作成します。

   ```
   postgres=> CREATE TABLE test (a text, b text, c text, d text, e text);
   ```

2. データを Amazon S3 からインポートするには、次の形式の aws_s3.table_import_from_s3 関数を使用します。

   または、aws_commons.create_s3_uri 関数の呼び出しのインラインを aws_s3.table_import_from_s3 関数の呼び出し内に含めて、ファイルを指定することもできます。

   ```
   postgres=> SELECT aws_s3.table_import_from_s3(
      'test',
   ```

```
   'a,b,d,e',
   'DELIMITER ''|''',
   aws_commons.create_s3_uri('sampleBucket', 'pipeDelimitedSampleFile', 'us-
east-2')
);
```

データが、次の列のテーブル内に入りました。

```
postgres=> SELECT * FROM test;
a | b | c | d | e
---+------+---+---+------+-----------
1 | foo1 | | bar1 | elephant1
2 | foo2 | | bar2 | elephant2
3 | foo3 | | bar3 | elephant3
4 | foo4 | | bar4 | elephant4
```

Amazon S3 圧縮 (gzip) ファイルをインポートする

以下の例では、gzip で圧縮されているファイルを Amazon S3 からインポートする方法を示します。インポートするファイルには、次の Amazon S3 メタデータが必要です。

- キー: `Content-Encoding`
- 値: `gzip`

AWS Management Console を使用してファイルをアップロードする場合、通常このメタデータは、システムにより適用されます。AWS Management Console、AWS CLI、または API による Amazon S3 へのファイルのアップロードについては、「Amazon Simple Storage Service ユーザーガイド」の「オブジェクトのアップロード」を参照してください。

Amazon S3 のメタデータに関する情報、およびシステム提供メタデータの詳細については、「Amazon Simple Storage Service ユーザーガイド」の「Amazon S3 コンソールでのオブジェクトメタデータの編集」を参照してください。

以下に示されているように、gzip ファイルを Aurora PostgreSQL DB クラスターにインポートします。

```
postgres=> CREATE TABLE test_gzip(id int, a text, b text, c text, d text);
postgres=> SELECT aws_s3.table_import_from_s3(
 'test_gzip', '', '(format csv)',
```

```
 'myS3Bucket', 'test-data.gz', 'us-east-2'
);
```

エンコードされた Amazon S3 ファイルをインポートする

以下の例では、Windows-1252 でエンコードされているファイルを Amazon S3 からインポートする方法を示します。

```
postgres=> SELECT aws_s3.table_import_from_s3(
 'test_table', '', 'encoding ''WIN1252''',
 aws_commons.create_s3_uri('sampleBucket', 'SampleFile', 'us-east-2')
);
```

関数リファレンス

関数

- aws_s3.table_import_from_s3
- aws_commons.create_s3_uri
- aws_commons.create_aws_credentials

aws_s3.table_import_from_s3

Amazon S3 データを Aurora PostgreSQL テーブルにインポートします。aws_s3 拡張機能には、aws_s3.table_import_from_s3 関数が含まれます。戻り値はテキストです。

構文

必須のパラメータは、table_name、column_list、options です。これらのパラメータを使用して、データベースを特定し、データをテーブルにコピーする方法を指定します。

また、次のパラメータを使用することもできます。

- s3_info パラメータは、インポートする Amazon S3 ファイルを指定します。このパラメータを使用する場合、PostgreSQL DB クラスターの IAM ロールを使用して、Amazon S3 へのアクセス権を付与します。

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
```

```
   options text,
   s3_info aws_commons._s3_uri_1
)
```

- `credentials` パラメータは、Amazon S3 にアクセスするための認証情報を指定します。このパラメータを使用する場合、IAM ロールは使用しません。

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   s3_info aws_commons._s3_uri_1,
   credentials aws_commons._aws_credentials_1
)
```

パラメータ

table_name

データのインポート先となる PostgreSQL データベーステーブルの名前を含む必須のテキスト文字列。

column_list

データをコピーする PostgreSQL データベーステーブル列のオプションリストを含む必須のテキスト文字列。文字列が空の場合、テーブルの列がすべて使用されます。例については、「カスタム区切り文字を使用する Amazon S3 ファイルをインポートする」を参照してください。

options:

PostgreSQL COPY コマンドの引数を含む必須のテキスト文字列。これらの引数は PostgreSQL のテーブルにデータをコピーする方法を指定します。詳細については、「PostgreSQL COPY ドキュメント」を参照してください。

s3_info

S3 オブジェクトに関する以下の情報を含む `aws_commons._s3_uri_1` 複合型。

- `bucket` - ファイルを含む Amazon S3 バケット名。
- `file_path` - Amazon S3ファイルのパスを含むファイル名。
- `region` - ファイルがある AWS リージョン。AWS リージョン名と関連する値のリストについては、「リージョンとアベイラビリティーゾーン」を参照してください。

credentials

インポートオペレーションに使用する次の認証情報を含む aws_commons._aws_credentials_1 複合型。

- アクセスキー
- シークレットキー
- セッショントークン

aws_commons._aws_credentials_1 複合構造を作成する方法については、「aws_commons.create_aws_credentials」を参照してください。

代替構文

テストしやすいように、s3_info パラメータや credentials パラメータではなく、拡張されたパラメータセットを使用することができます。以下は、aws_s3.table_import_from_s3 関数の構文のバリエーションです。

- Amazon S3 ファイルを識別するために s3_info パラメータを使用する代わりに、bucket、file_path、および region パラメータの組み合わせを使用します。この関数の形式を使用する場合は、PostgreSQL DB インスタンスの IAM ロールを使用して、Amazon S3 へのアクセス権を付与します。

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   bucket text,
   file_path text,
   region text
)
```

- Amazon S3 アクセスを指定するために credentials パラメータを使用する代わりに、access_key、session_key、および session_token パラメータの組み合わせを使用します。

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
```

```
    bucket text,
    file_path text,
    region text,
    access_key text,
    secret_key text,
    session_token text
)
```

代替パラメータ

bucket (バケット)

ファイルを含む Amazon S3 バケットの名前を含むテキスト文字列。

file_path

ファイルのパスを含むAmazon S3ファイル名を含むテキスト文字列。

region

ファイルの AWS リージョンの場所を識別するテキスト文字列。AWS リージョン 名と関連する値のリストについては、「リージョンとアベイラビリティーゾーン」を参照してください。

access_key

インポートオペレーションに使用するアクセスキーを含むテキスト文字列。デフォルトは NULL です。

secret_key

インポートオペレーションに使用するシークレットキーを含むテキスト文字列。デフォルトは NULL です。

session_token

(オプション) インポートオペレーションに使用するセッションキーを含むテキスト文字列。デフォルトは NULL です。

aws_commons.create_s3_uri

Amazon S3 ファイル情報を保持するように、aws_commons._s3_uri_1 構造を作成します。aws_commons.create_s3_uri 関数の結果は、s3_info 関数の aws_s3.table_import_from_s3 パラメータで使用します。

構文

```
aws_commons.create_s3_uri(
   bucket text,
   file_path text,
   region text
)
```

パラメータ

bucket (バケット)

ファイルの Amazon S3 バケット名を含む必須のテキスト文字列。

file_path

ファイルのパスを含む Amazon S3 ファイル名を含む必須テキスト文字列。

region

ファイルがある AWS リージョン を含む必須のテキスト文字列。AWS リージョン 名と関連する値のリストについては、「リージョンとアベイラビリティーゾーン」を参照してください。

aws_commons.create_aws_credentials

`aws_commons._aws_credentials_1` 構造でアクセスキーとシークレットキーを設定します。`aws_commons.create_aws_credentials` 関数の結果は、`credentials` 関数の aws_s3.table_import_from_s3 パラメータで使用します。

構文

```
aws_commons.create_aws_credentials(
   access_key text,
   secret_key text,
   session_token text
)
```

パラメータ

access_key

Amazon S3 ファイルのインポートに使用するアクセスキーを含む必須のテキスト文字列。デフォルトは NULL です。

secret_key

Amazon S3 ファイルのインポートに使用するシークレットキーを含む必須のテキスト文字列。デフォルトは NULL です。

session_token

Amazon S3 ファイルのインポートに使用するセッショントークンを含む必須のテキスト文字列。デフォルトは NULL です。オプションの `session_token` を指定した場合は、一時的な認証情報を使用することができます。

Aurora PostgreSQL DB クラスターから Amazon S3 へのデータのエクスポート

Aurora PostgreSQL DB クラスター からデータをクエリし、Amazon S3 バケットに保存されているファイルに直接エクスポートできます。これを行うには、Aurora PostgreSQL aws_s3拡張機能を最初にインストールします。このエクステンションでは、Amazon S3 へのクエリの結果のエクスポートに使用する関数が利用できます。次に、拡張機能のインストール方法と Amazon S3 へのデータのエクスポート方法を説明します。

プロビジョニングされた DB インスタンスまたは Aurora Serverless v2 DB インスタンスからエクスポートできます。これらの手順は Aurora Serverless v1 ではサポートされていません。

> **Note**
>
> クロスアカウントでの Amazon S3 はサポートされていません。

現在利用可能な Aurora PostgreSQL のバージョンでは、データの Amazon Simple Storage Service へのエクスポートがサポートされています。詳細なバージョン情報については、「Aurora PostgreSQL リリースノート」の「Amazon Aurora PostgreSQL の更新」を参照してください。

エクスポートにバケットを設定していない場合は、Amazon Simple Storage Service ユーザーガイドで次のトピックを参照してください。

- Amazon S3 のセットアップ
- バケットの作成

デフォルトでは、Aurora PostgreSQL から Amazon S3 にエクスポートされたデータは、AWS マネージドキー によるサーバー側の暗号化が使用されます。また、既に作成したカスタマーマネージドキーを使用することもできます。バケット暗号化を使用している場合は、Amazon S3 バケットは AWS Key Management Service (AWS KMS) キー (SSE-KMS) で暗号化されている必要があります。現在、Amazon S3 マネージドキー (SSE-S3) で暗号化されたバケットはサポートされていません。

> **Note**
>
> AWS Management Console、AWS CLI、または Amazon RDS API を使用して、DB および DB クラスターのスナップショットデータを Amazon S3 に保存できます。詳細については、「Amazon S3 への DB クラススナップショットデータのエクスポート」を参照してください。

トピック

- aws_s3 拡張機能のインストール
- Amazon S3 へのデータのエクスポートの概要
- エクスポート先の Amazon S3 ファイルパスを指定する
- Amazon S3 バケットへのアクセスを設定する
- aws_s3.query_export_to_s3 関数を使用したクエリデータのエクスポート
- Amazon S3 へのアクセスのトラブルシューティング
- 関数リファレンス

aws_s3 拡張機能のインストール

Aurora PostgreSQL DB クラスター で Amazon Simple Storage Service を使用する前に、aws_s3 拡張機能をインストールする必要があります。この拡張機能には、Aurora PostgreSQL DB クラスターのライターインスタンス から Amazon S3 バケットへデータをエクスポートするための関数も含まれています。また、Amazon S3 からデータをインポートするための関数も含まれます。詳しくは、「Amazon S3 から Aurora PostgreSQL DB クラスターにデータをインポートする」を参照してください。aws_s3 拡張機能は aws_commons 拡張機能の一部のヘルパー関数に依存しており、必要に応じて自動的にインストールされます。

aws_s3 拡張機能をインストールするには

1. rds_superuser 権限があるユーザーとして、psql (または pgAdmin) を使用して Aurora PostgreSQL DB クラスターのライターインスタンス に接続します。設定プロセス中にデフォルトの名前を保持している場合は、postgres として接続します。

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password
```

2. 拡張機能をインストールするには、次のコマンドを実行します。

```
postgres=> CREATE EXTENSION aws_s3 CASCADE;
NOTICE: installing required extension "aws_commons"
CREATE EXTENSION
```

3. 拡張機能がインストールされていることを確認するには、psql \dx メタコマンドを使用します。

```
postgres=> \dx
       List of installed extensions
    Name     | Version |   Schema   |                 Description
-------------+---------+------------+---------------------------------------------
 aws_commons | 1.2     | public     | Common data types across AWS services
 aws_s3      | 1.1     | public     | AWS S3 extension for importing data from S3
 plpgsql     | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

Amazon S3 からデータをインポートし、データを Amazon S3 にエクスポートするための関数が使用できるようになりました。

ご使用の Aurora PostgreSQL バージョンで、Amazon S3 へのエクスポートがサポートされていることを確認します

describe-db-engine-versions コマンドを使用して、Aurora PostgreSQL バージョンが Amazon S3 へのエクスポートをサポートしていることを確認できます。次の例では、バージョン 10.14 が Amazon S3 にエクスポートできるかどうかを確認します。

```
aws rds describe-db-engine-versions --region us-east-1 \
--engine aurora-postgresql --engine-version 10.14 | grep s3Export
```

出力に "s3Export" の文字列が含まれている場合 、エンジンは Amazon S3 エクスポートをサポートします。それ以外の場合、エンジンはエクスポートをサポートしません。

Amazon S3 へのデータのエクスポートの概要

Aurora PostgreSQL データベースに格納されたデータを Amazon S3 バケットにエクスポートするには、以下の手順に従います。

Aurora PostgreSQL データを S3 にエクスポートするには

1. データのエクスポートに使用する Amazon S3 ファイルパスを指定します。このプロセスの詳細については、「エクスポート先の Amazon S3 ファイルパスを指定する」を参照してください。
2. Amazon S3 バケットへのアクセス許可を提供します。

   Amazon S3 ファイルにデータをエクスポートするには、Aurora PostgreSQL DB クラスターに、エクスポートの際に保存に使用される Amazon S3 バケットへのアクセス許可を付与する必要があります。これには、次のステップが含まれます。

   1. エクスポート先の Amazon S3 バケットへのアクセスを提供する IAM ポリシーを作成します。
   2. IAM ロールを作成します。
   3. 作成したポリシーを、作成したロールにアタッチします。
   4. この IAM ロールを DB クラスターに追加します。

   このプロセスの詳細については、「Amazon S3 バケットへのアクセスを設定する」を参照してください。
3. データを取得するためのデータベースクエリを識別します。aws_s3.query_export_to_s3 関数を呼び出して、クエリデータをエクスポートします。

   前述の準備タスクを完了したら、aws_s3.query_export_to_s3 関数を使用してクエリ結果を Amazon S3 にエクスポートします。このプロセスの詳細については、「aws_s3.query_export_to_s3 関数を使用したクエリデータのエクスポート」を参照してください。

エクスポート先の Amazon S3 ファイルパスを指定する

次の情報を指定して、Amazon S3 データのエクスポート先となる場所を指定します。

- バケット名 - バケットは、Amazon S3 オブジェクトまたはファイルのコンテナです。

  Amazon S3 を使用したデータの保存の詳細については、Amazon Simple Storage Service ユーザーガイドの「Create a bucket」と「View an object」を参照してください。
- ファイルパス - ファイルパスは、Amazon S3 バケット内のエクスポートが格納される場所を識別します。ファイルパスは、次のもので構成されます。
  - 仮想フォルダパスを識別するオプションのパスプレフィックス。
  - 保存する 1 つ以上のファイルを識別するファイルプレフィックス。より大きなエクスポートは複数のファイルに格納され、それぞれの最大サイズは約 6 GB です。追加のファイル名には、同じファイルプレフィックスが付いていますが、末尾に _part*XX* が付加されます。*XX* は、2、3 などを表します。

  例えば、`exports` フォルダとファイルプレフィックスを持つ `query-1-export` ファイルパスは `/exports/query-1-export` です。
- AWS リージョン (オプション) - Amazon S3 バケットがある AWS リージョン。AWS リージョンの値を指定しない場合、Aurora は、エクスポートする DB クラスターと同じ AWS リージョンの Amazon S3 にファイルを保存します。

  > **Note**
  >
  > 現在、AWS リージョンは、エクスポートする DB クラスターのリージョンと同じである必要があります。

  AWS リージョン名と関連する値のリストについては、「リージョンとアベイラビリティーゾーン」を参照してください。

エクスポートの保存先に関する Amazon S3 ファイル情報を保持するには、aws_commons.create_s3_uri 関数を使用して、次のように `aws_commons._s3_uri_1` 複合構造を作成します。

```
psql=> SELECT aws_commons.create_s3_uri(
   'sample-bucket',
   'sample-filepath',
   'us-west-2'
) AS s3_uri_1 \gset
```

その後、この s3_uri_1 値を aws_s3.query_export_to_s3 関数の呼び出しでパラメータとして指定します。例については、「aws_s3.query_export_to_s3 関数を使用したクエリデータのエクスポート」を参照してください。

Amazon S3 バケットへのアクセスを設定する

データを Amazon S3 にエクスポートするには、PostgreSQL DB クラスターに、ファイルが入る Amazon S3 バケットに対するアクセス許可を付与します。

これには、以下の手順を使用します。

IAM ロールを介して PostgreSQLDB のクラスターに Amazon S3 へのアクセスを許可するには

1. IAM ポリシーを作成します。

   このポリシーは、PostgreSQL DB クラスターに、Amazon S3 のバケットとオブジェクトに対するアクセス許可を付与します。

   このポリシーの作成の一環として、次のステップを実行します。

   a. ポリシーに、PostgreSQL DB クラスターから Amazon S3 バケットへのファイル転送を許可するための以下の必須アクションを含めます。

      - `s3:PutObject`
      - `s3:AbortMultipartUpload`

   b. Amazon S3 バケットとバケット内のオブジェクトを識別する Amazon リソースネーム (ARN) を含めます。Amazon S3 アクセス用の ARN 形式は `arn:aws:s3:::`*`your-s3-bucket`*`/*` です。

   Aurora PostgreSQL の IAM ポリシーの作成の詳細については、IAM データベースアクセス用の IAM ポリシーの作成と使用 を参照してください。IAM ユーザーガイドの「チュートリアル: はじめてのカスタマー管理ポリシーの作成とアタッチ」も参照してください。

   以下の AWS CLI コマンドでは、これらのオプションを指定して、`rds-s3-export-policy` という名前の IAM ポリシーを作成します。このポリシーでは、`your-s3-bucket` という名前のバケットへのアクセス権が付与されます。

> **Warning**
>
> 特定のバケットにアクセスするようにエンドポイントポリシーが設定されているプライベート VPC 内にデータベースをセットアップすることをお勧めします。詳細については、Amazon VPC ユーザーガイドの「Amazon S3 のエンドポイントポリシーの使用」を参照してください。
> すべてのリソースへのアクセスを持つポリシーを作成しないことを強くお勧めします。このアクセスは、データセキュリティにとって脅威になる可能性があります。S3:PutObject を使用してすべてのリソースへのアクセスを "Resource":"*" に許可するポリシーを作成すると、エクスポート権限を持つユーザーはアカウント内のすべてのバケットにデータをエクスポートできます。さらに、ユーザーは AWS リージョン内のパブリックに書き込み可能なバケットにデータをエクスポートできます。

ポリシーを作成したら、そのポリシーの Amazon リソースネーム (ARN) を書き留めます。ポリシーを IAM ロールにアタッチする場合、後続のステップで ARN が必要です。

```
aws iam create-policy  --policy-name rds-s3-export-policy  --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3export",
         "Action": [
           "s3:PutObject",
           "s3:AbortMultipartUpload"
         ],
         "Effect": "Allow",
         "Resource": [
           "arn:aws:s3:::your-s3-bucket/*"
         ]
       }
     ]
   }'
```

2. IAM ロールを作成します。

これを行うと、Aurora PostgreSQL がユーザーに代わってこの IAM ロールを引き受け、Amazon S3 バケットにアクセスできます。詳細については、IAM ユーザーガイドの「IAM ユーザーにアクセス許可を委任するロールの作成」を参照してください。

リソースポリシー内では aws:SourceArn および aws:SourceAccount のグローバル条件コンテキストキーを使用して、サービスに付与するリソースへのアクセス許可を制限することをお勧めします。これは、混乱した使節の問題に対する最も効果的な保護方法です。

グローバル条件コンテキストキーの両方を使用し、aws:SourceArn の値にアカウント ID が含まれている場合、同じポリシーステートメントで使用する場合は、aws:SourceArn の値と aws:SourceAccount の値のアカウントでは同じアカウント ID を使用する必要があります。

- 単一リソースに対するクロスサービスアクセスが必要な場合は aws:SourceArn を使用します。
- そのアカウント内の任意のリソースをクロスサービス使用に関連付けることを許可する場合、aws:SourceAccountを使用します。

ポリシーでは、必ずリソースの完全な ARN を持つ aws:SourceArn グローバル条件コンテキストキーを使用してください。以下の例は、AWS CLI コマンドを使用して、rds-s3-export-role という名前のロールを作成する方法を示しています。

Example

Linux、macOS、Unix の場合:

```
aws iam create-role  \
    --role-name rds-s3-export-role  \
    --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-east-1:111122223333:db:dbname"
                }
             }
       }
     ]
```

```
   }'
```

Windows の場合:

```
aws iam create-role  ^
    --role-name rds-s3-export-role  ^
    --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-east-1:111122223333:db:dbname"
                }
             }
       }
     ]
   }'
```

3. 作成した IAM ポリシーを、作成した IAM ロールにアタッチします。

   次の AWS CLI コマンドは、先ほど作成したポリシーを `rds-s3-export-role` という名前のロールにアタッチします。*`your-policy-arn`* を前のステップでメモしたポリシー ARN に置き換えます。

   ```
   aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-
   export-role
   ```

4. DB クラスターに IAM ロールを追加します。これを行うには、以下で説明するように、AWS Management Console または AWS CLI を使用します。

コンソール

コンソールを使用して PostgreSQL DB クラスターの IAM ロールを追加するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. 詳細を表示するには、PostgreSQL DB クラスターの名前を選択します。
3. [接続とセキュリティ] タブの [IAM ロールの管理] セクションで、[このインスタンスに IAM ロールを追加] で追加するロールを選択します。
4. [Feature] で、[s3Export] を選択します。
5. [Add role] を選択します。

AWS CLI

CLI を使用して PostgreSQL DB クラスターの IAM ロールを追加するには

- 次のコマンドを使用して、`my-db-cluster` という名前の PostgreSQL DB クラスターにロールを追加します。*`your-role-arn`* を、以前のステップで書き留めたロール ARN に置き換えます。`s3Export` オプションの値に `--feature-name` を使用します。

  Example

  Linux、macOS、Unix の場合:

  ```
  aws rds add-role-to-db-cluster \
     --db-cluster-identifier my-db-cluster \
     --feature-name s3Export \
     --role-arn your-role-arn   \
     --region your-region
  ```

  Windows の場合:

  ```
  aws rds add-role-to-db-cluster ^
     --db-cluster-identifier my-db-cluster ^
     --feature-name s3Export ^
     --role-arn your-role-arn ^
     --region your-region
  ```

aws_s3.query_export_to_s3 関数を使用したクエリデータのエクスポート

aws_s3.query_export_to_s3 関数を呼び出して、PostgreSQL データを Amazon S3 にエクスポートします。

トピック

前提条件

aws_s3.query_export_to_s3 関数を使用する前に、以下の前提条件を満たしていることを確認してください。

- 「Amazon S3 へのデータのエクスポートの概要」の説明に従って、必要な PostgreSQL エクステンションをインストールします。
- 「エクスポート先の Amazon S3 ファイルパスを指定する。」の説明に従って、データの Amazon S3 のエクスポート先を決定します。
- 「Amazon S3 バケットへのアクセスを設定する」の説明にとおり、DB クラスターが Amazon S3 へのエクスポートアクセス権があることを確認します。

次の例では、sample_table というデータベーステーブルを使用しています。次の例では、データを sample-bucket というバケットにエクスポートします。サンプルのテーブルとデータは、psql で次の SQL ステートメントを使用して作成されます。

```
psql=> CREATE TABLE sample_table (bid bigint PRIMARY KEY, name varchar(80));
psql=> INSERT INTO sample_table (bid,name) VALUES (1, 'Monday'), (2,'Tuesday'), (3,
 'Wednesday');
```

aws_s3.query_export_to_s3 の呼び出し

次に、 aws_s3.query_export_to_s3 関数を呼び出す基本的な方法を示します。

これらの例では、可変 s3_uri_1 を使用して、Amazon S3 ファイルを識別する情報を含む構造を指定しています。aws_commons.create_s3_uri 関数を使用して構造を作成します。

```
psql=> SELECT aws_commons.create_s3_uri(
   'sample-bucket',
   'sample-filepath',
   'us-west-2'
) AS s3_uri_1 \gset
```

以下の 2 つの aws_s3.query_export_to_s3 関数呼び出しのパラメータは異なりますが、これらの例の結果は同じです。sample_table テーブルのすべての行が sample-bucket というバケットにエクスポートされます。

```
psql=> SELECT * FROM aws_s3.query_export_to_s3('SELECT * FROM
 sample_table', :'s3_uri_1');

psql=> SELECT * FROM aws_s3.query_export_to_s3('SELECT * FROM
 sample_table', :'s3_uri_1', options :='format text');
```

パラメータの説明は次のとおりです。

- 'SELECT * FROM sample_table' - 初期のパラメータは、SQL クエリを含む必須のテキスト文字列です。PostgreSQL エンジンはこのクエリを実行します。クエリの結果は、他のパラメータで指定された S3 バケットにコピーされます。
- :'s3_uri_1' - このパラメータは、Amazon S3 ファイルを識別する構造です。この例では、可変を使用して、前に作成した構造を指定します。代わりに、以下のように aws_commons.create_s3_uri 関数呼び出し内にインラインで aws_s3.query_export_to_s3 関数呼び出しを含めることで、同じ構造を作成できます。

  ```
  SELECT * from aws_s3.query_export_to_s3('select * from sample_table',
     aws_commons.create_s3_uri('sample-bucket', 'sample-filepath', 'us-west-2')
  );
  ```

- options :='format text' - options パラメータは、PostgreSQL COPY 引数を含むオプションのテキスト文字列です。このコピープロセスでは、PostgreSQL COPY コマンドの引数と形式を使用します。

指定したファイルが Amazon S3 バケットに存在しない場合は、作成されます。このファイルが存在している場合は、上書きされます。Amazon S3 でエクスポートされたデータにアクセスするための構文は次のとおりです。

```
s3-region://bucket-name[/path-prefix]/file-prefix
```

より大きなエクスポートは複数のファイルに格納され、それぞれの最大サイズは約 6 GB です。追加のファイル名には、同じファイルプレフィックスが付いていますが、末尾に _part*XX* が付加されます。*XX* は、2、3 などを表します。例えば、次のようにデータファイルを格納するパスを指定するとします。

```
s3-us-west-2://my-bucket/my-prefix
```

エクスポートで 3 つのデータファイルを作成する必要がある場合、Amazon S3 バケットには次のデータファイルが含まれます。

```
s3-us-west-2://my-bucket/my-prefix
s3-us-west-2://my-bucket/my-prefix_part2
s3-us-west-2://my-bucket/my-prefix_part3
```

この関数の完全なリファレンスと、それを呼び出すその他の方法については、「aws_s3.query_export_to_s3」を参照してください。Amazon S3 でファイルにアクセスする方法の詳細については、Amazon Simple Storage Service ユーザーガイドの「View an object」を参照してください。

カスタム区切り文字を使用する CSV ファイルへのエクスポート

次の例は、aws_s3.query_export_to_s3 関数を呼び出して、カスタム区切り文字を使用するファイルにデータをエクスポートする方法を示しています。この例では、PostgreSQL COPY コマンドの引数を使用して、カンマ区切り値 (CSV) 形式とコロン (:) 区切り文字を指定します。

```
SELECT * from aws_s3.query_export_to_s3('select * from basic_test', :'s3_uri_1',
 options :='format csv, delimiter $$:$$');
```

エンコードを使用したバイナリファイルへのエクスポート

次の例は、aws_s3.query_export_to_s3 関数を呼び出して、Windows-1253 エンコーディングのバイナリファイルにデータをエクスポートする方法を示しています。

```
SELECT * from aws_s3.query_export_to_s3('select * from basic_test', :'s3_uri_1',
 options :='format binary, encoding WIN1253');
```

Amazon S3 へのアクセスのトラブルシューティング

Amazon S3 へのデータのエクスポート試行時に接続の問題が発生した場合は、まず DB インスタンスに関連付けられた VPC セキュリティグループのアウトバウンドアクセスルールがネットワーク接続を許可していることを確認します。具体的には、DB インスタンスにポート 443 および任意の IPv4 アドレス (0.0.0.0/0) への TCP トラフィックの送信を許可するルールをセキュリティグループに作成します。詳細については、「セキュリティグループを作成して VPC 内の DB クラスターへのアクセスを提供する」を参照してください。

推奨事項については、以下も参照してください。

- Amazon Aurora のアイデンティティおよびアクセスのトラブルシューティング
- Amazon Simple Storage Service ユーザーガイド の「Troubleshooting Amazon S3」
- IAM ユーザーガイド の Amazon S3 のトラブルシューティングと IAM

関数リファレンス

関数

- aws_s3.query_export_to_s3
- aws_commons.create_s3_uri

aws_s3.query_export_to_s3

PostgreSQL クエリ結果を Amazon S3 バケットにエクスポートします。aws_s3 エクステンションには、aws_s3.query_export_to_s3 関数が含まれます。

2 つの必須パラメータは、query および s3_info です。これらは、エクスポートするクエリを定義し、エクスポート先の Amazon S3 バケットを特定します。options と呼ばれるオプションのパラメータは、さまざまなエクスポートパラメータを定義するために用意されています。aws_s3.query_export_to_s3 関数の使用例については、「aws_s3.query_export_to_s3 関数を使用したクエリデータのエクスポート」を参照してください。

[Syntax] (構文)

```
aws_s3.query_export_to_s3(
    query text,
    s3_info aws_commons._s3_uri_1,
    options text,
```

```
    kms_key text
)
```

入力パラメータ

query

PostgreSQL エンジンが実行する SQL クエリを含む必須のテキスト文字列。このクエリ結果は、`s3_info` パラメータで指定された S3 バケットにコピーされます。

s3_info

S3 オブジェクトに関する以下の情報を含む `aws_commons._s3_uri_1` 複合型。

- `bucket` - ファイルを格納する Amazon S3 バケットの名前。
- `file_path` - Amazon S3 ファイル名とパス
- `region` - バケットが存在する AWS リージョン。AWS リージョン名と関連する値のリストについては、「リージョンとアベイラビリティーゾーン」を参照してください。

  現在、この値は、エクスポートする DB クラスターの AWS リージョンと同じリージョンである必要があります。デフォルトは、エクスポートする DB クラスターの AWS リージョンです。

`aws_commons._s3_uri_1` 複合構造を作成するには、aws_commons.create_s3_uri 関数を参照してください。

options:

PostgreSQL `COPY` コマンドの引数を含むオプションのテキスト文字列。これらの引数は、エクスポート時のデータのコピー方法を指定します。詳細については、「PostgreSQL COPY ドキュメント」を参照してください。

kms_key text

データのエクスポート先となる S3 バケットのカスタマーマネージド KMS キーを含む任意のテキスト文字列。

代替入力パラメータ

テストしやすいように、`s3_info` パラメータではなく、拡張されたパラメータセットを使用することができます。以下は、`aws_s3.query_export_to_s3` 関数の構文のバリエーションです。

Amazon S3 ファイルを識別するために s3_info パラメータを使用する代わりに、bucket、file_path、および region パラメータの組み合わせを使用します。

```
aws_s3.query_export_to_s3(
    query text,
    bucket text,
    file_path text,
    region text,
    options text,
    kms_key text
)
```

query

PostgreSQL エンジンが実行する SQL クエリを含む必須のテキスト文字列。このクエリ結果は、s3_info パラメータで指定された S3 バケットにコピーされます。

bucket (バケット)

ファイルを含む Amazon S3 バケットの名前を含む必須テキスト文字列。

file_path

ファイルのパスを含む Amazon S3 ファイル名を含む必須テキスト文字列。

region

バケットが存在する AWS リージョンを含むオプションのテキスト文字列。AWS リージョン名と関連する値のリストについては、「リージョンとアベイラビリティーゾーン」を参照してください。

現在、この値は、エクスポートする DB クラスターの AWS リージョンと同じリージョンである必要があります。デフォルトは、エクスポートする DB クラスターの AWS リージョンです。

options:

PostgreSQL COPY コマンドの引数を含むオプションのテキスト文字列。これらの引数は、エクスポート時のデータのコピー方法を指定します。詳細については、「PostgreSQL COPY ドキュメント」を参照してください。

kms_key text

データのエクスポート先となる S3 バケットのカスタマーマネージド KMS キーを含む任意のテキスト文字列。

出力パラメータ

```
aws_s3.query_export_to_s3(
    OUT rows_uploaded bigint,
    OUT files_uploaded bigint,
    OUT bytes_uploaded bigint
)
```

rows_uploaded

指定されたクエリで Amazon S3 に正常にアップロードされたテーブルローの数。

files_uploaded

Amazon S3 にアップロードされたファイルの数。ファイルは、約 6 GB のサイズで作成されます。作成される各追加ファイルは、名前に _part*XX* が付加されています。*XX* は、必要に応じて 2、3 などを表します。

bytes_uploaded

Amazon S3 にアップロードされた合計バイト数。

例

```
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'sample-
bucket', 'sample-filepath');
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'sample-
bucket', 'sample-filepath','us-west-2');
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'sample-
bucket', 'sample-filepath','us-west-2','format text');
```

aws_commons.create_s3_uri

Amazon S3 ファイル情報を保持するように、aws_commons._s3_uri_1 構造を作成します。aws_commons.create_s3_uri 関数の結果は、s3_info 関数の aws_s3.query_export_to_s3 パラメータで使用します。aws_commons.create_s3_uri 関数の使用例については、「エクスポート先の Amazon S3 ファイルパスを指定する」を参照してください。

Syntax

```
aws_commons.create_s3_uri(
```

```
    bucket text,
    file_path text,
    region text
)
```

入力パラメータ

bucket (バケット)

ファイルの Amazon S3 バケット名を含む必須のテキスト文字列。

file_path

ファイルのパスを含む Amazon S3 ファイル名を含む必須テキスト文字列。

region

ファイルがある AWS リージョンを含む必須のテキスト文字列。AWS リージョン名と関連する値のリストについては、「リージョンとアベイラビリティーゾーン」を参照してください。

Aurora PostgreSQL DB クラスターから AWS Lambda 関数を呼び出す

AWS Lambda は、サーバーのプロビジョニングや管理を行わなくてもコードの実行が可能な、イベント駆動型のコンピューティングサービスです。この機能は、Aurora PostgreSQL を含む多くの AWS サービスで利用可能です。例えば、データベースからのイベント通知の処理や、新しいファイルが Amazon S3 にアップロードされるたびに行うファイルからのデータロードのために、Lambda を使用することができます。詳細については、「AWS Lambda デベロッパーガイドの「AWS Lambda とは」を参照してください。

> **Note**
>
> AWS Lambda 関数の呼び出しは、Aurora PostgreSQL 11.9 以上のバージョン (Aurora Serverless v2 を含む) でサポートされています。

Aurora PostgreSQL で Lambda 関数を操作するためのセットアップは、AWS Lambda、IAM、VPC、および Aurora PostgreSQL DB クラスターが関係する複数ステップのプロセスとなります。以下に、必要なステップの概要を示します。

Lambda 関数の詳細については、「AWS Lambda デベロッパーガイド」の「Lambda の開始方法」および「AWS Lambda の基礎」を参照してください。

トピック

ステップ 1: Aurora PostgreSQL DB クラスターで、AWS Lambda へのアウトバウンド接続を設定する。

Lambda 関数は、常に AWS Lambda サービスが所有する Amazon VPC 内で実行されます。Lambda はこの VPC にネットワークアクセスとセキュリティルールを適用し、この VPC を自動的にモニタリングおよび維持します。Aurora PostgreSQL DB クラスターは、Lambda サービスの VPC にネットワークトラフィックを送信します。このための構成方法は、Aurora DB クラスターのプライマリDB インスタンスが、パブリックであるかプライベートであるかにより異なります。

- パブリック Aurora PostgreSQL DB クラスター — VPC のパブリックサブネット内に置かれた DB クラスターのプライマリ DB インスタンスで、「PubliclyAccessible」プロパティに `true` が設定されている場合、そのインスタンスはパブリックです。このプロパティの値は、AWS CLI コマンド describe-db-instances を使用して確認できます。または、AWS Management Console を使用して [Connectivity & security] (接続とセキュリティ) タブを開き、[Publicly accessible] (パブリックアクセス可能) が「はい」となっているかを確認します。インスタンスが VPC のパブリックサブネット内に置かれていることを確認するには、AWS Management Console または AWS CLI を使用します。

  Lambda へのアクセスを設定するには、AWS Management Console または AWS CLI を使用して、VPC のセキュリティグループでアウトバウンドルールを作成します。アウトバウンドルールでは、TCP がポート 443 を使用して任意の IPv4 アドレス (0.0.0.0/0) にパケットを送信するように定義しています。

- プライベートAurora PostgreSQL DB クラスター — この例では、インスタンスの「PubliclyAccessible」プロパティが `false` に指定されているか、インスタンスがプライベートサブネット内に置かれています。インスタンスが Lambda で動作できるようにするには、ネットワークアドレス変換 (NAT) ゲートウェイを使用します。詳細については、「NAT ゲートウェイ」を参照してください。または、VPC で Lambda の VPC エンドポイントを設定できます。詳細については、Amazon VPC ユーザーガイドの「VPC エンドポイント」を参照してください。このエンドポイントは、Aurora PostgreSQL DB クラスターが Lambda 関数に対して発行した、呼び出しに対して応答します。

ご使用の VPC は、ネットワークレベルで AWS Lambda VPC とやり取りできるようになります。次に、IAM を使用してアクセス権限を設定します。

ステップ 2: Aurora PostgreSQL DB クラスターおよび AWS Lambda のために IAM を設定する

Aurora PostgreSQL DB クラスターからの Lambda 関数の呼び出しには、特定の権限が必要です。必要な権限を設定するには、Lambda 関数の呼び出しを許可する IAM ポリシーを作成し、そのポリシーをロールに割り当てた上で、そのロールを DB クラスターに適用することをお勧めします。このアプローチでは、指定された Lambda 関数をユーザーに代わって呼び出すための権限を、DB クラスターに対し付与します。以下のステップで、AWS CLI を使用してこれを行う方法を示します。

クラスターで Lambda を使用するために IAM のアクセス許可を設定するには

1. AWS CLI コマンド create-policy を実行して、指定された Lambda 関数を、Aurora PostgreSQL DB クラスターが呼びだすことを許可する、IAM ポリシーを作成します。(ステートメント ID (Sid) は、ポリシーステートメントのオプションの記述であり、使用には影響しません。) このポリシーは、Aurora DB クラスターに対し、指定された Lambda 関数を呼び出すための最小限のアクセス許可を付与します。

```
aws iam create-policy  --policy-name rds-lambda-policy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Sid": "AllowAccessToExampleFunction",
        "Effect": "Allow",
        "Action": "lambda:InvokeFunction",
        "Resource": "arn:aws:lambda:aws-region:444455556666:function:my-function"
        }
    ]
```

```
}'
```

または、任意の Lambda 関数の呼び出しを許可する、事前定義済みの AWSLambdaRole ポリシーを使用することもできます。詳細については、「Lambda のアイデンティティベースの IAM ポリシー」を参照してください。

2. AWS CLI コマンド create-role を使用して、実行時にポリシーが引き受けることができる IAM ロールを作成します。

```
aws iam create-role  --role-name rds-lambda-role --assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Principal": {
            "Service": "rds.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
        }
    ]
}'
```

3. AWS CLI コマンド attach-role-policy を使用して、このポリシーをロールに適用します。

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::444455556666:policy/rds-lambda-policy \
    --role-name rds-lambda-role --region aws-region
```

4. AWS CLI コマンド add-role-to-db-cluster を使用して、このロールを Aurora PostgreSQL DB クラスターに適用します。この最後のステップにより、DB クラスターのデータベースユーザーに対し、Lambda 関数の呼び出しを許可します。

```
aws rds add-role-to-db-cluster \
       --db-cluster-identifier my-cluster-name \
       --feature-name Lambda \
       --role-arn  arn:aws:iam::444455556666:role/rds-lambda-role   \
       --region aws-region
```

VPC と IAM の設定が完了したので、ここで aws_lambda 拡張をインストールできます。(拡張機能は任意のタイミングでインストールできますが、先に VPC サポートと IAM 権限を適切に設定する必

要があります。aws_lambda 拡張機能は、Aurora PostgreSQL DB クラスターの機能に対し何も追加しません。)

ステップ 3: Aurora PostgreSQL DB クラスター用に **aws_lambda** 拡張機能をインストールする

Aurora PostgreSQL DB クラスターで AWS Lambda を使用し、Aurora PostgreSQL DB クラスターに対し aws_lambda PostgreSQL 拡張機能を追加します。この拡張機能は、Aurora PostgreSQL DB クラスターに対し、PostgreSQL からの Lambda 関数呼び出し機能を追加します。

Aurora PostgreSQL DB クラスターに **aws_lambda** 拡張機能をインストールするには

PostgreSQL の psql コマンドライン、または pgAdmin ツールを使用して、Aurora PostgreSQL DB クラスターに接続します。

1. Aurora PostgreSQL DB クラスターに、rds_superuser 権限を持つユーザーとして接続します。例では、デフォルトの postgres ユーザが示されています。

   ```
   psql -h cluster-instance.444455556666.aws-region.rds.amazonaws.com -U postgres -p
    5432
   ```

2. aws_lambda 拡張機能をインストールします。aws_commons 拡張機能も必要です。これは、aws_lambda や、他の多数の PostgreSQL向け Aurora 拡張機能にヘルパー関数を提供します。この拡張機能が、Aurora PostgreSQL DB クラスター上で見つからない場合は、次のように aws_lambda を使用してインストールされています。

   ```
   CREATE EXTENSION IF NOT EXISTS aws_lambda CASCADE;
   NOTICE:  installing required extension "aws_commons"
   CREATE EXTENSION
   ```

aws_lambda 拡張機能は、Aurora PostgreSQL DB クラスターのプライマリ DB インスタンスにインストールされています。この段階で、Lambda 関数を呼び出すための、使いやすい構造を作成することが可能です。

ステップ 4: Aurora PostgreSQL DB クラスターで Lambda のヘルパー関数を使用する (オプション)

aws_commons 拡張機能のヘルパー関数を使用すると、PostgreSQL からより簡単に呼び出すことができるエンティティを準備することができます。これを行うには、Lambda 関数に関する以下の情報が必要です。

- [Function name] (関数名) – Lambda 関数の名前、Amazon リソースネーム (ARN)、バージョンまたはエイリアス。ステップ 2: クラスターおよび Lambda のために IAM を設定する で作成された IAM ポリシーは ARN を必要とするため、関数の ARN を使用することをお勧めします。
- [AWS Region] (リージョン) – (オプション) Lambda 関数が Aurora PostgreSQL DB クラスターと同じリージョンに存在しない場合の、Lambda 関数が置かれている AWS リージョン。

Lambda 関数名の情報を保持するには、aws_commons.create_lambda_function_arn 関数を使用します。このヘルパー関数は、呼び出し関数に必要な詳細を含む aws_commons._lambda_function_arn_1 複合構造を作成します。以下に、この複合構造を設定するための 3 つの代替手段を説明します。

```
SELECT aws_commons.create_lambda_function_arn(
   'my-function',
   'aws-region'
) AS aws_lambda_arn_1 \gset
```

```
SELECT aws_commons.create_lambda_function_arn(
   '111122223333:function:my-function',
   'aws-region'
) AS lambda_partial_arn_1 \gset
```

```
SELECT aws_commons.create_lambda_function_arn(
   'arn:aws:lambda:aws-region:111122223333:function:my-function'
) AS lambda_arn_1 \gset
```

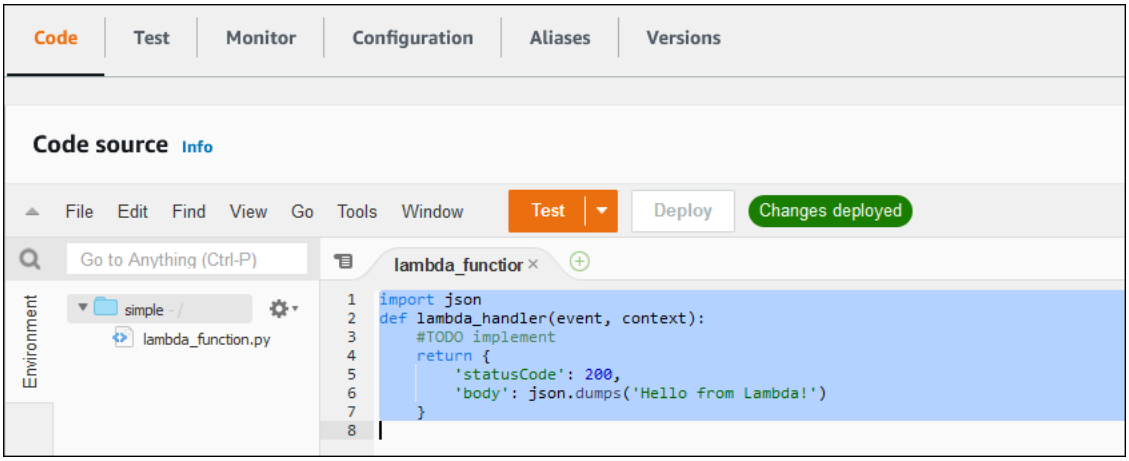
これらの値はいずれも、aws_lambda.invoke 関数の呼び出し時に使用されます。例については、「ステップ 5: Aurora PostgreSQL DB クラスターから Lambda 関数を呼びだす」を参照してください。

ステップ 5: Aurora PostgreSQL DB クラスターから Lambda 関数を呼びだす

aws_lambda.invoke 関数は、invocation_type に応じて同期または非同期的で動作します。以下のように、このパラメーターには 2 つの選択肢、RequestResponse (デフォルト) と Event があります。

- **RequestResponse** – この呼び出しタイプは同期です。これは、呼び出しタイプを指定せずに呼び出しが行われた場合のデフォルトの動作です。レスポンスペイロードには、aws_lambda.invoke 関数の結果が含まれます。処理を続行する前に Lambda 関数から結果を受け取る必要があるワークフローの場合は、この呼び出しタイプを使用します。
- **Event** – この呼び出しタイプは非同期です。この場合の応答には、結果を含むペイロードは含まれません。この呼び出しタイプは、処理を続行するために Lambda 関数の結果を必要としないワークフローで使用します。

セットアップの簡単なテストとして、psql を使用して DB インスタンスに接続し、コマンドラインからサンプル関数を起動します。今、次のスクリーンショットに示すシンプルな Python 関数のような基本的関数の 1 つが、Lambda サービスに設定されているとします。



サンプル関数を呼び出すには

1. psql または pgAdmin を使用して、プライマリ DB インスタンスに接続します。

```
psql -h cluster.444455556666.aws-region.rds.amazonaws.com -U postgres -p 5432
```

2. ARN を使用して関数を呼びだします。

```
SELECT * from
 aws_lambda.invoke(aws_commons.create_lambda_function_arn('arn:aws:lambda:aws-
```

```
region:444455556666:function:simple', 'us-west-1'), '{"body": "Hello from
 Postgres!"}'::json );
```

この応答は次のようになります。

```
status_code |                         payload                          |
 executed_version | log_result
-------------+----------------------------------------------------------
+------------------+------------
         200 | {"statusCode": 200, "body": "\"Hello from Lambda!\""} | $LATEST
     |
(1 row)
```

呼び出しが成功しなかった場合は、「Lambda 関数のエラーメッセージ」を参照してください。

ステップ 6: Lambda 関数を呼び出すその他のユーザー許可を付与する

手順のこの時点で、rds_superuser であるユーザーだけが Lambda 関数を呼び出すことができます。作成した関数の呼び出しを他のユーザーに許可するには、許可を付与する必要があります。

Lambda 関数を呼び出すアクセス許可を付与するには

1. psql または pgAdmin を使用して、プライマリ DB インスタンスに接続します。

```
psql -h cluster.444455556666.aws-region.rds.amazonaws.com -U postgres -p 5432
```

2. 次の SQL コマンドを実行します。

```
postgres=>  GRANT USAGE ON SCHEMA aws_lambda TO db_username;
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA aws_lambda TO db_username;
```

例: Aurora PostgreSQL DB クラスターから Lambda 関数を呼びだす

以下に、aws_lambda.invoke 関数の呼び出し例をいくつか示します。ほとんどの例では、関数の詳細を簡単に渡せるように、ステップ 4: Aurora PostgreSQL DB クラスターで Lambda のヘルパー関数を使用する (オプション) で作成した複合構造 aws_lambda_arn_1 を使用しています。非同期呼び出しの例については、「例: Lambda 関数の (Event による) 非同期呼び出し」を参照してください。ここに示されたその他の例はすべて、同期呼び出しを使用します。

Lambda 呼び出しタイプの詳細については、「AWS Lambdaデベロッパーガイド」の「Lambda 関数を呼び出す」を参照してください。aws_lambda_arn_1の詳細については、「aws_commons.create_lambda_function_arn」を参照してください。

サンプルリスト

- 例: Lambda 関数の (RequestResponse による) 同期呼び出し
- 例: Lambda 関数の (Event による) 非同期呼び出し
- 例: 関数レスポンスからの Lambda 実行ログのキャプチャリング
- 例: Lambda 関数にクライアントコンテキストを含める
- 例: Lambda 関数の特定のバージョンの呼び出し

例: Lambda 関数の (RequestResponse による) 同期呼び出し

以下に、Lambda 関数の同期呼び出しの例を 2 つ示します。これらの aws_lambda.invoke 関数呼び出しの結果は同じです。

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json);
```

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'RequestResponse');
```

パラメータの説明は次のとおりです。

- :'aws_lambda_arn_1' – このパラメータは、ヘルパー関数 aws_commons.create_lambda_function_arn を使用して、ステップ 4: Aurora PostgreSQL DB クラスターで Lambda のヘルパー関数を使用する (オプション) で作成される複合構造を識別します。この構造は、次のように aws_lambda.invoke 呼び出しの中で、インラインで作成することもできます。

  ```
  SELECT * FROM aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-function',
   'aws-region'),
  '{"body": "Hello from Postgres!"}'::json
  );
  ```

- '{"body": "Hello from PostgreSQL!"}'::json - Lambda関数に渡す JSON ペイロード。
- 'RequestResponse'-Lambda 呼び出しタイプ。

例: Lambda 関数の (Event による) 非同期呼び出し

以下は、Lambda 関数の非同期呼び出しの例です。Event 呼び出しタイプは、指定された入力ペイロードを使用して Lambda 関数の呼び出しをスケジュールし、すぐに返します。Lambda 関数の結果に依存しない特定のワークフローでは、Event 呼び出しタイプを使用します。

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'Event');
```

例: 関数レスポンスからの Lambda 実行ログのキャプチャリング

関数レスポンスに実行ログの最後の 4 KB を含めるには、log_type パラメーターを使用しながら aws_lambda.invoke 関数を呼び出します。デフォルトでは、このパラメータには None が設定されています。レスポンス内の Lambda 実行ログの結果をキャプチャする場合は、以下のように Tail を指定します。

```
SELECT *, select convert_from(decode(log_result, 'base64'), 'utf-8') as log FROM
 aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"}'::json,
 'RequestResponse', 'Tail');
```

aws_lambda.invoke 関数の log_type パラメータを Tail に設定して、実行ログをレスポンスに含めます。この log_type パラメータのデフォルト値は None です。

返された log_result は、base64 エンコードされた文字列です。このコンテンツは、decode と convert_from PostgreSQL 関数の組み合わせを使用してデコードできます。

log_type の詳細については、「aws_lambda.invoke」を参照してください。

例: Lambda 関数にクライアントコンテキストを含める

aws_lambda.invoke 関数では、次に示すとおり context パラメータを使用して、ペイロードとは別の情報を渡すことができます。

```
SELECT *, convert_from(decode(log_result, 'base64'), 'utf-8') as log FROM
 aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"}'::json,
 'RequestResponse', 'Tail');
```

クライアントコンテキストを含めるときは、aws_lambda.invoke 関数の context パラメータに JSON オブジェクトを使用します。

context パラメータの詳細については、「aws_lambda.invoke」でリファレンスを参照してください。

例: Lambda 関数の特定のバージョンの呼び出し

aws_lambda.invoke 呼び出しに qualifier パラメータを含めることで、Lambda 関数の特定のバージョンを指定することが可能です。以下は、'*custom_version*' をバージョンのエイリアスに使用してこれを行う場合の例です。

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'RequestResponse', 'None', NULL, 'custom_version');
```

代わりに、Lambda 関数名の詳細により、次のように関数の修飾子を指定することもできます。

```
SELECT * FROM aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-
function:custom_version', 'us-west-2'),
'{"body": "Hello from Postgres!"}'::json);
```

qualifier および他のパラメータの詳細については、「aws_lambda.invoke」でリファレンスを参照してください。

Lambda 関数のエラーメッセージ

次のリストには、エラーメッセージに関する情報と、考えられる原因と解決策が表示されます。

- VPC 設定の問題

  VPC の設定の問題により、接続しようとすると次のエラーメッセージが表示されることがあります。

  ```
  ERROR:  invoke API failed
  DETAIL: AWS Lambda client returned 'Unable to connect to endpoint'.
  CONTEXT:  SQL function "invoke" statement 1
  ```

  このエラーの一般的な原因は、VPC セキュリティグループが不適切に設定されていることです。VPC セキュリティグループのポート 443 で TCP のアウトバウンドルールが開いており、VPC が Lambda VPC に接続できるようになっていることを確認します。

- Lambda 関数を呼び出すために必要な許可がない

次のいずれかのエラーメッセージが表示された場合、関数を呼び出すユーザー (ロール) に適切な許可がありません。

```
ERROR:  permission denied for schema aws_lambda
```

```
ERROR:  permission denied for function invoke
```

Lambda 関数を呼び出すには、ユーザー (ロール) に特定の許可を付与する必要があります。詳しくは、「ステップ 6: Lambda 関数を呼び出すその他のユーザー許可を付与する」を参照してください。

- Lambda 関数でのエラーの不適切な処理

  リクエストの処理中に Lambda 関数が例外をスローした場合、`aws_lambda.invoke` は、次のように PostgreSQL エラーで失敗します。

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json);
ERROR:  lambda invocation failed
DETAIL:  "arn:aws:lambda:us-west-2:555555555555:function:my-function" returned error
 "Unhandled", details: "<Error details string>".
```

  Lambda 関数または PostgreSQL アプリケーションの中でエラーに対処します。

AWS Lambda 関数とパラメータのリファレンス

以下は、Aurora PostgreSQL で Lambda 関数を呼び出すために使用する関数とパラメータのリファレンスです。

関数とパラメータ

- aws_lambda.invoke
- aws_commons.create_lambda_function_arn
- aws_lambda パラメータ

aws_lambda.invoke

Aurora PostgreSQL DB クラスターの の Lambda 関数を実行します。

Lambda関数の呼び出しの詳細については、AWS Lambda デベロッパーガイドの「呼び出し」も参照してください。

Syntax

JSON

```
aws_lambda.invoke(
IN function_name TEXT,
IN payload JSON,
IN region TEXT DEFAULT NULL,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSON DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSON,
OUT executed_version TEXT,
OUT log_result TEXT)
```

```
aws_lambda.invoke(
IN function_name aws_commons._lambda_function_arn_1,
IN payload JSON,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSON DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSON,
OUT executed_version TEXT,
OUT log_result TEXT)
```

JSONB

```
aws_lambda.invoke(
IN function_name TEXT,
IN payload JSONB,
IN region TEXT DEFAULT NULL,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSONB DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
```

```
OUT status_code INT,
OUT payload JSONB,
OUT executed_version TEXT,
OUT log_result TEXT)
```

```
aws_lambda.invoke(
IN function_name aws_commons._lambda_function_arn_1,
IN payload JSONB,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSONB DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSONB,
OUT executed_version TEXT,
OUT log_result TEXT
)
```

入力パラメータ

function_name

Lambda 関数の識別名。値には、関数名、ARN、または部分的な ARN を指定できます。可能な形式のリストについては、AWS Lambda デベロッパーガイドの「Lambda関数名の形式」を参照してください。

payload

Lambda 関数の入力。形式には、JSON または JSONB を使用できます。詳細については、PostgreSQL ドキュメントの「JSON タイプ」を参照してください。

リージョン

(オプション) 関数の Lambda リージョン。デフォルトでは、Aurora は AWS の完全な ARN から `function_name` リージョンを解決するか、Aurora PostgreSQL DB インスタンスのリージョンを使用します。このリージョン値が `function_name` ARN で指定されたものと競合する場合、エラーが発生します。

invocation_type

Lambda 関数の呼び出しタイプ。 値は大文字と小文字が区別されます。以下に示しているのは、可能な値です。

- RequestResponse-デフォルト。Lambda 関数の呼び出しタイプは同期で、結果にレスポンスペイロードを返します。ワークフローが Lambda 関数の結果をすぐに受け取ることに依存しているときは、RequestResponse 呼び出しのタイプを使用します。
- Event- Lambda 関数の呼び出しタイプは非同期で、返されたペイロードなしにすぐに返されます。ワークフローを先に進める前に Lambda 関数の結果を知る必要がないときは、Event の呼び出しタイプを使用します。
- DryRun- この呼び出しタイプは、Lambda 関数を実行せずに、アクセスをテストします。

log_type

log_result出力パラメータで返される Lambda ログのタイプ。 値は大文字と小文字が区別されます。以下に示しているのは、可能な値です。

- Tail - 返された log_result 出力パラメータには、実行ログの最後の 4 KB が含まれます。
- None - Lambda のないログ情報は返されません。

context

JSON または JSONB形式のクライアントコンテキスト。使用されるフィールドには custom と env が含まれます 。

修飾子

呼び出される Lambda 関数のバージョンを識別する修飾子。この値が function_name ARN で指定されたものと競合する場合、エラーが発生します。

出力パラメータ

status_code

HTTP ステータスレスポンスコード。詳細については、AWS Lambda デベロッパーガイドの「Lambda 応答要素の呼び出し」を参照してください。

payload

実行された Lambda 関数から返された情報。形式は JSON または JSONB です。

executedversion

実行された Lambda 関数のバージョン。

result

Lambda 関数が呼び出されたとき log_type 値が Tail である場合に返される実行ログ情報。結果には、Base64 でエンコードされた実行ログの最後の 4 KB が含まれます。

aws_commons.create_lambda_function_arn

Lambda 関数名情報を保持するように、aws_commons._lambda_function_arn_1 構造を作成します。aws_commons.create_lambda_function_arn 関数の結果は、aws_lambda.invoke function_name 関数の aws_lambda.invoke パラメータで使用します。

Syntax

```
aws_commons.create_lambda_function_arn(
    function_name TEXT,
    region TEXT DEFAULT NULL
    )
    RETURNS aws_commons._lambda_function_arn_1
```

入力パラメータ

function_name

Lambda 関数名を含む必須のテキスト文字列。値には、関数名、部分的な ARN、または完全な ARN を指定します。

リージョン

Lambda 関数がある AWS リージョンを含む、オプションのテキスト文字列。 リージョン名と関連する値のリストについては、「」を参照してください。リージョンとアベイラビリティーゾーン

aws_lambda パラメータ

この表には、aws_lambda 関数に関連するパラメータが記載されています。

パラメータ	説明
aws_lambda.connect_timeout_ms	これは動的パラメータであり、AWS Lambda への接続中の最大待機時間を設定します。デフォルト値は 1000 です。このパラメータに指定できる値は、1 ~ 900000 です。
aws_lambda.request_timeout_ms	これは動的パラメータであり、AWS Lambda からのレスポンスの最大待機時間を設定します。デフォルト値は 3000 です。このパラメータに指定できる値は、1 ~ 900000 です。

パラメータ	説明
aws_lambda.endpoint_override	AWS Lambda への接続に使用できるエンドポイントを指定します。空の文字列は、リージョンのデフォルトの AWS Lambda エンドポイントを選択します。この静的パラメータの変更を有効にするには、データベースを再起動する必要があります。

Amazon CloudWatch Logs への Aurora PostgreSQL ログの発行

Aurora PostgreSQL DB クラスターでは、Amazon CloudWatch Logs にログデータを定期的にエクスポートするように設定できます。そうすると、Aurora PostgreSQL DB クラスターの PostgreSQL ログからのイベントが自動的に Amazon CloudWatch Logs として Amazon CloudWatch に発行されます。CloudWatch では、Aurora PostgreSQL DB クラスターのロググループでエクスポートされたログデータを検索できます。このロググループには、クラスター内の各インスタンスからの PostgreSQL ログのイベントを含む 1 つまたは複数のログストリームがあります。

CloudWatch Logs にログを発行することで、クラスターの PostgreSQL ログレコードを耐久性の高いストレージに保存できます。CloudWatch Logs で利用可能なログデータを使用して、クラスターの運用を評価し、改善できます。また、CloudWatch を使用すると、アラームの作成やメトリクスの表示が可能です。詳細については、Amazon CloudWatch でログイベントをモニタリングする を参照してください。

> **Note**
>
> PostgreSQL ログを CloudWatch Logs に発行するとストレージが消費され、その分の料金が発生します。不要になった CloudWatch Logs は削除してください。

既存の Aurora PostgreSQL DB クラスターのログエクスポートオプションをオフにしても、CloudWatch Logs で既に保存されているデータには影響しません。既存のログは、ログ保存設定に基づいて CloudWatch Logs で引き続き使用できます。CloudWatch Logs の詳細については、「Amazon CloudWatch Logs とは」を参照してください。

Aurora PostgreSQL では、次のバージョンで CloudWatch Logs へのログの発行をサポートしています。

- バージョン 14 の中の 14.3 以降

- 13.3 以上の 13 バージョン
- 12.8 以上の 12 バージョン
- 11.12 以上の 11 バージョン

Amazon CloudWatch にログを発行するオプションをオンにする

Aurora PostgreSQL DB クラスターの PostgreSQL ログを CloudWatch Logs に発行するには、クラスターの [Log export] (ログのエクスポート) オプションを選択します。Aurora PostgreSQL DB クラスターを作成する場合、ログのエクスポート設定を選択できます。または、後でクラスターを変更することもできます。既存のクラスターを変更すると、その時点で各インスタンスの PostgreSQL ログが CloudWatch クラスターに発行されます。Aurora PostgreSQL の場合、Amazon CloudWatch に公開されるログは、PostgreSQL ログ (`postgresql.log`) のみです。

AWS Management Console、AWS CLI、RDS API を使用して Aurora PostgreSQL DB クラスターのログエクスポート機能をオンにできます。

コンソール

[Log export] (ログのエクスポート) オプションを選択すると、Aurora PostgreSQL DB クラスターから CloudWatch Logs に PostgreSQL ログの発行が開始されます。

コンソールからログエクスポート機能をオンにするには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで [データベース] を選択します。
3. CloudWatch Logs にログデータを発行する Aurora PostgreSQL DB クラスターを選択します。
4. [Modify] を選択します。
5. [Log exports] (ログのエクスポート) セクションで、[PostgreSQL] を選択します。
6. [続行] を選択し、概要ページで [クラスターの変更] を選択します。

AWS CLI

ログのエクスポートオプションをオンにすると、AWS CLI で Amazon CloudWatch Logs に Aurora PostgreSQL ログの発行が開始されます。これを行うには、以下のオプションを指定して modify-db-cluster AWS CLI コマンドを実行します。

- `--db-cluster-identifier`— DB クラスター識別子。

- `--cloudwatch-logs-export-configuration`–DB クラスターで、CloudWatch Logs へエクスポートするログタイプを設定するための環境設定。

また、以下の AWS CLI コマンドのいずれかを実行して、Aurora PostgreSQL ログを発行することもできます。

- create-db-cluster
- restore-db-cluster-from-s3
- restore-db-cluster-from-snapshot
- restore-db-cluster-to-point-in-time

以下のオプションを使用して、この AWS CLI コマンドの 1 つを実行します。

- `--db-cluster-identifier`— DB クラスター識別子。
- `--engine` — データベースエンジン。
- `--enable-cloudwatch-logs-exports` — DB クラスターの CloudWatch Logs へのエクスポートに使用できるログタイプの構成設定。

実行する AWS CLI コマンドに応じて、他のオプションが必要となる場合があります。

Example

次のコマンドでは、ログファイルが CloudWatch Logs に発行されるよう Aurora PostgreSQL DB クラスターを作成します。

Linux、macOS、Unix の場合:

```
aws rds create-db-cluster \
    --db-cluster-identifier my-db-cluster \
    --engine aurora-postgresql \
    --enable-cloudwatch-logs-exports postgresql
```

Windows の場合:

```
aws rds create-db-cluster ^
    --db-cluster-identifier my-db-cluster ^
    --engine aurora-postgresql ^
```

```
    --enable-cloudwatch-logs-exports postgresql
```

Example

次のコマンドでは、ログファイルが CloudWatch Logs に発行されるよう既存の Aurora PostgreSQL DB クラスターを変更します。`--cloudwatch-logs-export-configuration` 値は JSON オブジェクトです。このオブジェクトのキーは `EnableLogTypes` で、その値は `postgresql` です。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster \
    --db-cluster-identifier my-db-cluster \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":["postgresql"]}'
```

Windows の場合:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier my-db-cluster ^
    --cloudwatch-logs-export-configuration '{\"EnableLogTypes\":[\"postgresql\"]}'
```

> **Note**
>
> Windows コマンドプロンプトを使用する場合、バックラッシュ (\)でプレフィックスして JSON コードの二重引用符 (")をエスケープする必要があります。

Example

次の例では、既存の Aurora PostgreSQL DB クラスターを変更して、CloudWatch Logs へのログファイルの発行を無効にします。`--cloudwatch-logs-export-configuration` 値は JSON オブジェクトです。このオブジェクトのキーは `DisableLogTypes` で、その値は `postgresql` です。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbinstance \
    --cloudwatch-logs-export-configuration '{"DisableLogTypes":["postgresql"]}'
```

Windows の場合:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbinstance ^
    --cloudwatch-logs-export-configuration "{\"DisableLogTypes\":[\"postgresql\"]}"
```

> **Note**
>
> Windows コマンドプロンプトを使用する場合、JSON コードでは、二重引用符 (") の前にバックスラッシュ (\) を付けてエスケープする必要があります。

RDS API

ログのエクスポートオプションをオンにすると、RDS API で Aurora PostgreSQL ログの発行が開始されます。これを行うには、以下のオプションを指定して [ModifyDBCluster] オペレーションを実行します。

- `DBClusterIdentifier` - DB クラスター識別子。
- `CloudwatchLogsExportConfiguration` - DB クラスターの CloudWatch Logs にエクスポートするログタイプを有効化するための環境設定。

以下の RDS API オペレーションのいずれかを実行することで、RDS API を使用して Aurora PostgreSQL ログを発行することもできます。

- CreateDBCluster
- RestoreDBClusterFromS3
- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime

次のパラメータを指定して RDS API アクションを実行します。

- `DBClusterIdentifier`— DB クラスター識別子。
- `Engine` — データベースエンジン。
- `EnableCloudwatchLogsExports` — DB クラスターの CloudWatch Logs へのエクスポートに使用できるログタイプの構成設定。

実行する AWS CLI コマンドに応じて、他のパラメータが必要となる場合があります。

Amazon CloudWatch でログイベントをモニタリングする

Aurora PostgreSQL ログイベントを Amazon CloudWatch Logs として発行して利用可能にすることで、Amazon CloudWatch を使用してイベントを表示およびモニタリングできます。モニタリングの詳細については、「CloudWatch Logs に送信されたログデータを表示する」を参照してください。

[Log exports] (ログのエクスポート) をオンにすると、プレフィックス `/aws/rds/cluster/` を使用して、次のパターンのように、Aurora PostgreSQL の名前とログタイプで新規のロググループが自動的に作成されます。

```
/aws/rds/cluster/your-cluster-name/postgresql
```

例として、`docs-lab-apg-small` という名前の Aurora PostgreSQL DB クラスターが Amazon CloudWatch Logs にログをエクスポートするとします。Amazon CloudWatch のロググループ名を以下に示します。

```
/aws/rds/cluster/docs-lab-apg-small/postgresql
```

指定する名前のロググループがすでに存在する場合、Aurora はこのロググループを使用して Aurora DB クラスターにログデータをエクスポートします。Aurora PostgreSQL DB クラスターの各 DB インスタンスは、その PostgreSQL ログを個別のログストリームとしてロググループにアップロードします。Amazon CloudWatch で利用できるさまざまなグラフィカルツールや分析ツールを使用して、ロググループとそのログストリームを調べることができます。

例えば、Aurora PostgreSQL DB クラスターのログイベント内の情報を検索することや、CloudWatch Logs コンソール、AWS CLI、CloudWatch Logs API を使用してイベントをフィルタリングすることができます。詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「ログデータの検索およびフィルタリング」を参照してください。

デフォルトでは、新しいロググループは保持時間として [Never expire] (期限なし) を使用して作成します。ログの保持期間を変更するには、CloudWatch Logs コンソール、AWS CLI、または CloudWatch Logs API を使用します。詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「CloudWatch ログでのログデータ保持期間の変更」を参照してください。

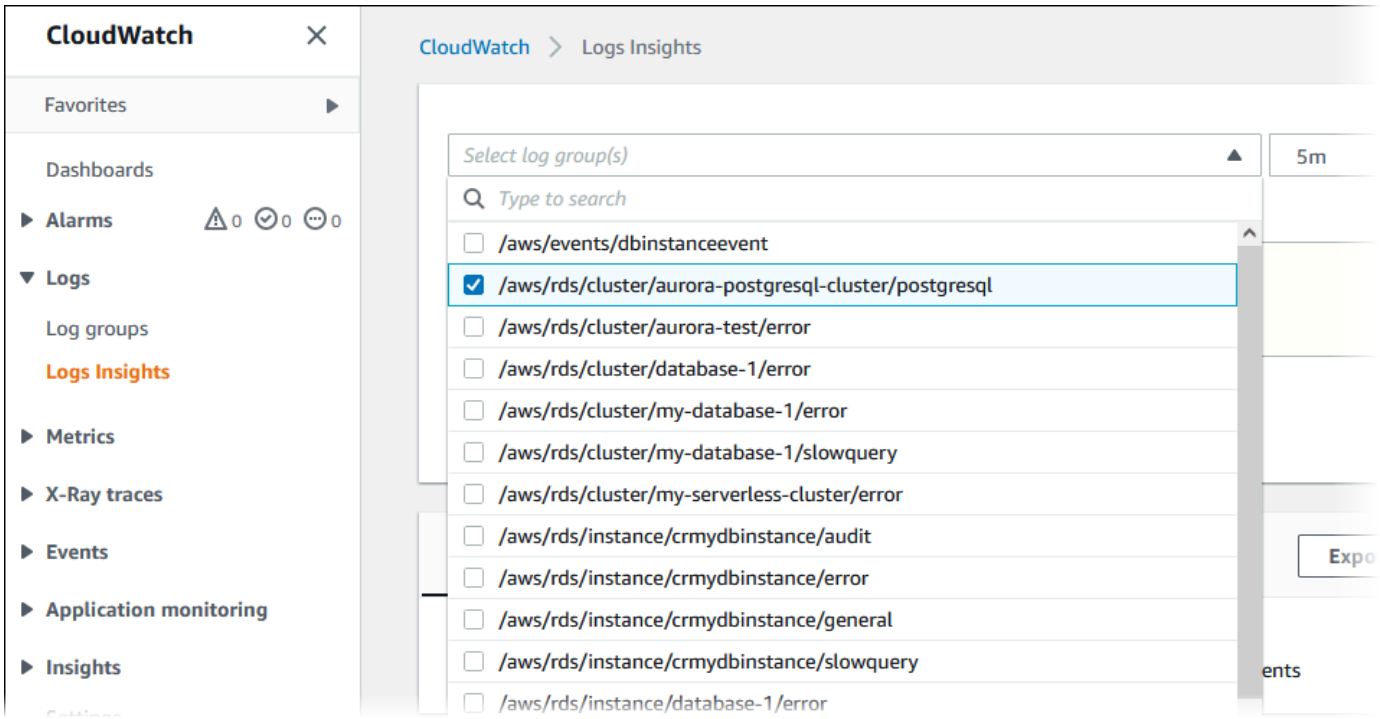
> **Tip**
>
> AWS CloudFormation などの自動設定を使用して、ログの保持期間、メトリックフィルター、アクセス権などを事前定義したロググループを作成できます。

CloudWatch Logs Insights を使用した PostgreSQL のログ分析

Aurora PostgreSQL DB クラスターの PostgreSQL ログが CloudWatch Logs として発行されると、CloudWatch Logs Insights を使用して Amazon CloudWatch Logs のログデータをインタラクティブに検索および分析できます。CloudWatch Logs Insights には、ログデータを分析するためのクエリ言語、サンプルクエリなどのツールが含まれており、潜在的な問題を特定して修正を確認できます。詳細については、「Amazon CloudWatch Logs ユーザーガイド」の「CloudWatch Logs Insights を使用したログデータの分析」を参照してください。Amazon CloudWatch Logs

CloudWatch Logs Insights を使用して PostgreSQL ログを分析する方法

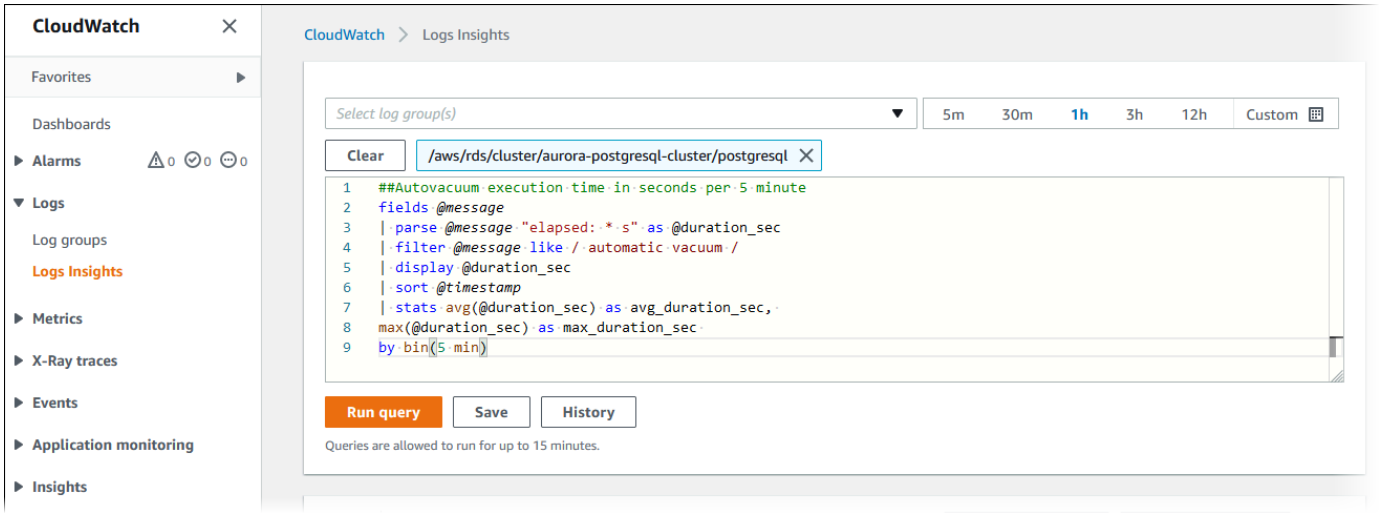
1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. ナビゲーションペインでログを開いて ログインサイト を選択します。
3. [Select log group(s)] (ロググループの選択) で、Aurora PostgreSQL DB クラスターのロググループを選択します。



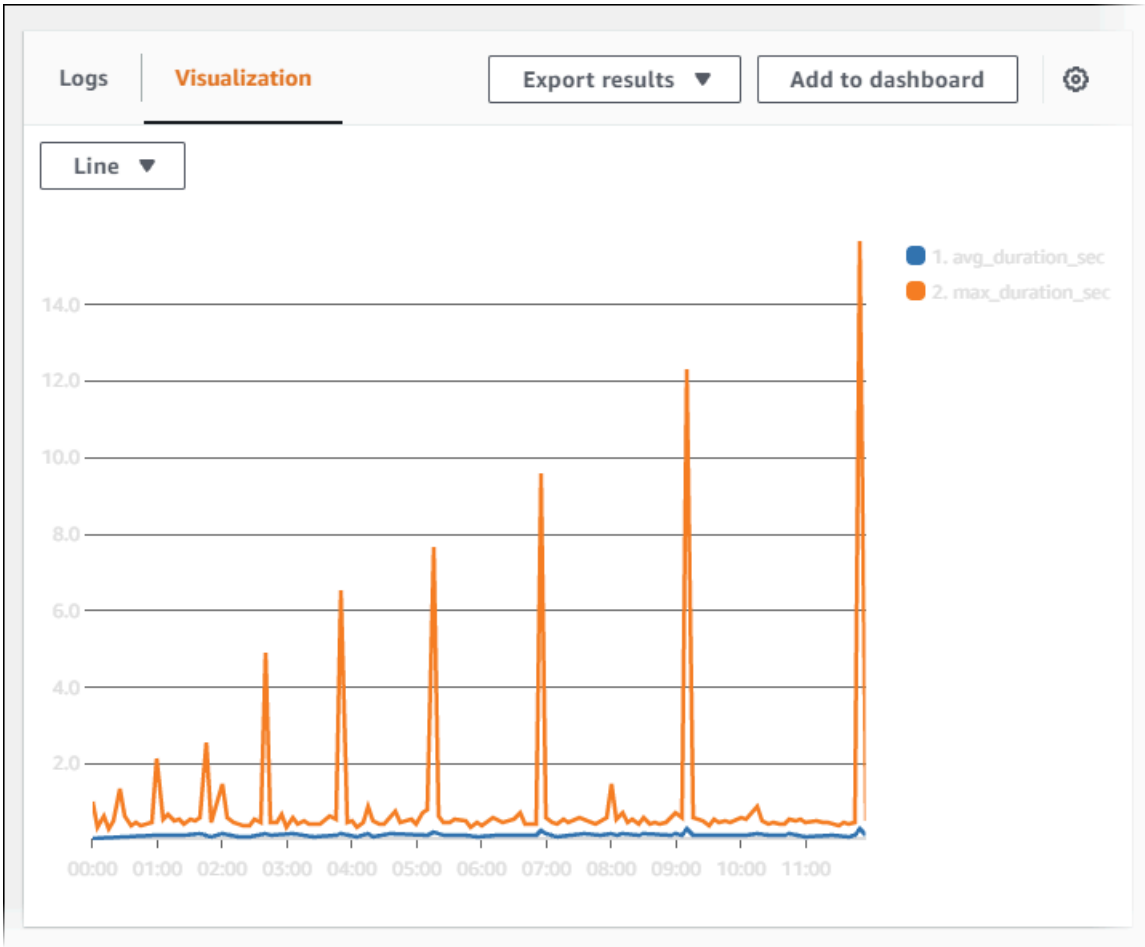
4. クエリエディタで、現在表示されているクエリを削除し、以下を入力してクエリの実行を選択します。

```
##Autovacuum execution time in seconds per 5 minute
fields @message
| parse @message "elapsed: * s" as @duration_sec
| filter @message like / automatic vacuum /
```

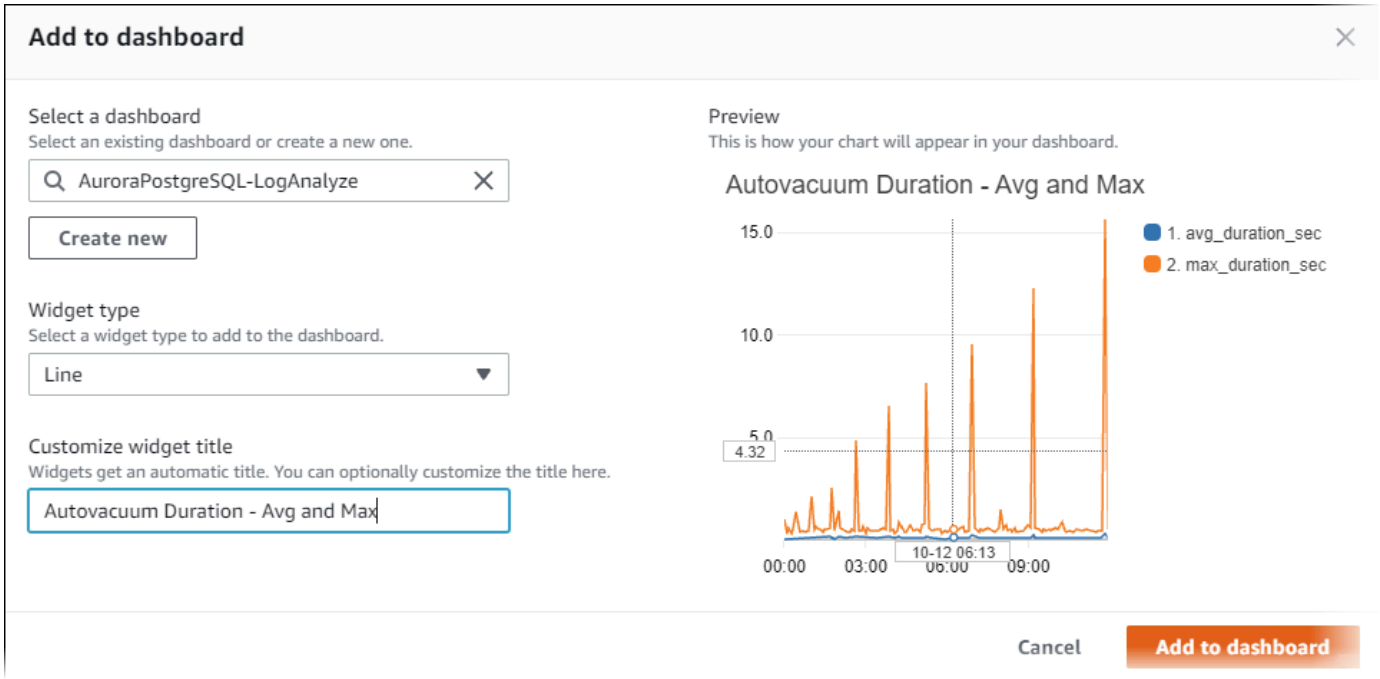
```
| display @duration_sec
| sort @timestamp
| stats avg(@duration_sec) as avg_duration_sec,
max(@duration_sec) as max_duration_sec
by bin(5 min)
```



5. [Visualization (視覚化)] タブを選択します。



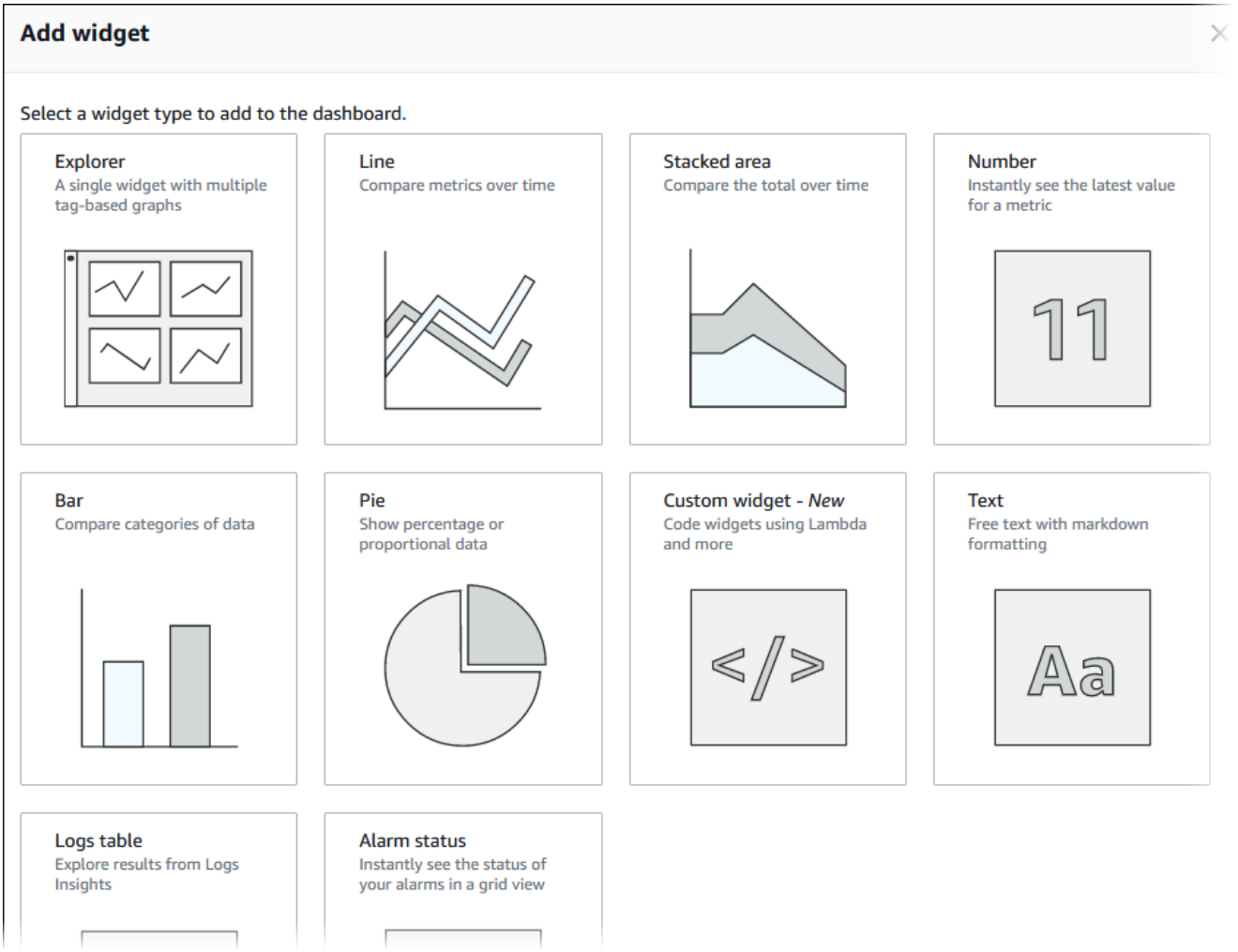
6. [ダッシュボードに追加] を選択します。
7. ダッシュボードを選択で、ダッシュボードを選択するか、名前を入力して新しいダッシュボードを作成します。
8. ウィジェットのタイプで、視覚化のウィジェットタイプを選択します。



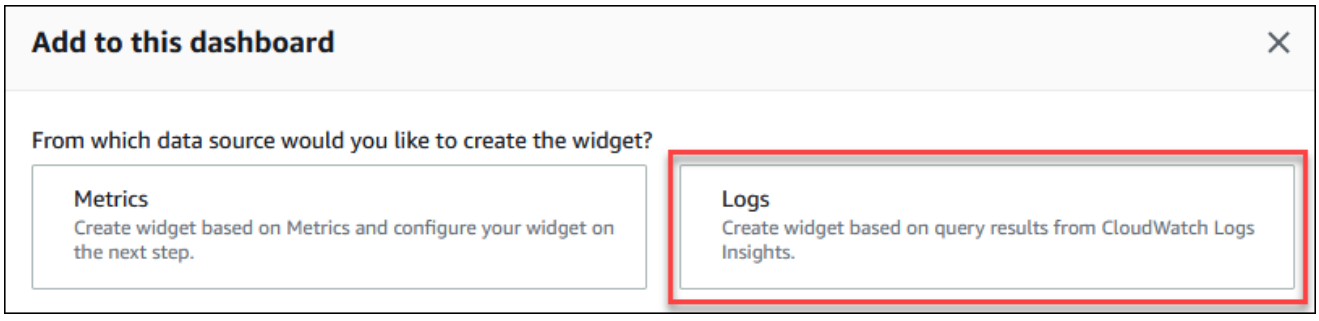
9. (オプション) ログクエリの結果に基づいてウィジェットを追加します。

   a. [ウィジェットの追加] を選択します。

   b. Line などの、ウィジェットタイプを選択します。



c. このダッシュボードに追加するウィンドウで、ログを選択します。

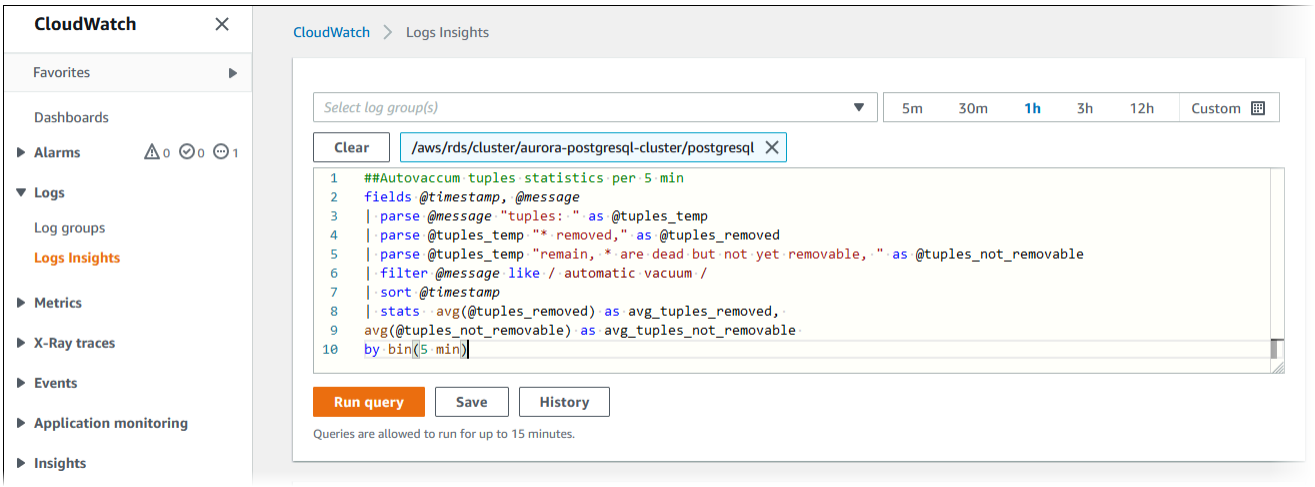


d. ロググループの選択で、DB クラスターのロググループを選択します。

e. クエリエディタで、現在表示されているクエリを削除し、以下を入力してクエリの実行を選択します。

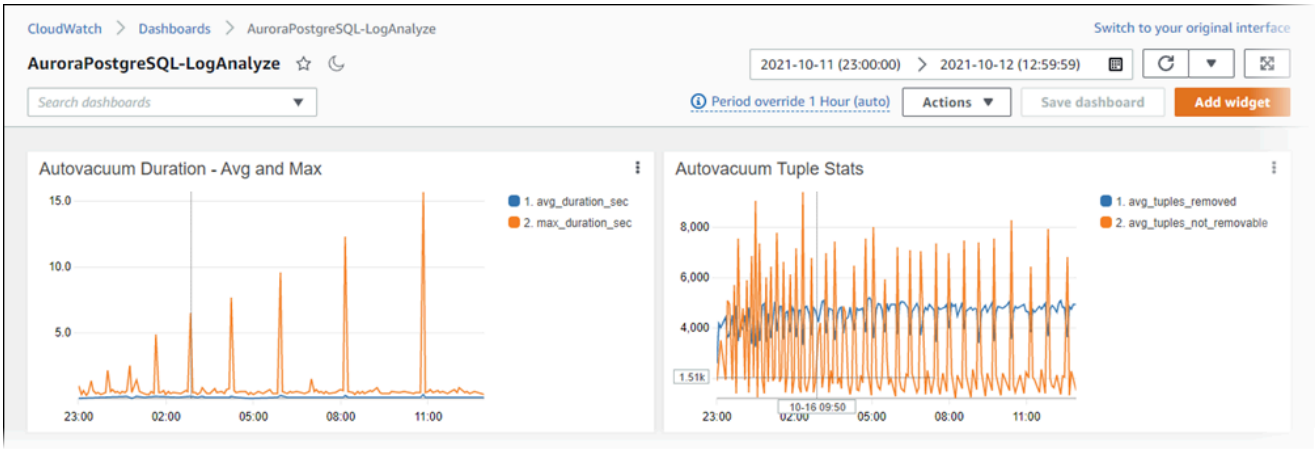
```
##Autovacuum tuples statistics per 5 min
fields @timestamp, @message
| parse @message "tuples: " as @tuples_temp
```

```
| parse @tuples_temp "* removed," as @tuples_removed
| parse @tuples_temp "remain, * are dead but not yet removable, " as
 @tuples_not_removable
| filter @message like / automatic vacuum /
| sort @timestamp
| stats  avg(@tuples_removed) as avg_tuples_removed,
avg(@tuples_not_removable) as avg_tuples_not_removable
by bin(5 min)
```



f. [ウィジェットの作成] を選択します。

ダッシュボードは次のイメージのような見た目になっている必要があります。



Aurora PostgreSQL のクエリ実行計画のモニタリング

Aurora PostgreSQL DB インスタンスのクエリ実行計画をモニタリングして、現在のデータベース負荷の原因となる実行計画を検出し、aurora_compute_plan_id パラメータを使用して実行計画のパフォーマンス統計を経時的に追跡できます。クエリを実行するたびに、クエリで使用される実行計画に識別子が割り当てられ、同じ計画の後続の実行でも同じ識別子が使用されます。

Aurora PostgreSQL バージョン 14.10、15.5 以降のバージョンの DB パラメータグループでは、aurora_compute_plan_id がデフォルトでオンになっています。計画識別子の割り当てはデフォルトの動作であり、パラメータグループで aurora_compute_plan_id を OFF に設定することでオフにできます。

このプラン識別子は、異なる用途の複数のユーティリティで使用されます。

トピック

- Aurora 関数を使用したクエリ実行計画へのアクセス
- Aurora PostgreSQL クエリ実行計画のパラメータリファレンス

Aurora 関数を使用したクエリ実行計画へのアクセス

aurora_compute_plan_id では、次の関数を使用して実行計画にアクセスできます。

- aurora_stat_activity
- aurora_stat_plans

これらの関数の詳細については、「Aurora PostgreSQL 関数のリファレンス」を参照してください。

Aurora PostgreSQL クエリ実行計画のパラメータリファレンス

DB パラメータグループの以下のパラメータを使用して、クエリ実行計画をモニタリングできます。

パラメータ

- aurora_compute_plan_id
- aurora_stat_plans.minutes_until_recapture
- aurora_stat_plans.calls_until_recapture
- aurora_stat_plans.with_costs

- aurora_stat_plans.with_analyze
- aurora_stat_plans.with_timing
- aurora_stat_plans.with_buffers
- aurora_stat_plans.with_wal
- aurora_stat_plans.with_triggers

> **Note**
>
> `aurora_stat_plans.with_*` パラメータの設定は、新しくキャプチャされた計画に対してのみ有効になります。

aurora_compute_plan_id

`off` に設定すると、計画識別子が割り当てられなくなります。

デフォルト値	許可される値	説明
オン	0 (オフ)	`off` に設定すると、計画識別子が割り当てられなくなります。
	1 (オン)	プラン識別子を割り当てるには、on に設定します。

aurora_stat_plans.minutes_until_recapture

計画が再キャプチャされるまでに経過する分数。デフォルトは 0 で、計画の再キャプチャが無効になります。`aurora_stat_plans.calls_until_recapture` しきい値を超えた場合、計画は再キャプチャされます。

デフォルト値	許可される値	説明
0	0-1073741823	計画が再キャプチャされるまでに経過する分数を設定します。

aurora_stat_plans.calls_until_recapture

計画が再キャプチャされるまでの計画の呼び出し回数。デフォルトは 0 で、呼び出し回数後に計画の再キャプチャが無効になります。aurora_stat_plans.minutes_until_recapture しきい値を超えた場合、計画は再キャプチャされます。

デフォルト値	許可される値	説明
0	0-1073741823	計画が再キャプチャされるまでの計画の呼び出し回数を設定します。

aurora_stat_plans.with_costs

推定コストを含む EXPLAIN 計画を取得します。指定できる値は、on および off です。デフォルト: on。

デフォルト値	許可される値	説明
on	0 (オフ)	各計画ノードの推定コストと行は表示されません。
	1 (オン)	各計画ノードの推定コストと行が表示されます。

aurora_stat_plans.with_analyze

ANALYZE で EXPLAIN 計画を制御します。このモードは、計画が初めてキャプチャされたときにのみ使用されます。指定できる値は、on および off です。デフォルト: off。

デフォルト値	許可される値	説明
オフ	0 (オフ)	計画の実際のランタイム統計は含まれません。
	1 (オン)	計画の実際のランタイム統計が含まれます。

aurora_stat_plans.with_timing

ANALYZE を使用すると、計画のタイミングが説明にキャプチャされます。デフォルト: on。

デフォルト値	許可される値	説明
オン	0 (オフ)	実際の起動時間と各計画ノードで費やされた時間は含まれません。
	1 (オン)	実際の起動時間と各計画ノードで費やされた時間が含まれます。

aurora_stat_plans.with_buffers

ANALYZE を使用すると、計画のバッファの使用統計が説明にキャプチャされます。デフォルト: `off`。

デフォルト値	許可される値	説明
off	0 (オフ)	バッファの使用に関する情報は含まれません。
	1 (オン)	バッファの使用に関する情報が含まれます。

aurora_stat_plans.with_wal

ANALYZE を使用すると、計画の WAL 使用統計が説明にキャプチャされます。デフォルト: `off`。

デフォルト値	許可される値	説明
off	0 (オフ)	WAL レコード生成に関する情報は含まれません。
	1 (オン)	WAL レコード生成に関する情報が含まれます。

aurora_stat_plans.with_triggers

計画トリガー実行統計は、ANALYZE の使用時に説明にキャプチャされます。デフォルト: off。

デフォルト値	許可される値	説明
off	0 (オフ)	トリガー実行統計は含まれません。
	1 (オン)	トリガー実行統計が含まれます。

Aurora PostgreSQL のクエリ実行計画の管理

Aurora PostgreSQL クエリプラン管理は、Amazon Aurora の PostgreSQL 互換エディション DB クラスターで使用できるオプション機能です。この機能は、Aurora PostgreSQL DB クラスターにインストールできる apg_plan_mgmt エクステンションとしてパッケージ化されています。クエリプラン管理により、SQL アプリケーションについてオプティマイザによって生成されたクエリ実行プランを管理できます。apg_plan_mgmt AWS エクステンションは、PostgreSQL データベースエンジンのネイティブクエリ処理機能をベースにしています。

Aurora PostgreSQL クエリプラン管理機能、設定方法、Aurora PostgreSQL DB クラスターでの使用方法について、以下で説明します。始める前に、お使いの Aurora PostgreSQL バージョンで使用可能な特定のバージョンの apg_plan_mgmt エクステンションのリリースノートを確認することをお勧めします。詳細については、『Aurora PostgreSQL リリースノート』の「Aurora PostgreSQL apg_plan_mgmt エクステンションのバージョン」を参照してください。

トピック

Aurora PostgreSQL のクエリプラン管理の概要

Aurora PostgreSQL クエリプラン管理は、クエリプランのリグレッションを引き起こす可能性のあるデータベースの変更に関係なく、計画の安定性を確保するように設計されています。クエリプランのリグレッションは、システムまたはデータベースが変更された後に、オプティマイザが特定の SQL ステートメントに対して最適ではない計画を選択したときに発生します。統計情報、制限事項、環境設定、クエリパラメータのバインディングの変更、PostgreSQL データベースエンジンのアップグレードは、すべて計画のリグレッションの原因になる可能性があります。

Aurora PostgreSQL のクエリプラン管理を使用すると、クエリ実行計画の変更方法と変更時期を制御できます。Aurora PostgreSQL クエリプラン管理には次のような利点があります。

- オプティマイザに、問題のないことがわかっている少数の計画から強制的に選択させることで、計画の安定性を改善する。
- 計画を一元的に最適化してから、最善の計画を全体に配布する。
- 使用されていないインデックスを特定し、インデックスの作成または削除の影響を評価する。
- オプティマイザが発見した、新しい最小コスト計画を自動的に検出する。
- パフォーマンスを改善する計画変更のみが承認されるように選択し、新しいオプティマイザの機能を少ないリスクで試す。

クエリプラン管理が提供するツールを積極的に使用して、特定のクエリに最適な計画を指定できます。または、クエリプラン管理を使用して、変化する状況に対応し、計画のリグレッションを回避することもできます。詳細については、「Aurora PostgreSQL クエリ計画管理のベストプラクティス」を参照してください。

トピック

- サポートされている SQL ステートメント
- クエリプラン管理の制限事項
- クエリプラン管理用語
- Aurora PostgreSQL クエリプラン管理のバージョン
- Aurora PostgreSQL のクエリプラン管理をオンにする
- Aurora PostgreSQL のクエリ計画管理アップグレードする
- Aurora PostgreSQL のクエリプラン管理をオフにする

サポートされている SQL ステートメント

クエリプラン管理は、次のタイプの SQL ステートメントをサポートします。

- 複雑性に関係なく、任意の SELECT、INSERT、UPDATE、DELETE ステートメント。
- プリペアドステートメント。詳細については、PostgreSQL のドキュメントの「PREPARE」を参照してください。
- 動的ステートメント (即時モードで実行されるものを含む)。詳細については、PostgreSQL ドキュメントの「動的 SQL」と「EXECUTE IMMEDIATE」を参照してください。

- 埋め込み SQL コマンドとステートメント。詳細については、PostgreSQL ドキュメントの「埋め込み SQL コマンド」を参照してください。
- 名前付き関数内のステートメント。詳細については、PostgreSQL のドキュメントの「CREATE FUNCTION」を参照してください。
- 一時テーブルを含むステートメント。
- プロシージャと DO ブロック内のステートメント。

クエリプラン管理を EXPLAIN と手動モードで使用すると、実際に実行しなくてもプランをキャプチャできます。詳細については、「オプティマイザが選択した計画の分析」を参照してください。クエリプラン管理のモード (手動、自動) の詳細については、「Aurora PostgreSQL 実行計画のキャプチャ」を参照してください。

Aurora PostgreSQL クエリプラン管理は、パーティショニングされたテーブル、継承、行レベルセキュリティ、再帰的なテーブル共通表現 (CTE) など、PostgreSQL のすべての言語機能をサポートします。これらの PostgreSQL 言語機能の詳細については、PostgreSQL ドキュメントの「テーブルパーティショニング」、「行セキュリティポリシー」、「WITH クエリ (共通テーブル式) 」およびその他のトピックを参照してください。

Aurora PostgreSQL のクエリプラン管理機能のさまざまなバージョンの詳細については、『Aurora PostgreSQL のリリースノート』の「Aurora PostgreSQL apg_plan_mgmt 拡張機能バージョン」を参照してください。

クエリプラン管理の制限事項

Aurora PostgreSQL のクエリプラン管理の現在のリリースには、次のような制限があります。

- システムリレーションを参照するステートメントのプランはキャプチャされません。pg_class など、システムリレーションを参照するステートメントはキャプチャされません。これは設計によるものであり、内部で使用される多数のシステム生成計画がキャプチャされないようにするためです。これはシステムテーブル内部ビューにも当てはまります。
- Aurora PostgreSQL DB クラスターには、より大きな DB インスタンスクラスが必要になる場合があります。ワークロードによっては、クエリプラン管理に 3 つ以上の vCPU を持つ DB インスタンスクラスが必要になる場合があります。max_worker_processes の数は、DB インスタンスクラスのサイズによって制限されます。2 vCPU の DB インスタンスクラス (db.t3.medium など) によって提供される max_worker_processes の数は、特定のワークロードに対して十分ではない場合があります。クエリプラン管理を使用する場合は、Aurora PostgreSQL DB クラスター用に 3 つ以上の vCPU を備えた DB インスタンスクラスを選択することをお勧めします。

DB インスタンスクラスがワークロードをサポートできないと、クエリプラン管理は次のようなエラーメッセージをレイズします。

```
WARNING: could not register plan insert background process
HINT: You may need to increase max_worker_processes.
```

この場合、Aurora PostgreSQL DB クラスターを、より多くのメモリを搭載した DB インスタンスクラスのサイズにスケールアップする必要があります。詳細については、「DB インスタンスクラスでサポートされている DB エンジン」を参照してください。

- セッションに既に保存されているプランは影響を受けません。クエリプラン管理を使用すると、アプリケーションコードを変更せずにクエリプランに影響を与えることができます。ただし、ジェネリックプランが既存のセッションに既に保存されていて、そのクエリプランを変更する場合は、まず、DB クラスターパラメータグループで plan_cache_mode をforce_custom_plan に設定する必要があります。
- 次の場合、apg_plan_mgmt.dba_plans と pg_stat_statements の queryid は異なる可能性があります。
  - オブジェクトが apg_plan_mgmt.dba_plan に保存された後、削除され、再作成されたとき。
  - apg_plan_mgmt.plans テーブルが別のクラスターからインポートされたとき。

Aurora PostgreSQL のクエリプラン管理機能のさまざまなバージョンの詳細については、『Aurora PostgreSQL のリリースノート』の「Aurora PostgreSQL apg_plan_mgmt 拡張機能バージョン」を参照してください。

クエリプラン管理用語

このトピックでは、次の用語を使用します。

マネージドステートメント

クエリプラン管理でオプティマイザにより取得された SQL ステートメントです。マネージドステートメントには、apg_plan_mgmt.dba_plans ビューに 1 つ以上のクエリ実行計画が格納されています。

計画ベースライン

特定のマネージドステートメントで承認された計画のセット。つまり、dba_plan ビュー内の status 列が「承認済み」になっているマネージドステートメントのすべての計画です。

計画履歴

特定のマネージドステートメント用に取得された一連のすべての計画。計画履歴には、ステータスに関係なく、ステートメントにキャプチャされたすべての計画が含まれます。

クエリプランのリグレッション

オプティマイザが、新しい PostgreSQL バージョンや統計の変更など、データベース環境への特定の変更前よりも最適でない計画を選択した場合。

Aurora PostgreSQL クエリプラン管理のバージョン

クエリプラン管理は、現在の Aurora PostgreSQL リリースで利用可能なすべての Aurora PostgreSQL リリースでサポートされています。詳細な情報については、Aurora PostgreSQL リリースノート」の「Amazon Aurora PostgreSQL の更新」のリストを参照してください。

apg_plan_mgmt 拡張機能をインストールすると、クエリプラン管理機能が Aurora PostgreSQL DB クラスターに追加されます。Aurora PostgreSQL の異なるバージョンでは、異なるバージョンの apg_plan_mgmt 拡張機能がサポートされます。クエリプラン管理拡張機能を、ご使用の Aurora PostgreSQL バージョンの最新リリースにアップグレードすることをお勧めします。

> **Note**
>
> 各 apg_plan_mgmt の拡張機能バージョンのリリースノートについては、Aurora PostgreSQL のリリースノートの Aurora PostgreSQL apg_plan_mgmt 拡張機能バージョンを参照してください。

クラスターで実行されているバージョンを特定するには、psql を使用してインスタンスに接続し、メタコマンド \dx を使用して次に示すように拡張機能を一覧表示します。

```
labdb=> \dx
                      List of installed extensions
     Name      | Version |    Schema     |                              Description
---------------+---------+---------------
+-------------------------------------------------------------------
 apg_plan_mgmt | 1.0     | apg_plan_mgmt | Amazon Aurora with PostgreSQL compatibility
 Query Plan Management
 plpgsql       | 1.0     | pg_catalog    | PL/pgSQL procedural language
(2 rows)
```

出力は、このクラスターがエクステンションの 1.0 バージョンを使用していることを示しています。特定の Aurora PostgreSQL バージョンで使用できるのは特定の apg_plan_mgmt バージョンのみです。場合によっては、Aurora PostgreSQL DB クラスターを新しいマイナーリリースにアップグレードするか、最新バージョンのクエリプラン管理にアップグレードできるようにパッチを適用する必要があります。出力に表示される apg_plan_mgmt バージョン 1.0 は Aurora PostgreSQL バージョン 10.17 DB クラスターのもので、新しいバージョンの apg_plan_mgmt は利用できません。この場合、Aurora PostgreSQL DB クラスターをより新しいバージョンの PostgreSQL にアップグレードする必要があります。

Aurora PostgreSQL DB クラスターの新しいバージョンの PostgreSQL へのアップグレードの詳細については、「Amazon Aurora PostgreSQL の更新」を参照してください。

apg_plan_mgmt 拡張機能のアップグレード方法については、Aurora PostgreSQL のクエリ計画管理アップグレードする を参照してください。

Aurora PostgreSQL のクエリプラン管理をオンにする

Aurora PostgreSQL DB クラスターのクエリプラン管理を設定するには、拡張機能をインストールし、いくつかの DB クラスターパラメータ設定を変更する必要があります。apg_plan_mgmt 拡張機能をインストールし、Aurora PostgreSQL DB クラスターの機能を有効にする rds_superuser 権限が必要です。

拡張機能をインストールすると、新しいロール apg_plan_mgmt が作成されます。このロールにより、データベースユーザーはクエリプランを表示、管理、および管理できます。rds_superuser 権限を持つ管理者として、必要に応じてデータベースユーザーに apg_plan_mgmt ロールを付与してください。

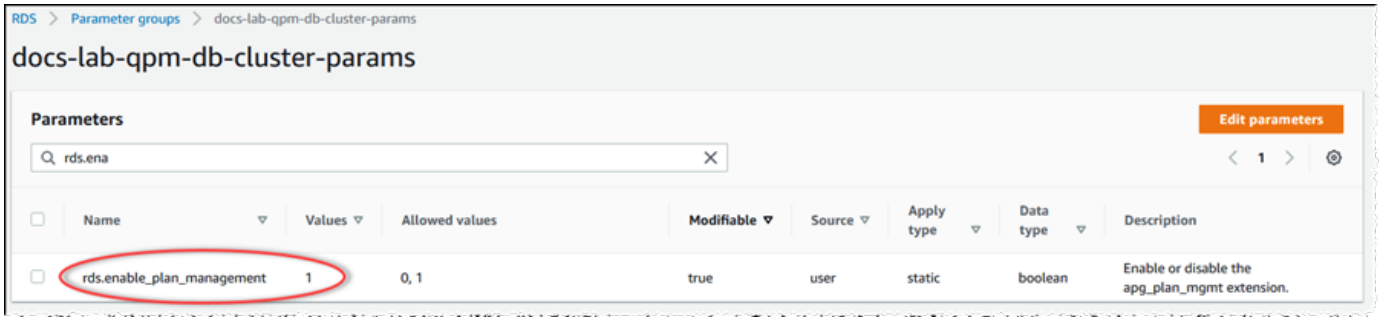
次の手順を完了できるのは、rds_superuser ロールを持つユーザーのみです。rds_superuser は、apg_plan_mgmt エクステンションとその apg_plan_mgmt ロールの作成に必要です。apg_plan_mgmt エクステンションを管理するには、ユーザーに apg_plan_mgmt ロールを付与する必要があります。

Aurora PostgreSQL DB クラスターのクエリプラン管理をオンにするには

以下のステップでは、Aurora PostgreSQL DB クラスターに送信されるすべての SQL ステートメントのクエリプラン管理を有効にします。これは自動モードと呼ばれます。モードの違いについての詳細は、「Aurora PostgreSQL 実行計画のキャプチャ」を参照してください。

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。

2. Aurora PostgreSQL DB クラスターに使用する DB クラスターパラメータグループを作成します。クエリプラン管理を有効にし、その動作を設定するには、特定のパラメータを変更する必要があります。詳細については、「DB パラメータグループを作成する」を参照してください。
3. 次の画像に示すように、カスタム DB クラスターパラメータグループを開き、`rds.enable_plan_management` パラメータを `1` に設定します。



詳細については、「DB クラスターパラメータグループのパラメータの変更」を参照してください。

4. インスタンスレベルでクエリプランパラメータを設定するために使用できるカスタム DB パラメータグループを作成します。詳細については、「DB クラスターのパラメータグループの作成」を参照してください。
5. Aurora PostgreSQL DB クラスターのライターインスタンスを変更して、カスタム DB パラメータグループを使用します。詳細については、「DB クラスター内の DB インスタンスの変更」を参照してください。
6. カスタム DB パラメータグループを使用するには、Aurora PostgreSQL DB クラスターを変更します。詳細については、「コンソール、CLI、API を使用した DB クラスターの変更」を参照してください。
7. DB インスタンスを再起動してカスタムパラメータグループ設定を有効にします。
8. `psql` または pgAdmin を使用して Aurora PostgreSQL DB クラスターの DB インスタンスエンドポイントに接続します。次の例では、`postgres` ロールにデフォルトの `rds_superuser` アカウントを使用しています。

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=my-db
```

9. 次に示すように、DB インスタンス用の `apg_plan_mgmt` エクステンションを作成します。

```
labdb=> CREATE EXTENSION apg_plan_mgmt;
CREATE EXTENSION
```

> **Tip**
>
> アプリケーションのテンプレートデータベースに apg_plan_mgmt エクステンションをインストールします。デフォルトのテンプレートデータベースは、template1 という名前です。詳細については、PostgreSQL ドキュメントの「テンプレートデータベース」を参照してください。

10. apg_plan_mgmt.capture_plan_baselines パラメータを automatic に変更します。この設定では、オプティマイザは、2 回以上計画または実行されるすべての SQL ステートメントのプランを生成します。

> **Note**
>
> クエリプラン管理には、特定の SQL ステートメントに使用できる手動モードもあります。詳細については、「Aurora PostgreSQL 実行計画のキャプチャ」を参照してください。

11. apg_plan_mgmt.use_plan_baselines パラメータの値を「オン」に変更します。このパラメータを指定すると、オプティマイザは計画ベースラインからステートメントの計画を選択します。詳細については、「Aurora PostgreSQL 管理計画を使用する」を参照してください。

> **Note**
>
> インスタンスを再起動しなくても、セッションのこれらの動的パラメータのいずれかの値を変更できます。

クエリプラン管理の設定が完了したら、クエリプランを表示、管理、または維持する必要があるすべてのデータベースユーザーに apg_plan_mgmt ロールを必ず付与してください。

Aurora PostgreSQL のクエリ計画管理アップグレードする

クエリプラン管理拡張機能を、ご使用の Aurora PostgreSQL バージョンの最新リリースにアップグレードすることをお勧めします。

1. rds_superuser 権限があるユーザーとして、Aurora PostgreSQL DB クラスターのライターインスタンスに接続します。インスタンスの設定時にデフォルトの名前を保持している場合

は、postgres として接続します。次の例では、psql の使用方法を示していますが、希望に応じて pgAdmin を使用することもできます。

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password
```

2. 拡張機能をアップグレードするには、次のクエリを実行します。

```
ALTER EXTENSION apg_plan_mgmt UPDATE TO '2.1';
```

3. apg_plan_mgmt.validate_plans 関数を使用して、すべての計画のハッシュを更新します。オプティマイザは、承認済み、未承認、拒否済みのすべての計画を検証して、新しいバージョンの拡張機能でも引き続き実行可能な計画であることを確認します。

```
SELECT apg_plan_mgmt.validate_plans('update_plan_hash');
```

この関数の使用の詳細については、「計画の検証」を参照してください。

4. apg_plan_mgmt.reload 関数を使用して、共有メモリ内のすべての計画を dba_plans ビューからの検証済み計画で更新します。

```
SELECT apg_plan_mgmt.reload();
```

クエリプラン管理に使用できるすべての機能の詳細については、「Aurora PostgreSQL クエリ計画管理の関数リファレンス 」を参照してください。

Aurora PostgreSQL のクエリプラン管理をオフにする

クエリプラン管理は、apg_plan_mgmt.use_plan_baselines および apg_plan_mgmt.capture_plan_baselines をオフにすることで、いつでも無効にできます。

```
labdb=> SET apg_plan_mgmt.use_plan_baselines = off;

labdb=> SET apg_plan_mgmt.capture_plan_baselines = off;
```

Aurora PostgreSQL クエリ計画管理のベストプラクティス

クエリ計画管理を使用すると、クエリ実行計画の変更方法と変更時期を制御できます。DBA として、QPM を使用する際の主な目標は、データベースに変更があったときのリグレッションの防止

と、オプティマイザが新しい計画を使用できるかどうかの制御です。クエリ計画管理の使用に関する推奨されるベストプラクティスについて、以下に示します。事前予防型および事後対応型の計画管理アプローチでは、新しい計画の使用を承認する方法とタイミングが異なります。

目次

パフォーマンス低下を防止する事前予防型の計画管理

計画のパフォーマンス低下を防止するには、新しく検出された計画のパフォーマンスを、承認済み計画の既存のベースラインのパフォーマンスと比較し、最速の計画セットを新しいベースラインとして自動的に承認する手順を実行して、計画ベースラインを進化させます。このように、より速い計画が検出されるにつれて、時間とともに計画のベースラインが改善されます。

1. 開発環境で、パフォーマンスまたはシステムのスループットに最も大きな影響を与える SQL ステートメントを特定します。その後、特定の SQL ステートメントの計画の手動取り込み と 自動的な計画の取得 の説明に従ってこれらのステートメントの計画を取得します。
2. 取得した計画を開発環境からエクスポートし、それらを実稼働環境にインポートします。詳細については、「計画のエクスポートおよびインポート」を参照してください。
3. 実稼働環境で、アプリケーションを実行し、承認された管理計画の使用を強制します。詳細については、「Aurora PostgreSQL 管理計画を使用する」を参照してください。アプリケーションの実行中にオプティマイザが新しい計画を発見したときは、それらも追加します。詳細については、「自動的な計画の取得」を参照してください。
4. 未承認の計画を分析し、適切に機能する計画を承認します。詳細については、「計画パフォーマンスの評価」を参照してください。
5. アプリケーションが継続して実行されている間、オプティマイザは必要に応じて新しい計画を使用し始めます。

メジャーバージョンのアップグレード後の計画の安定性の確保

PostgreSQL の各メジャーバージョンには、パフォーマンスを向上させるために設計されたクエリオプティマイザの機能強化と変更が含まれています。ただし、以前のバージョンでオプティマイザーによって生成されたクエリ実行プランは、アップグレードされた新しいバージョンではパフォーマン

スが低下する可能性があります。クエリプラン管理を使用して、これらのパフォーマンス問題を解決し、メジャーバージョンアップグレード後のプランの安定性を確保できます。

オプティマイザは、同じステートメントに承認されたプランが複数存在していても、常に最小コストのプランを使用します。アップグレード後、オプティマイザは新しいプランを検出する場合がありますが、それらは未承認プランとして保存されます。これらのプランは、unapproved_plan_execution_threshold パラメータによるリアクティブスタイルのプラン管理を使用して承認された場合にのみ実行されます。evolve_plan_baselines パラメータを使ったプロアクティブスタイルのプラン管理を使用すると、プランの安定性を最大限に高めることができます。これにより、新しいプランと古いプランのパフォーマンスが比較され、次善のプランよりも 10% 以上早いプランが承認または却下されます。

アップグレード後、`evolve_plan_baselines` 関数を使用して、クエリパラメータをバインドしながらアップグレード前とアップグレード後のプランのパフォーマンスを比較できます。以下のステップでは、「Aurora PostgreSQL 管理計画を使用する」で説明されているように、実稼働環境で承認済みのマネージドプランを使用していることを前提としています。

1. アップグレードする前に、クエリプランマネージャーを実行している状態でアプリケーションを実行します。アプリケーションの実行中にオプティマイザが新しいプランを発見したときは、それらも追加します。詳細については、「自動的な計画の取得」を参照してください。
2. 各プランのパフォーマンスを評価します。詳細については、「計画パフォーマンスの評価」を参照してください。
3. アップグレード後、`evolve_plan_baselines` 関数を使用して、承認済みのプランを再度分析します。クエリパラメータバインディングの使用前後のパフォーマンスを比較します。新しいプランが速い場合は、承認済みのプランに追加できます。同じパラメータバインディングの別のプランよりも速い場合は、遅いプランを「拒否」としてマークできます。

   詳細については、「より優れた計画の承認」を参照してください。この関数の詳細については、「apg_plan_mgmt.evolve_plan_baselines」を参照してください。

詳細については、「Amazon Aurora PostgreSQL 互換エディションのクエリ計画管理により、メジャーバージョンのアップグレード後に一貫したパフォーマンスを確保する」を参照してください。

> **Note**
>
> 論理レプリケーションまたはを使用してメジャーバージョンアップグレードを実行する場合はAWS DMS、アップグレードしたインスタンスに既存のプランが確実にコピーされるように、`apg_plan_mgmt`スキーマを必ず複製してください。論理レプリケーションの詳細につ

いては、「論理レプリケーションを使用して Aurora PostgreSQL のメジャーバージョンアップグレードを実行する」を参照してください。

パフォーマンス低下を検出して修復する事後対応型の計画管理

アプリケーションの実行をモニタリングすることによって、パフォーマンスの低下を引き起こす計画を検出できます。パフォーマンス低下を検出したときは、以下のステップに従って、不適切な計画を手動で拒否または修正します。

1. アプリケーションの実行中に、管理計画の使用を強制し、新しく検出された計画を未承認として自動的に追加します。詳細については、「Aurora PostgreSQL 管理計画を使用する」および「自動的な計画の取得」を参照してください。
2. 実行中のアプリケーションのパフォーマンス低下をモニタリングします。
3. 計画のパフォーマンス低下を発見した場合、計画のステータスを `rejected` に設定します。次に SQL ステートメントを実行する際、オプティマイザは拒否された計画を自動的に無視し、代わりに別の承認済み計画を使用します。詳細については、「低速な計画の拒否または無効化」を参照してください。

   場合によっては、不適切な計画を拒否、無効化、削除せずに、修正した方が良いこともあります。計画の改善を試すには、`pg_hint_plan` エクステンションを使用してください。`pg_hint_plan` では、特別なコメントを使用して、オプティマイザに通常の計画作成方法を上書きするよう指示します。詳細については、「pg_hint_plan を使用した計画の修正」を参照してください。

Aurora PostgreSQL のクエリ計画管理を理解する

Aurora PostgreSQL DB クラスターのクエリ計画管理を有効にすると、オプティマイザは複数回処理する SQL ステートメントのクエリ実行計画を生成して保存します。オプティマイザは、マネージドステートメントの初期に生成された計画のステータスを常に `Approved` に設定し、`dba_plans`ビューに保存します。

管理ステートメントの一連の承認済み計画は、計画ベースラインと呼ばれます。アプリケーションが実行されると、オプティマイザが管理ステートメントに追加の計画を生成することがあります。オプティマイザは、追加でキャプチャされた計画を `Unapproved` ステータスに設定します。

後で、Unapproved 計画が正常に機能するかどうかを確認し、Approved、Rejected、または Preferred に変更できます。そのためには、apg_plan_mgmt.evolve_plan_baselines 関数または apg_plan_mgmt.set_plan_status 関数を使用します。

オプティマイザが SQL ステートメントの計画を生成すると、クエリ計画管理はその計画を apg_plan_mgmt.plans テーブルに保存します。apg_plan_mgmt ロールが付与されたデータベースユーザーは、apg_plan_mgmt.dba_plans ビューをクエリすることで計画の詳細を確認できます。例えば、次のクエリでは、本番環境以外の Aurora PostgreSQL DB クラスターのビューに現在表示されている計画の詳細が一覧表示されます。

- sql_hash — SQL ステートメントの正規化されたテキストのハッシュ値である SQL ステートメントの識別子。
- plan_hash — sql_hash と計画のハッシュを組み合わせた一意の識別子。
- status - 計画のステータス。オプティマイザは、承認済みの計画を実行できます。
- enabled — 計画がすぐに使用できる (true) か、使用できない (false) かを示します。
- plan_outline – 実際の実行計画を再作成するために使用された計画の表現。ツリー構造内の演算子は EXPLAIN 出力の演算子にマップされます。

apg_plan_mgmt.dba_plans ビューには、計画が最後に使用された日時など、計画のすべての詳細を含むさらに多くの列があります。詳細については、「apg_plan_mgmt.dba_plans ビューのリファレンス」を参照してください。

正規化と SQL ハッシュ

apg_plan_mgmt.dba_plans ビューでは、SQL のハッシュ値により管理ステートメントを識別できます。SQL ハッシュは、リテラル値などのいくつかの違いを取り除く SQL ステートメントの正規化表現で計算されます。

各 SQL ステートメントの正規化プロセスでは、スペースと大文字と小文字が区別されないため、SQL ステートメントの要点を読んで理解することができます。正規化により、次の項目が削除されるか置き換えられます。

- 先頭のブロックコメント
- EXPLAIN キーワードと EXPLAIN オプション、EXPLAIN ANALYZE
- 末尾のスペース
- すべてのリテラル

例として、以下のステートメントを見てみましょう。

```
/*Leading comment*/ EXPLAIN SELECT /* Query 1 */ * FROM t WHERE x > 7 AND y = 1;
```

オプティマイザは、次に示すように、このステートメントを正規化します。

```
SELECT /* Query 1 */ * FROM t WHERE x > CONST AND y = CONST;
```

正規化を使用することで、リテラル値またはパラメータ値のみが異なる可能性がある類似の SQL ステートメントで同じ SQL ハッシュを使用できます。つまり、同じ SQL ハッシュに対して、異なる計画が異なる条件下で最適となる、複数の計画がある可能性があります。

> **Note**
>
> 異なるスキーマで使用される単一の SQL ステートメントは、実行時に特定のスキーマにバインドされるため、計画が異なります。プランナーはスキーマバインディングの統計を使用して最適な計画を選択します。

オプティマイザが計画を選択する方法の詳細については、「Aurora PostgreSQL 管理計画を使用する」を参照してください。このセクションでは、EXPLAIN および EXPLAIN ANALYZE の使用方法と、実際に使用される前に計画をプレビューする方法を学ぶことができます。詳細については、「オプティマイザが選択した計画の分析」を参照してください。計画を選択するプロセスの概要を示す画像については、を参照してくださいオプティマイザが実行する計画を選択する方法。。

Aurora PostgreSQL 実行計画のキャプチャ

Aurora PostgreSQL クエリ計画管理には、自動と手動の 2 つの異なるクエリ実行計画のキャプチャモードが用意されています。apg_plan_mgmt.capture_plans_baselines を automatic に、または manual に設定してモードを選択します。特定の SQL ステートメントの実行計画を取り込むには、手動計画取り込みを使用します。または、自動計画取り込みを使用して、アプリケーションの実行時に 2 回以上実行されるすべての (または最も遅い) 計画を取得することもできます。

オプティマイザは計画取り込み時に、管理ステートメントの初期に取得された計画のステータスを approved に設定します。オプティマイザは、管理ステートメント用に取得されたすべての追加の計画のステータスを unapproved に設定します。ただし、複数の計画が approved ステータスで保存される場合があります。例えば、1 つのステートメントに対して複数の計画が並行して作成された場合や、そのステートメントの初期の計画がコミットされる前に発生する可能性があります。

dba_plans ビューに取得して保管できる計画の最大数を制御するには、DB インスタンスレベルのパラメータグループで apg_plan_mgmt.max_plans パラメータを設定します。apg_plan_mgmt.max_plans パラメータを変更した場合、新しい値を有効にするために DB インスタンスを再起動する必要があります。詳細については、apg_plan_mgmt.max_plans パラメータを参照してください。

特定の SQL ステートメントの計画の手動取り込み

管理が必要な既知の SQL ステートメントがある場合は、ステートメントを SQL スクリプトファイルに入れてから、手動で計画を取得します。以下は、一連の SQL ステートメントに対してクエリ計画を手動で取得する方法を psql を例にとって示しています。

```
psql> SET apg_plan_mgmt.capture_plan_baselines = manual;
psql> \i my-statements.sql
psql> SET apg_plan_mgmt.capture_plan_baselines = off;
```

各 SQL ステートメントに対して計画を取得した後、オプティマイザは apg_plan_mgmt.dba_plans ビューに新しい行を追加します。

EXPLAIN ステートメントまたは EXPLAIN EXECUTE ステートメントを SQL スクリプトファイルで使用することをお勧めします。対象となるすべての計画の取り込みに十分なバリエーションがパラメータ値に含まれていることを確認します。

オプティマイザの最小コスト計画よりも優れた計画がある場合は、オプティマイザでその優れた計画が使用されるようになります。そのためには、オプティマイザのヒントを 1 つ以上指定します。詳細については、「pg_hint_plan を使用した計画の修正」を参照してください。unapproved 計画と approved 計画のパフォーマンスを比較して、承認、拒否、または削除する方法については、「計画パフォーマンスの評価」を参照してください。

自動的な計画の取得

以下のような状況では、自動計画キャプチャを使用します。

- 管理する特定の SQL ステートメントが分からない場合。
- 管理対象となる SQL ステートメントが数百、数千ある場合。
- ユーザーのアプリケーションがクライアント API を使用している場合。例えば、JDBC では、psql では表現できない、名前のない準備済みステートメントまたは一括モードのステートメントを使用します。

自動的に計画を取得するには

1. DB インスタンスレベルのパラメータグループで apg_plan_mgmt.capture_plan_baselines を automatic に設定して、自動計画取り込みをオンにします。詳細については、「DB パラメータグループのパラメータの変更」を参照してください。
2. DB インスタンスを再起動します。
3. アプリケーションが実行されると、オプティマイザは少なくとも 2 回以上実行される各 SQL ステートメントの計画を取得します。

   デフォルトのクエリ計画管理パラメータ設定を使用してアプリケーションが実行されると、オプティマイザは少なくとも 2 回以上実行される各 SQL ステートメントの計画を取得します。デフォルト設定を使用してすべての計画を取り込むことで、実行時のオーバーヘッドを大幅に削減し、実稼働環境で有効にすることができます。

自動計画取り込みをオフにするには

- DB インスタンスレベルのパラメータグループから apg_plan_mgmt.capture_plan_baselines パラメータを off に設定します。

未承認の計画のパフォーマンスを測定して、それらを承認、拒否、または削除する方法については、「計画パフォーマンスの評価」を参照してください。

Aurora PostgreSQL 管理計画を使用する

オプティマイザが管理ステートメントに対して取り込んだ計画を使用するには、パラメータ apg_plan_mgmt.use_plan_baselines を true に設定します。以下はローカルインスタンスの例です。

```
SET apg_plan_mgmt.use_plan_baselines = true;
```

この設定により、アプリケーションの実行中、オプティマイザでは、管理ステートメントごとに最小コストの計画、推奨される計画または承認済み計画が使用されるようになります。

オプティマイザが選択した計画の分析

apg_plan_mgmt.use_plan_baselines パラメータが true に設定されていれば、EXPLAIN ANALYZE SQL ステートメントを使用して、オプティマイザがステートメントを実行する場合に使用する計画を表示することができます。次に例を示します。

```
EXPLAIN ANALYZE EXECUTE rangeQuery (1,10000);
```

```
                                                        QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=393.29..393.30 rows=1 width=8) (actual time=7.251..7.251 rows=1
 loops=1)
   ->  Index Only Scan using t1_pkey on t1 t  (cost=0.29..368.29 rows=10000 width=0)
 (actual time=0.061..4.859 rows=10000 loops=1)
Index Cond: ((id >= 1) AND (id <= 10000))
         Heap Fetches: 10000
 Planning time: 1.408 ms
 Execution time: 7.291 ms
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1984047223, Plan Hash: 512153379
```

出力には、実行するベースラインの承認済み計画が表示されます。ただし、出力にはより低コストの計画が見つかったことがわかります。この場合、自動的な計画の取得 で説明されているように自動計画取り込みをオンにして、この新しい最小コスト計画を取得します。

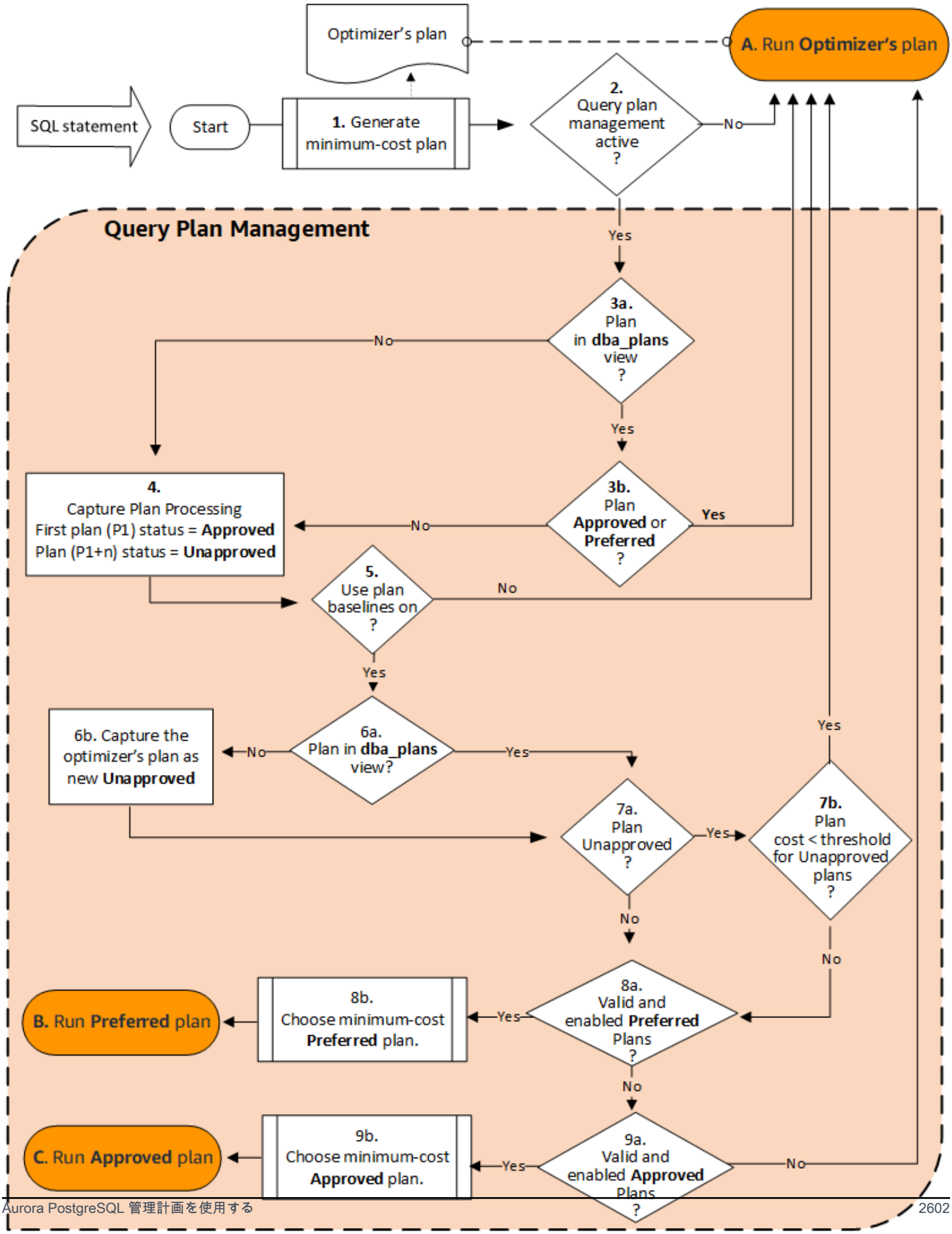
Unapproved新しい計画は常にオプティマイザによって次のようにキャプチャされます。計画を比較し、それらを承認済み、拒否、または無効に変更するに

は、apg_plan_mgmt.evolve_plan_baselines 関数を使用します。詳細については、「計画パフォーマンスの評価」を参照してください。

オプティマイザが実行する計画を選択する方法。

実行計画のコストは、オプティマイザが異なる計画を比較するために行う見積もりです。計画のコストを計算する際、オプティマイザではその計画で必要な CPU や I/O オペレーションなどの要素を考慮します。PostgreSQL クエリプランナーのコスト見積もりの詳細については、PostgreSQL のドキュメントの「Query Planning」(クエリ計画) を参照してください。

次のイメージは、クエリ計画管理が有効な場合とそうでない場合に、特定の SQL ステートメントに対して計画がどのように選択されるかを示しています。


Optimizer's plan
A. Run Optimizer's plan
SQL statement
Start
1. Generate minimum-cost plan
2. Query plan management active ?
No
Query Plan Management
Yes
3a. Plan in dba_plans view ?
No
Yes
4. Capture Plan Processing
First plan (P1) status = Approved
Plan (P1+n) status = Unapproved
3b. Plan Approved or Preferred ?
Yes
No
5. Use plan baselines on ?
No
Yes
6a. Plan in dba_plans view?
No
Yes
6b. Capture the optimizer's plan as new Unapproved
7a. Plan Unapproved ?
Yes
7b. Plan cost < threshold for Unapproved plans ?
Yes
No
No
8a. Valid and enabled Preferred Plans ?
Yes
8b. Choose minimum-cost Preferred plan.
B. Run Preferred plan
No
9a. Valid and enabled Approved Plans ?
Yes
9b. Choose minimum-cost Approved plan.
C. Run Approved plan
No

フローは次のとおりです。

1. オプティマイザでは、SQL ステートメントの最小コストの計画が生成されます。
2. クエリ計画管理が有効ではない場合、オプティマイザの計画が直ちに実行されます (A. オプティマイザの計画を実行)。クエリ計画管理は、apg_plan_mgmt.capture_plan_baselines と apg_plan_mgmt.use_plan_baselines のパラメータがどちらもデフォルト設定 (それぞれ「off」と「false」) の場合は無効になります。

   それ以外の場合は、クエリ計画管理が有効になります。この場合、SQL ステートメントとそれに対するオプティマイザの計画がさらに評価されてから計画が選択されます。

   > **Tip**
   >
   > apg_plan_mgmt ロールのデータベースユーザーは、必要に応じて計画をプロアクティブに比較する、計画のステータスを変更する、特定の計画を強制的に使用することができます。詳細については、「Aurora PostgreSQL 実行計画の管理」を参照してください。

3. SQL ステートメントには、過去にクエリ計画管理によって保存された計画が既に含まれている場合があります。計画は、その計画の作成に使用された SQL ステートメントに関する情報とともに apg_plan_mgmt.dba_plans に保存されます。計画に関する情報には、そのステータスが含まれます。計画のステータスによって、その計画が使用されているかどうかが次のように決まります。
   a. 計画が SQL ステートメントの保存計画に含まれていない場合は、特定の SQL ステートメントのオプティマイザによってこの特定の計画が初めて生成されたことになります。計画は、キャプチャ計画処理 (4) に送信されます。
   b. 計画が保存されている計画の中にあり、ステータスが「承認済み」または「優先」の場合、その計画が実行されます (A. オプティマイザの計画を実行)。

      計画が保存されている計画に含まれていても、承認済みでも優先でもない場合、計画はキャプチャ計画処理 (4) に送信されます。
4. 特定の SQL ステートメントの計画が初めてキャプチャされると、計画のステータスは常に承認済み (P1) に設定されます。その後、オプティマイザが同じ SQL ステートメントに対して同じ計画を生成すると、その計画のステータスは未承認 (P1+n) に変更されます。

   計画がキャプチャされ、ステータスが更新されると、次のステップ (5) で評価が継続されます。

5. 計画のベースラインは、さまざまなステータスでの SQL ステートメントの履歴と計画で構成されています。クエリ計画管理では、計画のベースラインを使用するオプションがオンになっているかどうかによって、次のように計画を選択する際にベースラインを考慮できます。
   - 計画ベースラインの使用は「オフ」の場合、apg_plan_mgmt.use_plan_baselines パラメータはデフォルト値 (false) に設定されています。計画は、実行前にベースラインと比較されません (A. オプティマイザの計画を実行)。
   - 計画ベースラインの使用が「オン」の場合、apg_plan_mgmt.use_plan_baselines パラメータは true に設定されます。計画はベースライン (6) を使用してさらに評価されます。
6. この計画は、ベースラインのステートメントの他の計画と比較されます。
   a. オプティマイザの計画がベースラインの計画に含まれる場合、そのステータスがチェックされます (7a)。
   b. オプティマイザの計画がベースラインの計画にない場合、その計画は新規の Unapproved 計画としてステートメントの計画に追加されます。
7. 未承認の場合のみ、計画のステータスを確認します。
   a. 計画のステータスが [Unapproved] (未承認) の場合、計画のコストの見積もりは、未承認の実行計画のしきい値に指定されたコストの見積もりと比較されます。
      - 計画のコストの見積もりがしきい値を下回る場合、オプティマイザでは [Unapproved] (未承認) の計画であってもその計画を使用します (A. オプティマイザの計画を実行)。通常、オプティマイザは [Unapproved] (未承認) の計画を実行しません。ただし、apg_plan_mgmt.unapproved_plan_execution_threshold パラメータでコストのしきい値を指定すると、オプティマイザは [Unapproved] (未承認) の計画のコストをしきい値と比較します。コストの見積もりがしきい値を下回る場合、オプティマイザは計画を実行します。詳細については、「apg_plan_mgmt.unapproved_plan_execution_threshold」を参照してください。
      - 計画のコストの見積もりがしきい値を下回っていない場合は、計画の他の属性がチェックされます (8a)。
   b. 計画のステータスが [Unapproved] (未承認) 以外の場合、他の属性が確認されます (8a)。
8. オプティマイザは、無効の計画を使用しません。つまり、enable 属性が「f」 (false) に設定されている計画です。また、オプティマイザはステータスが Rejected (拒否) の計画を使用しません。

   オプティマイザは、無効の計画を使用できません。管理計画が依存するオブジェクト (インデックスやテーブルパーティションなど) が削除されると、時間の経過とともに計画が無効になる可能性があります。

   a. ステートメントに有効な推奨計画がある場合、オプティマイザではこの SQL ステートメントに保存されている推奨計画の中から最小コストの計画を選択します。その後、オプティマイザは最小コストの推奨計画を実行します。
   b. そのステートメントに有効化された計画や、有効で推奨される計画がない場合は、次のステップ (9) で評価されます。
9. そのステートメントに有効な推奨計画がある場合、オプティマイザではこの SQL ステートメントに保存されている推奨計画の中から最小コストの計画を選択します。その後、オプティマイザは最小コストの承認済み計画を実行します。

   そのステートメントに有効化された計画や、有効で推奨される計画がない場合、オプティマイザは最小コスト計画 (A. オプティマイザの計画を実行) を使用します。

dba_plans ビューで Aurora PostgreSQL クエリ計画を検証する

`apg_plan_mgmt`ロールを付与されたデータベースユーザーと管理者は、に保存されている計画を表示および管理できます`apg_plan_mgmt.dba_plans`。Aurora PostgreSQL DB クラスターの管理者 (`rds_superuser`権限を持つ人) は、クエリ計画管理を行う必要があるデータベースユーザーにこのロールを明示的に付与する必要があります。

`apg_plan_mgmt`ビューには、Aurora PostgreSQL DB クラスターのライターインスタンス上のすべてのデータベースのすべてのマネージド SQL ステートメントの計画履歴が含まれます。このビューでは、計画、その状態、最終使用日時、その他すべての関連詳細を確認できます。

「正規化と SQL ハッシュ」で説明しているように、管理計画はそれぞれ、SQL ハッシュ値と計画ハッシュ値を組み合わせて識別されます。これらの識別子を使用すると、Amazon RDS Performance Insights などのツールを使用して個別の計画のパフォーマンスを追跡できます。Performance Insights の詳細については、「Amazon RDS Performance Insights の使用」を参照してください。

管理計画のリスト化

管理計画をリスト化するには、`apg_plan_mgmt.dba_plans` ビューの SELECT ステートメントを使用します。次の例では、`dba_plans` ビューに、承認済みの計画および未承認の計画を識別する `status` などの列が表示されます。

```
SELECT sql_hash, plan_hash, status, enabled, stmt_name
FROM apg_plan_mgmt.dba_plans;
```

```
 sql_hash   | plan_hash |   status   | enabled | stmt_name
------------+-----------+------------+---------+------------
 1984047223 | 512153379 | Approved   | t       | rangequery
 1984047223 | 512284451 | Unapproved | t       | rangequery
 (2 rows)
```

読みやすくするために、表示されるクエリと出力には、dba_plans ビューの一部の列のみがリストされています。詳細については、「apg_plan_mgmt.dba_plans ビューのリファレンス」を参照してください。

Aurora PostgreSQL 実行計画の管理

クエリ計画管理は、実行計画を追加、維持、および改善するための手法と機能を提供します。

計画パフォーマンスの評価

オプティマイザが計画を未承認として取得した後、apg_plan_mgmt.evolve_plan_baselines 関数を使用して、実際のパフォーマンスに基づいて計画を比較します。パフォーマンステストの結果に応じて、計画のステータスを未承認から承認済みまたは拒否に変更できます。要件に合わない場合は、代わりに apg_plan_mgmt.evolve_plan_baselines 関数を使用して計画を一時的に無効にすることができます。

より優れた計画の承認

以下の例は、apg_plan_mgmt.evolve_plan_baselines 関数を使用して、管理計画のステータスを承認済みに変更する方法を示しています。

```
SELECT apg_plan_mgmt.evolve_plan_baselines (
   sql_hash,
   plan_hash,
   min_speedup_factor := 1.0,
   action := 'approve'
)
FROM apg_plan_mgmt.dba_plans WHERE status = 'Unapproved';
```

```
NOTICE:     rangequery (1,10000)
NOTICE:     Baseline   [ Planning time 0.761 ms, Execution time 13.261 ms]
NOTICE:     Baseline+1 [ Planning time 0.204 ms, Execution time 8.956 ms]
NOTICE:     Total time benefit: 4.862 ms, Execution time benefit: 4.305 ms
NOTICE:     Unapproved -> Approved
```

```
evolve_plan_baselines
----------------------
0
(1 row)
```

出力は、1 と 10,000 のパラメータのバインディングを持つ rangequery ステートメントのパフォーマンス報告を示しています。新しい未承認の計画 (Baseline+1) は、以前に承認された最も良い計画 (Baseline) よりも優れています。新しい計画が Approved になったことを確認するには、apg_plan_mgmt.dba_plans ビューをチェックします。

```
SELECT sql_hash, plan_hash, status, enabled, stmt_name
FROM apg_plan_mgmt.dba_plans;
```

```
sql_hash  | plan_hash |  status  | enabled | stmt_name
------------+-----------+----------+---------+------------
1984047223 | 512153379 | Approved | t       | rangequery
1984047223 | 512284451 | Approved | t       | rangequery
(2 rows)
```

管理計画には、ステートメントの計画ベースラインである 2 つの承認済み計画が含まれるようになりました。apg_plan_mgmt.set_plan_status 関数を呼び出して、計画のステータスフィールドを直接 'Approved'、'Rejected'、'Unapproved'、または 'Preferred' に設定することもできます。

低速な計画の拒否または無効化

計画を拒否または無効化するには、'reject' または 'disable' をアクションパラメータとして apg_plan_mgmt.evolve_plan_baselines 関数に渡します。この例では、ステートメントの最適な Unapproved 計画と比較して 10% 以上低速なキャプチャ済み Approved 計画を無効にします。

```
SELECT apg_plan_mgmt.evolve_plan_baselines(
sql_hash,  -- The managed statement ID
plan_hash, -- The plan ID
1.1,       -- number of times faster the plan must be
'disable'  -- The action to take. This sets the enabled field to false.
)
FROM apg_plan_mgmt.dba_plans
WHERE status = 'Unapproved' AND   -- plan is Unapproved
origin = 'Automatic';       -- plan was auto-captured
```

直接、計画を拒否または無効に設定することもできます。計画の有効フィールドを直接 true または false に設定するには、apg_plan_mgmt.set_plan_enabled 関数を呼び出します。計画のステータスフィールドを直接 'Approved'、'Rejected'、'Unapproved'、または 'Preferred' に設定するには、apg_plan_mgmt.set_plan_status 関数を呼び出します。

計画の検証

利用不可能な計画を削除、または無効にするには、apg_plan_mgmt.validate_plans 関数を使用します。

管理計画が依存するオブジェクト (インデックスやテーブルなど) が削除されると、計画が利用不可能、または停滞となる可能性があります。ただし、削除されたオブジェクトが再作成されれば、計画が利用不可能なのは一時的です。利用不可能な計画が後で利用可能になる可能性がある場合は、利用不可能な計画を無効にするか、または削除せずに何もしないでいることができます。

利用不可能で、かつ過去 1 週間に使用されていないすべての計画を検索して削除するには、apg_plan_mgmt.validate_plans 関数を以下のように使用します。

```
SELECT apg_plan_mgmt.validate_plans(sql_hash, plan_hash, 'delete')
FROM apg_plan_mgmt.dba_plans
WHERE last_used < (current_date - interval '7 days');
```

計画を直接有効、または無効にするには、apg_plan_mgmt.set_plan_enabled 関数を使用します。

pg_hint_plan を使用した計画の修正

クエリオプティマイザは、すべてのステートメントに対して最適な計画を見つけるように設計されています。ほとんどの場合、オプティマイザは優れた計画を見つけます。ただし、オプティマイザが生成する計画よりもはるかに優れた計画が存在することがあります。オプティマイザに理想的な計画を生成させるために推奨される 2 つの方法は、pg_hint_plan エクステンションを使用すること、または PostgreSQL で Grand Unified Configuration (GUC) 可変を設定することです。

- pg_hint_plan エクステンション - PostgreSQL の pg_hint_plan エクステンションを使用して、プランナーの動作を変更するための「ヒント」を指定します。pg_hint_plan エクステンションのインストールおよび使用方法の詳細については、「pg_hint_plan ドキュメント」を参照してください。
- GUC 可変 - 1 つ以上のコストモデルパラメータ、または from_collapse_limit や GEQO_threshold などの他のオプティマイザパラメータを上書きします。

これらの手法のいずれかを使用してクエリオプティマイザに計画の使用を強制する場合、クエリ計画管理を使用して、新しい計画を取得した後で強制的に使用することもできます。

pg_hint_plan エクステンションを使用して、SQL ステートメントの結合の順序、結合メソッド、またはアクセスパスを変更することができます。オプティマイザによる計画の作成方法を変更するには、特別な pg_hint_plan 構文を持つ SQL コメントを使用します。例えば、問題のある SQL ステートメントに双方向の結合があるとします。

```
SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

次に、オプティマイザが結合順序 (t1、t2) を選択したとします。しかし、ユーザーは結合順序 (t2、t1) の方が速いことを知っています。以下のヒントは、より高速な結合順序 (t2、t1) を使用するようオプティマイザに強制します。オプティマイザが SQL ステートメントの計画を生成しても、そのステートメントを実行せずに、EXPLAIN を含めます。(出力は表示されていません。)

```
/*+ Leading ((t2 t1)) */ EXPLAIN SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

以下のステップは、pg_hint_plan の使用方法を示しています。

オプティマイザの生成した計画を変更し、pg_hint_plan を使用して計画を取得するには

1. 手動取り込みモードをオンにします。

   ```
   SET apg_plan_mgmt.capture_plan_baselines = manual;
   ```

2. 目的の SQL ステートメントのヒントを指定してください。

   ```
   /*+ Leading ((t2 t1)) */ EXPLAIN SELECT *
   FROM t1, t2
   WHERE t1.id = t2.id;
   ```

   これが実行された後、オプティマイザは apg_plan_mgmt.dba_plans ビューで計画をキャプチャします。クエリ計画管理は先頭のコメントを削除することでステートメントを正規化するため、キャプチャされた計画には特別な pg_hint_plan コメント構文は含まれません。

3. apg_plan_mgmt.dba_plans ビューを使用して管理計画を表示します。

```
SELECT sql_hash, plan_hash, status, sql_text, plan_outline
FROM apg_plan_mgmt.dba_plans;
```

4. 計画のステータスを Preferred に設定します。これにより、最小コスト計画がまだ Approved や Preferred になっていない場合、オプティマイザは一連の承認済み計画から選択せずに、その計画を実行することを選択します。

```
SELECT apg_plan_mgmt.set_plan_status(sql-hash, plan-hash, 'preferred' );
```

5. 手動計画取り込みをオフにして、管理計画の使用を強制します。

```
SET apg_plan_mgmt.capture_plan_baselines = false;
SET apg_plan_mgmt.use_plan_baselines = true;
```

これで、元の SQL ステートメントが実行されると、オプティマイザは Approved 計画または Preferred 計画のいずれかを選択します。最小コスト計画が Approved や Preferred でない場合、オプティマイザは Preferred 計画を選択します。

計画の削除

計画は、1 か月以上、具体的には 32 日間使用されなかった場合、自動的に削除されます。これは、apg_plan_mgmt.plan_retention_period パラメータのデフォルト設定です。計画の保持期間をより長い期間に変更できますが、最小期間は 1 です。計画が最後に使用された日付から、last_used の日付を現在の日付から引いて日数を算出します。last_used の日付は、オプティマイザが最小コスト計画としてプランを選択したか、プランが実行された最新の日付です。計画の日付は apg_plan_mgmt.dba_plans ビューに保存されます。

長期間使用されていない、または有用ではない計画を削除することをお勧めします。すべての計画には last_used の日付があり、オプティマイザは計画を実行するたびに更新するか、ステートメントの最小コスト計画として計画を選択します。最後の last_used の日付を確認して、安全に削除できる計画を確認してください。

次のクエリは、プランの総数、削除に失敗したプラン、削除に成功したプランの数を含む 3 列のテーブルを返します。apg_plan_mgmt.delete_plan 関数を使用して過去 31 日以内に最小コスト計画として選択されておらず、ステータスが Rejected ではないすべての計画を削除する方法の例であるネストしたクエリが入っています。

```
SELECT (SELECT COUNT(*) from apg_plan_mgmt.dba_plans) total_plans,
```

```
       COUNT(*) FILTER (WHERE result = -1) failed_to_delete,
       COUNT(*) FILTER (WHERE result = 0) successfully_deleted
       FROM (
            SELECT apg_plan_mgmt.delete_plan(sql_hash, plan_hash) as result
            FROM apg_plan_mgmt.dba_plans
            WHERE last_used < (current_date - interval '31 days')
            AND status <> 'Rejected'
            ) as dba_plans ;
```

```
 total_plans | failed_to_delete | successfully_deleted
-------------+------------------+----------------------
           3 |                0 |                    2
```

詳細については、「apg_plan_mgmt.delete_plan」を参照してください。

有効ではなく、無効のままになると見られる計画を削除するには、apg_plan_mgmt.validate_plans 関数を使用します。この関数により、無効な計画を削除または無効にすることができます。詳細については、「計画の検証」を参照してください。

> **Important**
>
> 無関係な計画を削除しないと、クエリプラン管理用に確保されている共有メモリが不足する可能性があります。マネージド計画に使用可能なメモリー量を制御するには、apg_plan_mgmt.max_plans パラメータを使用します。カスタム DB インスタンスのパラメータグループでこのパラメータを設定し、変更を有効にするために DB インスタンスを再起動します。詳細については、apg_plan_mgmt.max_plans パラメータを参照してください。

計画のエクスポートおよびインポート

管理計画をエクスポートしたり、別の DB インスタンスにインポートできます。

管理計画をエクスポートする

承認されたユーザーは apg_plan_mgmt.plans テーブルの任意のサブセットを別のテーブルにコピーしてから、pg_dump コマンドを使用してそれを保存することができます。次に例を示します。

```
CREATE TABLE plans_copy AS SELECT *
```

```
FROM apg_plan_mgmt.plans [ WHERE predicates ] ;
```

```
% pg_dump --table apg_plan_mgmt.plans_copy -Ft mysourcedatabase > plans_copy.tar
```

```
DROP TABLE apg_plan_mgmt.plans_copy;
```

管理計画をインポートする

1. エクスポートした管理計画の .tar ファイルを、計画を復元する予定のシステムにコピーします。
2. tar ファイルを新しいテーブルにコピーするには、pg_restore コマンドを使用します。

   ```
   % pg_restore --dbname mytargetdatabase -Ft plans_copy.tar
   ```

3. 次の例に示すように、plans_copy テーブルを apg_plan_mgmt.plans テーブルとマージします。

   > **Note**
   >
   > 場合によっては、あるバージョンの apg_plan_mgmt エクステンションからダンプして別のバージョンに復元することがあります。このような場合は、計画テーブルの列が異なる可能性があります。その場合は、SELECT * を使用せず、列に明示的に名前を付けてください。

```
INSERT INTO apg_plan_mgmt.plans SELECT * FROM plans_copy
 ON CONFLICT ON CONSTRAINT plans_pkey
 DO UPDATE SET
 status = EXCLUDED.status,
 enabled = EXCLUDED.enabled,
 -- Save the most recent last_used date
 --
 last_used = CASE WHEN EXCLUDED.last_used > plans.last_used
 THEN EXCLUDED.last_used ELSE plans.last_used END,
 -- Save statistics gathered by evolve_plan_baselines, if it ran:
 --
 estimated_startup_cost = EXCLUDED.estimated_startup_cost,
 estimated_total_cost = EXCLUDED.estimated_total_cost,
 planning_time_ms = EXCLUDED.planning_time_ms,
 execution_time_ms = EXCLUDED.execution_time_ms,
```

```
  total_time_benefit_ms = EXCLUDED.total_time_benefit_ms,
  execution_time_benefit_ms = EXCLUDED.execution_time_benefit_ms;
```

4. 管理計画を共有メモリにリロードし、一時的な計画テーブルを削除します。

```
SELECT apg_plan_mgmt.reload(); -- refresh shared memory
DROP TABLE plans_copy;
```

Aurora PostgreSQL クエリ計画管理のリファレンス

以下に、Aurora PostgreSQL のいくつかのクエリ計画管理機能のリファレンス情報を示します。

トピック

- Aurora PostgreSQL クエリ計画管理のパラメータリファレンス
- Aurora PostgreSQL クエリ計画管理の関数リファレンス
- apg_plan_mgmt.dba_plans ビューのリファレンス

Aurora PostgreSQL クエリ計画管理のパラメータリファレンス

このセクションに記載されているパラメータを使用して、apg_plan_mgmt 拡張機能を設定できます。これらは、カスタム DB クラスターパラメータと Aurora PostgreSQL DB クラスターに関連付けられた DB パラメータグループで使用できます。これらのパラメータは、クエリ計画管理機能の動作と、それがオプティマイザに与える影響を制御します。クエリ計画管理のセットアップの詳細については、「Aurora PostgreSQL のクエリプラン管理をオンにする」を参照してください。次のパラメータを変更しても、apg_plan_mgmt 拡張機能はそのセクションで詳述されているように設定されていない場合は効果がありません。パラメータの変更については、「DB クラスターパラメータグループのパラメータの変更」および「DB インスタンスでの DB パラメータグループの使用」を参照してください。

パラメータ

- apg_plan_mgmt.capture_plan_baselines
- apg_plan_mgmt.plan_capture_threshold
- apg_plan_mgmt.explain_hashes
- apg_plan_mgmt.log_plan_Enforcement_result
- apg_plan_mgmt.max_databases
- apg_plan_mgmt.max_plans

- apg_plan_mgmt.plan_hash_version
- apg_plan_mgmt.plan_retention_period
- apg_plan_mgmt.unapproved_plan_execution_threshold
- apg_plan_mgmt.use_plan_baselines
- auto_explain.hashes

apg_plan_mgmt.capture_plan_baselines

各 SQL ステートメントのオプティマイザによって生成されたクエリ実行計画をキャプチャし、dba_plans ビューに保存します。デフォルトでは、apg_plan_mgmt.max_plans パラメータで指定された保存可能な最大の計画数は 10,000 です。参考情報については、「apg_plan_mgmt.max_plans」を参照してください。

このパラメータは、カスタム DB クラスターのパラメータグループまたはカスタム DB パラメータグループで設定できます。このパラメータの値を変更しても、再起動は必要ありません。

デフォルト	許可される値	説明
オフ	自動	DB インスタンス上のすべてのデータベースの計画キャプチャを有効にします。2 回以上実行される SQL ステートメントごとに計画を収集します。大規模なワークロードや進化するワークロードにこの設定を使用すると、計画を安定化できます。
	手動	計画キャプチャは、再度オフにするまで、後続のステートメントでのみ有効です。この設定を使用すると、特定の重要な SQL ステートメントのみ、または問題がある既知のクエリのクエリ実行計画をキャプチャできます。
	オフ	計画のキャプチャをオフにします。

詳細については、「Aurora PostgreSQL 実行計画のキャプチャ」を参照してください。

apg_plan_mgmt.plan_capture_threshold

クエリ実行プランの合計コストがしきい値を下回った場合、プランがビューにキャプチャされないようにしきい値を指定します。apg_plan_mgmt.dba_plans

このパラメータの値を変更しても、再起動は必要ありません。

デフォルト	許可される値	説明
0	0 - 1.79769e+308	プランをキャプチャするための apg_plan_mgmt　クエリプランの合計実行コストのしきい値を設定します。

詳細については、「dba_plans ビューで Aurora PostgreSQL クエリ計画を検証する」を参照してください。

apg_plan_mgmt.explain_hashes

EXPLAIN [ANALYZE] が出力の最後に sql_hash と plan_hash を表示するかどうかを指定します。このパラメータの値を変更しても、再起動は必要ありません。

デフォルト	許可される値	説明
0	0 (オフ)	EXPLAIN は、hashes true オプションが指定されていないと、sql_hash と plan_hash を表示しません。
	1 (オン)	EXPLAIN は、hashes true オプションが指定されていないときでも、sql_hash と plan_hash を表示します。

apg_plan_mgmt.log_plan_Enforcement_result

QPM 管理プランが適切に使用されているかどうかを確認するために結果を記録する必要があるかどうかを指定します。保存されているジェネリックプランを使用すると、ログファイルにレコードは書き込まれません。このパラメータの値を変更しても、再起動は必要ありません。

デフォルト	許可される値	説明
none	none	プラン実施結果はログファイルに表示されません。

デフォルト	許可される値	説明
	on_error	QPM が管理プランを使用できなかった場合にのみ、プラン実施結果をログファイルに表示します。
	すべて	成功と失敗の両方を含むすべてのプラン実施結果をログファイルに表示します。

apg_plan_mgmt.max_databases

Aurora PostgreSQL DB クラスターのライターインスタンスで、クエリ計画管理を使用できるデータベースの最大数を指定します。デフォルトでは、最大 10 個のデータベースでクエリ計画管理を使用できます。インスタンスに 10 個を超えるデータベースがある場合は、この設定値を変更できます。特定のインスタンスにあるデータベースの数を確認するには、psql を使用してインスタンスに接続します。次に、psql メタコマンドの \l を使用して、データベースを一覧表示します。

このパラメータの値を変更した場合、設定を有効にするためにインスタンスを再起動する必要があります。

デフォルト	許可される値	説明
10	10-2147483647	インスタンスでクエリ計画管理を使用できるデータベースの最大数です。

このパラメータは、カスタム DB クラスターのパラメータグループまたはカスタム DB パラメータグループで設定できます。

apg_plan_mgmt.max_plans

クエリプランマネージャが apg_plan_mgmt.dba_plans ビューで保持できる SQL ステートメントの最大数を設定します。Aurora PostgreSQL のすべてのバージョンで、このパラメータを 10000 以上に設定することをお勧めします。

このパラメータは、カスタム DB クラスターのパラメータグループまたはカスタム DB パラメータグループで設定できます。このパラメータの値を変更した場合、設定を有効にするためにインスタンスを再起動する必要があります。

デフォルト	許可される値	説明
10000	10-2147483647	apg_plan_mgmt.dba_plans ビューに保存できる計画の最大数。 Aurora PostgreSQL バージョン 10 以前では、デフォルトは 1,000 です。

詳細については、「dba_plans ビューで Aurora PostgreSQL クエリ計画を検証する」を参照してください。

apg_plan_mgmt.plan_hash_version

plan_hash 計算が対象とするユースケースを指定します。apg_plan_mgmt.plan_hash_version の上位バージョンは、下位バージョンのすべての機能をカバーします。例えば、バージョン 3 はバージョン 2 でサポートされるユースケースをカバーします。

このパラメータの値を変更した後は、apg_plan_mgmt.validate_plans('update_plan_hash') を呼び出す必要があります。apg_plan_mgmt がインストールされている各データベースの plan_hash 値とプランテーブルのエントリを更新します。詳細については、「計画の検証」を参照してください。

デフォルト	許可される値	説明
1	1	デフォルトの plan_hash 計算。
	2	plan_hash 計算がマルチスキーマをサポートするように変更されました。
	3	plan_hash 計算が、マルチスキーマをサポートし、パーティション化されたテーブルをサポートするように変更されました。
	4	plan_hash 計算が並列演算子に対応し、マテリアライズノードをサポートするように変更されました。

apg_plan_mgmt.plan_retention_period

apg_plan_mgmt.dba_plans ビューで計画を保持する日数を指定し、それ以降は自動的に削除されます。デフォルトでは、計画は最後に使用されてから 32 日が経過すると削除されます (apg_plan_mgmt.dba_plans ビューの last_used 列)。この設定は、1 以上の任意の数に変更できます。

このパラメータの値を変更した場合、設定を有効にするためにインスタンスを再起動する必要があります。

デフォルト	許可される値	説明
32	1-2147483647	計画が最後に使用された日付から、計画が自動的に削除されるまでの最大日数。

詳細については、「dba_plans ビューで Aurora PostgreSQL クエリ計画を検証する」を参照してください。

apg_plan_mgmt.unapproved_plan_execution_threshold

オプティマイザによって未承認の計画を使用可能なコストのしきい値を指定します。デフォルトでは、しきい値は 0 で、オプティマイザは未承認の計画を実行しません。このパラメータを 100 などのごく低いコストのしきい値に設定すると、小さなプランに対するプラン適用のオーバーヘッドを回避できます。リアクティブスタイルのプラン管理では、このパラメータを 10000000 など、非常に大きな値に設定することもできます。これにより、オプティマイザはプラン適用のオーバーヘッドなしに、選択したすべてのプランを使用できます。ただし、不適切なプランが見つかった場合は、手動で「拒否」とマークして、次回使用されないようにすることができます。

このパラメータの値は、特定の計画を実行するためのコスト見積もりを表します。未承認計画が推定コストを下回る場合、オプティマイザはその計画を SQL ステートメントに使用します。dba_plans ビューで、キャプチャされた計画とそのステータス (承認済み、未承認) を確認できます。詳細については、「dba_plans ビューで Aurora PostgreSQL クエリ計画を検証する」を参照してください。

このパラメータの値を変更しても、再起動は必要ありません。

デフォルト	許可される値	説明
0	0-2147483647	未承認の計画が使用された場合、それ以下の見積もりの計画コスト

詳細については、「Aurora PostgreSQL 管理計画を使用する」を参照してください。

apg_plan_mgmt.use_plan_baselines

apg_plan_mgmt.dba_plans ビューでオプティマイザがキャプチャされ、保存された承認済み計画のいずれかを使用するように指定します。デフォルトでは、このパラメータはオフ (false) になっているため、オプティマイザが生成した最小コスト計画をそのまま使用します。このパラメータをオンにする (true に設定する) と、オプティマイザは計画ベースラインからステートメントのクエリ実行計画を選択するよう強制されます。詳細については、「Aurora PostgreSQL 管理計画を使用する」を参照してください。このプロセスの詳細を示すイメージを検索するには、「オプティマイザが実行する計画を選択する方法。」を参照してください。

このパラメータは、カスタム DB クラスターのパラメータグループまたはカスタム DB パラメータグループで設定できます。このパラメータの値を変更しても、再起動は必要ありません。

デフォルト	許可される値	説明
false	true	apg_plan_mgmt.dba_plans から、承認済み、優先、未承認のいずれかの計画を使用してください。いずれもオプティマイザの評価基準を満たさない場合は、オプティマイザが生成した最小コスト計画を使用できます。詳細については、「オプティマイザが実行する計画を選択する方法。」を参照してください。
	false	オプティマイザが生成した最小コスト計画を使用します。

必要に応じて、キャプチャしたさまざまな計画の応答時間を評価し、計画のステータスを変更できます。詳細については、「Aurora PostgreSQL 実行計画の管理」を参照してください。

auto_explain.hashes

auto_explain の出力に sql_hash と plan_hash を表示するかどうかを指定します。このパラメータの値を変更しても、再起動は必要ありません。

デフォルト	許可される値	説明
0 (オフ)	0 (オフ)	auto_explain 結果に sql_hash と plan_hash は表示されません。
	1 (オン)	auto_explain 結果に sql_hash と plan_hash が表示されます。

Aurora PostgreSQL クエリ計画管理の関数リファレンス

apg_plan_mgmt エクステンションでは、以下の関数を使用できます。

関数

- apg_plan_mgmt.copy_outline
- apg_plan_mgmt.delete_plan
- apg_plan_mgmt.evolve_plan_baselines
- apg_plan_mgmt.get_explain_plan
- apg_plan_mgmt.plan_last_used
- apg_plan_mgmt.reload
- apg_plan_mgmt.set_plan_enabled
- apg_plan_mgmt.set_plan_status
- apg_plan_mgmt.update_plans_last_used
- apg_plan_mgmt.validate_plans

apg_plan_mgmt.copy_outline

特定の SQL プランハッシュとプランアウトラインをターゲットの SQL プランハッシュとアウトラインにコピーして、ターゲットのプランハッシュとアウトラインを上書きします。この関数は apg_plan_mgmt 2.3 以降のリリースで使用できます。

構文

```
apg_plan_mgmt.copy_outline(
    source_sql_hash,
    source_plan_hash,
    target_sql_hash,
    target_plan_hash,
    force_update_target_plan_hash
)
```

戻り値

コピーが成功したときには、0 を返します。無効な入力に対して例外をレイズします。

パラメータ

Parameter	説明
source_sql_hash	ターゲットクエリにコピーする plan_hash に関連付けられた sql_hash ID。
source_plan_hash	ターゲットクエリにコピーする plan_hash ID。
target_sql_hash	ソースプランハッシュとアウトラインで更新するクエリの sql_hash ID。
target_plan_hash	ソースプランハッシュとアウトラインで更新するクエリの plan_hash ID。
force_update_target_plan_hash	(オプション) ソースプランが target_sql_hash に対して再現可能ではない場合でも、クエリの target_plan_hash ID は更新されます。true に設定すると、この関数を使用して、リレーション名と列が一貫しているスキーマ間で計画をコピーできます。

使用に関する注意事項

この関数を使用すると、ヒントを使用するプランハッシュとプランアウトラインを他の同様のステートメントにコピーできるため、ターゲットステートメントに出現するたびにインラインヒントステートメントを使用する必要がなくなります。更新されたターゲットクエリの結果、無効なプランになった場合、この関数はエラーをレイズして、試行された更新をロールバックします。

apg_plan_mgmt.delete_plan

管理計画を削除します。

構文

```
apg_plan_mgmt.delete_plan(
    sql_hash,
    plan_hash
)
```

戻り値

削除が成功した場合は 0 を返し、削除が失敗した場合は -1 を返します。

パラメータ

Parameter	説明
sql_hash	計画の管理 SQL ステートメントの sql_hash ID。
plan_hash	管理計画の plan_hash ID。

apg_plan_mgmt.evolve_plan_baselines

既に承認された計画が速いか、またはクエリオプティマイザによって最小コスト計画として識別された計画が速いかを確認します。

構文

```
apg_plan_mgmt.evolve_plan_baselines(
    sql_hash,
    plan_hash,
    min_speedup_factor,
```

```
    action
)
```

戻り値

最良の承認済み計画より遅かった計画の数。

パラメータ

Parameter	説明
sql_hash	計画の管理 SQL ステートメントの sql_hash ID。
plan_hash	管理計画の plan_hash ID。同じ sql_hash ID 値を持つすべての計画を意味するために NULL を使用します。
min_speedup_factor	最小高速化係数は、計画を承認するために最も速い承認済みの計画よりも速い回数です。または、計画がそれを拒否または無効にするよりも遅い回数を示します。 これは正の浮動値です。
action	関数が実行するアクション。有効な値には次のようなものがあります。大文字と小文字は区別されません。 • 'disable' - 最小高速化係数を満たさない各マッチング計画を無効にします。 • 'approve' - 最小高速化係数を満たす各マッチング計画を有効にし、そのステータスを と設定します。approved • 'reject' - 最小高速度係数を満たさない各マッチング計画について、そのステータスを と設定します。rejected • NULL - 関数は最小高速度係数を満たさないため、パフォーマンス上の利点がない計画の数を単に返します。

使用に関する注意事項

計画と実行時間が、最も速い承認済計画よりも設定可能な要素だけ速いかどうかに基づき指定された計画を承認済み、拒否、または無効に設定します。パフォーマンス基準を満たす計画を自動的に承認

または拒否するには、アクションパラメータを 'approve' または 'reject' に設定します。あるいは、パフォーマンス実験を実行してレポートを作成するために '' (空の文字列) に設定することもできますが、何も実行されません。

最近実行されたプランに対して apg_plan_mgmt.evolve_plan_baselines 関数を無意味に再実行するのを防ぐことができます。そのためには、計画を最近作成された未承認の計画だけに制限します。あるいは、最近の apg_plan_mgmt.evolve_plan_baselines タイムスタンプを持つ承認済み計画で last_verified 関数を実行しないようにすることもできます。

ベースライン内の他の計画に対して、各計画の計画と実行時間を比較するためのパフォーマンス実験を実行します。場合によっては 1 つのステートメントに対して 1 つの計画しかなく、その計画が承認されます。このような場合は、計画の計画および実行時間、および計画を使用していない計画および実行時間を比較します。

各計画の増分利益 (またはデメリット) は、apg_plan_mgmt.dba_plans 列の total_time_benefit_ms ビューに記録されます。この値が正の値の場合、この計画をベースラインに含めることには、測定可能なパフォーマンス上の利点があります。

各候補計画の計画および実行時間を収集することに加えて、last_verified ビューの apg_plan_mgmt.dba_plans 列が current_timestamp で更新されます。last_verified タイムスタンプを使用して、最近パフォーマンスが検証された計画でこの関数を再度実行しないようにすることができます。

apg_plan_mgmt.get_explain_plan

指定された SQL ステートメントの EXPLAIN ステートメントのテキストを生成します。

構文

```
apg_plan_mgmt.get_explain_plan(
    sql_hash,
    plan_hash,
    [explainOptionList]
)
```

戻り値

指定された SQL ステートメントの実行時統計を返します。簡単な explainOptionList プランを返すには EXPLAIN なしで使用します。

パラメータ

Parameter	説明
sql_hash	計画の管理 SQL ステートメントの sql_hash ID。
plan_hash	管理計画の plan_hash ID。
explainOptionList	カンマ区切りの説明オプション一覧。有効な値には、'analyze' 、'verbose' 、'buffers' 、'hashes'、および 'format json' があります。explainOptionList が NULL または空の文字列 ('') の場合、この関数は統計なしで EXPLAIN ステートメントを生成します。

使用に関する注意事項

explainOptionList については、EXPLAIN ステートメントで使用するのと同じオプションのいずれかを使用できます。Aurora PostgreSQL オプティマイザは、EXPLAIN ステートメントに指定されたオプションのリストを連結します。

apg_plan_mgmt.plan_last_used

指定された計画の last_used の日付を共有メモリから返します。

> **Note**
>
> DB クラスター内のプライマリ DB インスタンスの共有メモリ値は、常に最新です。この値は apg_plan_mgmt.dba_plans ビューの last_used 列に周期的にしかフラッシュされません。

構文

```
apg_plan_mgmt.plan_last_used(
    sql_hash,
    plan_hash
```

```
)
```

戻り値

last_used の日付を返します。

パラメータ

Parameter	説明
sql_hash	計画の管理 SQL ステートメントの sql_hash ID。
plan_hash	管理計画の plan_hash ID。

apg_plan_mgmt.reload

apg_plan_mgmt.dba_plans ビューから計画を共有メモリに再ロードします。

構文

```
apg_plan_mgmt.reload()
```

戻り値

なし。

パラメータ

なし。

使用に関する注意事項

次の状況では reload を呼び出してください。

- 新しい計画がレプリカに伝播されるのを待たずに、読み取り専用レプリカの共有メモリをただちに更新するために使用する。
- 管理計画をインポートした後に使用する。

apg_plan_mgmt.set_plan_enabled

管理計画を有効または無効にします。

構文

```
apg_plan_mgmt.set_plan_enabled(
    sql_hash,
    plan_hash,
    [true | false]
)
```

戻り値

設定が成功した場合は 0 を返し、設定に失敗した場合は -1 を返します。

パラメータ

Parameter	説明
sql_hash	計画の管理 SQL ステートメントの sql_hash ID。
plan_hash	管理計画の plan_hash ID。
enabled	true または false のブール値。 • 値が true の場合、計画は有効になります。 • 値が false の場合、計画は無効になります。

apg_plan_mgmt.set_plan_status

管理計画のステータスを Approved、Unapproved、Rejected、または Preferred に設定します。

構文

```
apg_plan_mgmt.set_plan_status(
    sql_hash,
    plan_hash,
```

```
    status
)
```

戻り値

設定が成功した場合は 0 を返し、設定に失敗した場合は -1 を返します。

パラメータ

Parameter	説明
sql_hash	計画の管理 SQL ステートメントの sql_hash ID。
plan_hash	管理計画の plan_hash ID。
status	次のいずれかの値を持つ文字列: • 'Approved' • 'Unapproved' • 'Rejected' • 'Preferred' 大文字と小文字の使い分けは重要ではありませんが、ステータス値は apg_plan_mgmt.dba_plans ビューで先頭文字が大文字に設定されます。これらの値についての詳細は status の apg_plan_mgmt.dba_plans ビューのリファレンス を参照してください。

apg_plan_mgmt.update_plans_last_used

プランテーブルを共有メモリに格納されている last_used の日付に即座に更新する。

構文

```
apg_plan_mgmt.update_plans_last_used()
```

戻り値

なし。

パラメータ

なし。

使用に関する注意事項

update_plans_last_used を呼び出して dba_plans.last_used 列に対するクエリが最新の情報を使用しているか確認します。last_used の日付が即座に更新されない場合、バックグラウンドプロセスはデフォルトで毎時間に一回、プランテーブルを last_used 日付で更新します。

例えば、特定の sql_hash ステートメントの実行速度が遅くなった場合、パフォーマンスリグレッションスタート以降、そのステートメントにどのプランが実行されたかを判断できます。これを行うには、まず共有メモリ内のデータをディスクにフラッシュして last_used の日付を最新のものにし、その後パフォーマンスリグレッションのある sql_hash ステートメントのすべてのプランにクエリを実行します。クエリでは、last_used の日付がパフォーマンスリグレッションがスタートされた日付と一緒か、それ以降になるようにしてください。クエリは、パフォーマンスリグレッションの原因の可能性があるプランまたは一連のプランを識別します。verbose, hashes に設定された explainOptionList で apg_plan_mgmt.get_explain_plan を使用することができます。また apg_plan_mgmt.evolve_plan_baselines を使用して、より優れたパフォーマンスを得れるかもしれないプランや代行プランを分析することができます。

update_plans_last_used 関数は、DB クラスターのプライマリ DB インスタンスにのみ影響します。

apg_plan_mgmt.validate_plans

オプティマイザがまだ計画を再作成できることを確認してください。オプティマイザは Approved 計画、Unapproved 計画、および Preferred 計画について、計画が有効か無効かを検証します。Rejected 計画は検証されません。オプションで、apg_plan_mgmt.validate_plans 関数を使用して無効な計画を削除または無効にすることができます。

構文

```
apg_plan_mgmt.validate_plans(
    sql_hash,
    plan_hash,
    action)

apg_plan_mgmt.validate_plans(
```

```
    action)
```

戻り値

無効な計画の数です。

パラメータ

Parameter	説明
sql_hash	計画の管理 SQL ステートメントの sql_hash ID。
plan_hash	管理計画の plan_hash ID。同じ sql_hash ID 値のすべての計画を意味するために NULL を使用します。
action	関数が無効な計画に実行するアクションです。有効な文字列値は次のとおりです。大文字と小文字は区別されません。 • 'disable' - 無効な各計画は無効にされます。 • 'delete' - 無効な各計画は削除されます。 • 'update_plan_hash' - 正確に再現できないプランの plan_hash ID を更新します。また、SQL を書き換えてプランを修正することもできます。元の SQL の Approved プランとして、優れたプランを登録できるようになります。 • NULL - 関数は単に無効な計画の数を返します。他のアクションは実行されません。 • '' - 空のストリングは、有効計画と無効計画の両方の数を示すメッセージを生成します。 他の値は空の文字列のように扱われます。

使用に関する注意事項

validate_plans(action) ビュー全体で、すべての管理ステートメントのすべての計画を検証するには、apg_plan_mgmt.dba_plans 形式を使用してください。

validate_plans(sql_hash, plan_hash, action) で指定された管理ステートメントについて、plan_hash の形式を使用して、sql_hash で指定された管理計画を検証します。

validate_plans(sql_hash, NULL, action) で指定した管理ステートメントのすべての管理計画を検証するには、sql_hash の形式を使用してください。

apg_plan_mgmt.dba_plans ビューのリファレンス

apg_plan_mgmt.dba_plans ビューの計画情報の列には以下のものが含まれます。

dba_plans 列	説明
cardinality_error	推定濃度と実際の濃度の間の誤差の程度。濃度とは、計画により処理されるテーブルの行数です。濃度誤差が大きい場合、計画が最適ではない可能性が高くなります。この列は apg_plan_mgmt.evolve_plan_baselines 関数によって入力されます。
compatibility_level	Aurora PostgreSQL オプティマイザの機能レベル。
created_by	計画を作成した認証済みユーザー (session_user)。
enabled	計画が有効か無効かを示すインジケータ。すべての計画はデフォルトで有効になっています。計画を無効にして、オプティマイザが計画を使用しないようにすることができます。この値を変更するには、apg_plan_mgmt.set_plan_enabled 関数を使用します。
environment_variables	計画の取得時点でオプティマイザにより上書きされた、PostgreSQL Grand Unified Configuration (GUC) のパラメータおよび値。
estimated_startup_cost	オプティマイザがテーブルの行を配信する前の、推定オプティマイザ設定コスト。
estimated_total_cost	最終テーブル行を配信するための推定オプティマイザコスト。
execution_time_benefit_ms	計画を有効にすることで速くなる実行時間。この列は apg_plan_mgmt.evolve_plan_baselines 関数によって入力されます。
execution_time_ms	計画が実行される推定時間 (ミリ秒)。この列は apg_plan_mgmt.evolve_plan_baselines 関数によって入力されます。

dba_plans 列	説明
has_side_effects	SQL ステートメントがデータ操作言語 (DML) ステートメント、または VOLATILE 関数を含む SELECT ステートメントであることを示す値。
last_used	この値は、計画の実行時、または計画がクエリオプティマイザの最小コスト計画であるときに、現在の日付に更新されます。この値は共有メモリに保存され、定期的にディスクにフラッシュされます。最新の値を取得するには、apg_plan_mgmt.plan_last_used(sql_hash, plan_hash) の値を読み取る代わりに関数 last_used を呼び出して、共有メモリから日付を読み取ります。詳細については、apg_plan_mgmt.plan_retention_period パラメータを参照してください。
last_validated	計画が apg_plan_mgmt.validate_plans 関数または apg_plan_mgmt.evolve_plan_baselines 関数のいずれかにより再作成可能であることが検証された最新の日時。
last_verified	計画が、apg_plan_mgmt.evolve_plan_baselines 関数で指定されたパラメータに対して最もパフォーマンスの高い計画であることが検証された最新の日時。
origin	apg_plan_mgmt.capture_plan_baselines パラメータを使用して計画が取得された方法。有効な値には次のようなものがあります。 M - 計画は、手動計画取り込みで取得されています。 A - 計画は、自動計画取り込みで取得されています。
param_list	ステートメントに渡されたパラメータ値 (これが準備済みステートメントである場合)。
plan_created	計画が作成された日時。
plan_hash	計画の識別子。plan_hash と sql_hash の組み合わせにより、特定の計画を一意に識別できます。

dba_plans 列	説明
plan_outline	実際の実行計画を再作成するために使用された、データベースに依存しない計画の表現。ツリーの演算子は、EXPLAIN 出力に表示される演算子に対応しています。
planning_time_ms	プランナーを実際に実行する時間 (ミリ秒)。この列は apg_plan_mgmt.evolve_plan_baselines 関数によって入力されます。
queryId	pg_stat_statements エクステンションによって計算されたステートメントのハッシュ。これはオブジェクト識別子 (OID) に依存しています。安定識別子や非データベース依存の識別子ではありません。クエリプランをキャプチャするとき、compute_query_id が off の場合、値は 0 になります。
sql_hash	リテラルを削除して正規化した、SQL ステートメントのテキストのハッシュ値。
sql_text	SQL ステートメントのフルテキスト。

dba_plans 列	説明
status	オプティマイザによる計画の使用方法を決定する、計画のステータス。有効な値には次のようなものがあります。 • Approved - オプティマイザで選択して実行できる使用可能な計画。オプティマイザは、管理ステートメントの一連の承認済み計画 (ベースライン) から最小コスト計画を実行します。計画を承認済みにリセットするには、apg_plan_mgmt.evolve_plan_baselines 関数を使用します。 • Unapproved - 使用可能性が検証されていない取得済み計画。詳細については、「計画パフォーマンスの評価」を参照してください。 • Rejected - オプティマイザが使用しない計画。詳細については、「低速な計画の拒否または無効化」を参照してください。 • Preferred - 管理ステートメントでの使用に推奨されると判断した計画。 オプティマイザの最小コスト計画が承認済みまたは推奨される計画ではない場合は、計画実施のオーバーヘッドを削減できます。そのためには、承認済み計画のサブセットを Preferred にします。オプティマイザの最小コストが Approved 計画ではない場合、Preferred 計画の前に Approved 計画が選択されます。 計画を Preferred にリセットするには、apg_plan_mgmt.set_plan_status 関数を使用します。
stmt_name	PREPARE ステートメント内の SQL ステートメントの名前。名前のない準備済みステートメントの場合、この値は空の文字列になります。名前のないステートメントの場合、この値は NULL になります。

dba_plans 列	説明
total_time_benefit_ms	この計画を有効にすることで速くなる実行時間の合計。この値には、計画時間と実行時間の両方が考慮されます。 この値が負の場合、この計画を有効にすることは推奨されません。この列は apg_plan_mgmt.evolve_plan_baselines 関数によって入力されます。

クエリ計画管理の高度な機能

Aurora PostgreSQL クエリプラン管理 (QPM) の高度な機能に関する情報は、以下に記載されています。

トピック

- レプリカでの Aurora PostgreSQL 実行プランのキャプチャ
- テーブルパーティションのサポート

レプリカでの Aurora PostgreSQL 実行プランのキャプチャ

QPM (クエリプラン管理) を使用すると、Aurora Replicas によって生成されたクエリプランをキャプチャし、Aurora DB クラスターのプライマリ DB インスタンスに保存できます。すべての Aurora Replicas からクエリプランを収集し、プライマリインスタンスの中央永続テーブルで最適なプランのセットを管理できます。その後、必要に応じてこれらのプランを他のレプリカに適用できます。これにより、DB クラスターやエンジンバージョン全体にわたって実行プランの安定性を維持し、クエリのパフォーマンスを向上させることができます。

トピック

- 前提条件
- Aurora Replicas のプランキャプチャの管理
- トラブルシューティング

前提条件

Aurora レプリカで **capture_plan_baselines parameter** を有効にする - Aurora Replicas でプランをキャプチャするには、capture_plan_baselines パラメーターを自動または手動に設定します。詳細については、「apg_plan_mgmt.capture_plan_baselines」を参照してください。

postgres_fdw 拡張機能をインストールする - Aurora Replicas でプランをキャプチャするには、postgres_fdw 外部データラッパー拡張機能をインストールする必要があります。拡張機能をインストールするには、各データベースで次のコマンドを実行します。

```
postgres=> CREATE EXTENSION IF NOT EXISTS postgres_fdw;
```

Aurora Replicas のプランキャプチャの管理

Aurora Replicas のプランキャプチャをオンにする

Aurora Replicas でプランキャプチャを作成または削除するには、rds_superuser 権限が必要です。ユーザーのロールとアクセス許可の詳細については、「PostgreSQL のロールとアクセス許可の理解」を参照してください。

プランをキャプチャするには、以下に示すように、ライター DB インスタンスで関数 apg_plan_mgmt.create_replica_plan_capture を呼び出します。

```
postgres=> CALL
 apg_plan_mgmt.create_replica_plan_capture('cluster_endpoint', 'password');
```

- cluster_endpoint-cluster_endpoint (ライターエンドポイント) は、Aurora Replicas でのプランキャプチャに対してフェイルオーバーのサポートを提供します。
- パスワード - セキュリティを強化するためにパスワードを作成する際は、以下のガイドラインに従うことをお勧めします。
  - 少なくとも 8 文字を含める必要があります。
  - 少なくとも大文字 1 つ、小文字 1 つ、および 数字 1 つを含める必要があります。
  - 少なくとも 1 つの特殊文字 (?、!、#、<、>、*、など) を含める必要です。

> **Note**
>
> クラスターエンドポイント、パスワード、またはポート番号を変更した場合は、クラスターエンドポイントとパスワードを使用して

apg_plan_mgmt.create_replica_plan_capture() を再実行し、プランキャプチャを再初期化する必要があります。そうでない場合、Aurora Replicas からのプランのキャプチャは失敗します。

Aurora Replicas のプランキャプチャをオフにする

Aurora Replicas の capture_plan_baselines パラメータをオフにするには、パラメータグループの値を off に設定します。

Aurora Replicas のプランキャプチャを削除する

Aurora Replicas のプランキャプチャは完全に削除できますが、削除する前に確認してください。プランキャプチャを削除するには、以下のように apg_plan_mgmt.remove_replica_plan_capture を呼び出します。

```
postgres=> CALL apg_plan_mgmt.remove_replica_plan_capture();
```

クラスターエンドポイントとパスワードを使用して Aurora Replicas のプランキャプチャをオンにするには、apg_plan_mgmt.create_replica_plan_capture() を再度呼び出す必要があります。

トラブルシューティング

以下に、プランが想定どおりに Aurora Replicas にキャプチャされない場合のトラブルシューティングのヒントと回避策を示します。

- パラメータ設定 - プランキャプチャを有効にするために capture_plan_baselines パラメータが適切な値に設定されているかどうかを確認してください。
- **postgres_fdw** 拡張機能がインストールされている - 次のクエリを使用して postgres_fdw がインストールされているかどうかを確認してください。

  ```
  postgres=> SELECT * FROM pg_extension WHERE extname = 'postgres_fdw'
  ```

- create_replica_plan_capture() が呼び出されている - 以下のコマンドを使用して、ユーザーマッピングが終了しているかどうかを確認します。それ以外の場合は、create_replica_plan_capture() を呼び出して機能を初期化します。

  ```
  postgres=> SELECT * FROM pg_foreign_server WHERE srvname =
   'apg_plan_mgmt_writer_foreign_server';
  ```

- クラスターエンドポイントとポート番号 - クラスターエンドポイントとポート番号が適切かどうかを確認してください。これらの値が間違っていても、エラーメッセージは表示されません。

  次のコマンドを使用して、エンドポイントが create() で使用されているかどうかを確認し、そのエンドポイントがどのデータベースにあるかを確認します。

```
postgres=> SELECT srvoptions FROM pg_foreign_server WHERE srvname =
 'apg_plan_mgmt_writer_foreign_server';
```

- reload() - 削除機能を有効にするには、Aurora Replicas で apg_plan_mgmt.delete_plan() を呼び出した後に apg_plan_mgmt.reload() を呼び出す必要があります。これにより、変更が正常に実装されたことが保証されます。
- パスワード - 記載されているガイドラインに従って create_replica_plan_capture() にパスワードを入力する必要があります。入力しないと、エラーメッセージが表示されます。詳細については、「Aurora Replicas のプランキャプチャの管理」を参照してください。要件に合った別のパスワードを使用してください。
- クロスリージョン接続 - Aurora Replicas でのプランキャプチャは Aurora グローバルデータベースでもサポートされており、ライターインスタンスと Aurora Replicas は異なるリージョンに配置できます。ライターインスタンスとクロスリージョンレプリカは VPC ピアリングを使用して通信できる必要があります。詳細については、「VPC ピアリング接続」を参照してください。クロスリージョンフェイルオーバーが発生した場合、エンドポイントを新しいプライマリ DB クラスターエンドポイントに再設定する必要があります。

テーブルパーティションのサポート

Aurora PostgreSQL クエリプラン管理 (QPM) は、以下のバージョンで宣言的テーブルパーティショニングをサポートしています。

- 15.3 以降の 15 バージョン
- 14.8 以降の 14 バージョン
- 13.11 以降の 13 バージョン

詳細については、「テーブルのパーティション」を参照してください。

トピック

- テーブルパーティションの設定
- テーブルパーティションのプランのキャプチャ

- テーブルパーティションプランの実施
- 命名規則

テーブルパーティションの設定

Aurora PostgreSQL QPM でテーブルパーティションをセットアップするには、以下の手順を実行してください。

1. DB クラスターパラメータグループで apg_plan_mgmt.plan_hash_version を 3 以上に設定します。
2. クエリプラン管理を使用し、apg_plan_mgmt.dba_plans ビューにエントリがあるデータベースに移動します。
3. apg_plan_mgmt.validate_plans('update_plan_hash') を呼び出して、プランテーブルの plan_hash 値を更新します。
4. クエリプラン管理が有効になっていて、apg_plan_mgmt.dba_plans ビューにエントリがあるすべてのデータベースについて、手順 2～3 を繰り返します。

これらのパラメータの詳細については、「Aurora PostgreSQL クエリ計画管理のパラメータリファレンス」を参照してください。

テーブルパーティションのプランのキャプチャ

QPM では、さまざまなプランが plan_hash 値によって区別されます。plan_hash がどのように変化するかを理解するには、まず、同様の種類のプランを理解する必要があります。

Append ノードレベルで蓄積されるアクセス方法、数字を取り除いたインデックス名、および数字を取り除いたパーティション名の組み合わせが同じであれば、プランは同じと見なされます。プランでアクセスされる特定のパーティションは重要ではありません。次の例では、テーブル tbl_a は 4 つのパーティションで作成されます。

```
postgres=>create table tbl_a(i int, j int, k int, l int, m int) partition by range(i);
CREATE TABLE
postgres=>create table tbl_a1 partition of tbl_a for values from (0) to (1000);
CREATE TABLE
postgres=>create table tbl_a2 partition of tbl_a for values from (1001) to (2000);
CREATE TABLE
postgres=>create table tbl_a3 partition of tbl_a for values from (2001) to (3000);
CREATE TABLE
```

```
postgres=>create table tbl_a4 partition of tbl_a for values from (3001) to (4000);
CREATE TABLE
postgres=>create index t_i on tbl_a using btree (i);
CREATE INDEX
postgres=>create index t_j on tbl_a using btree (j);
CREATE INDEX
postgres=>create index t_k on tbl_a using btree (k);
CREATE INDEX
```

クエリが検索するパーティションの数に関係なく、単一のスキャン方法を使用して tbl_a をスキャンするため、次のプランは同じと見なされます。

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 999 and j < 9910 and k > 50;

                        QUERY PLAN
-----------------------------------------------------------------------
Seq Scan on tbl_a1 tbl_a
    Filter: ((i >= 990) AND (i <= 999) AND (j < 9910) AND (k > 50))
SQL Hash: 1553185667, Plan Hash: -694232056
(3 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1100 and j < 9910 and k > 50;

                        QUERY PLAN
-----------------------------------------------------------------------
Append
    ->  Seq Scan on tbl_a1 tbl_a_1
            Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
    ->  Seq Scan on tbl_a2 tbl_a_2
            Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
    SQL Hash: 1553185667, Plan Hash: -694232056
    (6 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                QUERY PLAN
-----------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
```

```
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a3 tbl_a_3
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -694232056
(8 rows)
```

次の 3 つのプランも同じとみなされます。親レベルで、アクセス方法、数字を取り除いたインデックス名、および数字を取り除いたパーティション名は、SeqScan tbl_a、IndexScan (i_idx) tbl_a であるためです。

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a2_i_idx on tbl_a2 tbl_a_2
         Index Cond: ((i >= 990) AND (i <= 1100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(7 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(10 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 3100 and j < 9910 and k > 50;

                                QUERY PLAN
------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a3 tbl_a_3
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a4_i_idx on tbl_a4 tbl_a_4
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(11 rows)
```

子パーティション内の出現順序や出現回数が異なっていても、アクセス方法、数字を取り除いたインデックス名、および数字を取り除いたパーティション名は、上記の各プランの親レベルで同じです。

ただし、次の条件のいずれかが満たされる場合、プランは異なると見なされます。

- プランで追加のアクセス方法が使用される。

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between
 990 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Bitmap Heap Scan on tbl_a3 tbl_a_3
         Recheck Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a3_i_idx
               Index Cond: ((i >= 990) AND (i <= 2100))
 SQL Hash: 1553185667, Plan Hash: 1134525070
(11 rows)
```

- プラン内のどのアクセス方法もそれ以上使用されない。

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between
 990 and 1100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -694232056
(6 rows)
```

- インデックスメソッドに関連付けられたインデックスが変更された。

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between
 990 and 1100 and j < 9910 and k > 50;

                              QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a2_j_idx on tbl_a2 tbl_a_2
         Index Cond: (j < 9910)
         Filter: ((i >= 990) AND (i <= 1100) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993343726
(7 rows)
```

テーブルパーティションプランの実施

パーティションテーブルについて承認されたプランは、位置対応で実施されます。プランはパーティションに固有ではなく、元のクエリで参照されているプラン以外のパーティションにも適用できます。プランには、最初に承認されたアウトラインとは異なる数のパーティションにアクセスするクエリを強制する機能もあります。

例えば、承認されたアウトラインが次のプランに関するものであるとします。

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;
```

```
                                QUERY PLAN
---------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(10 rows)
```

この場合、このプランは、2、4、またはそれ以上のパーティションを参照する SQL クエリにも適用できます。これらのシナリオで、2 パーティションと 4 パーティションのアクセスについて考えられるプランは以下のとおりです。

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1100 and j < 9910 and k > 50;

                                QUERY PLAN
---------------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 1100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1553185667, Plan Hash: -993736942, Minimum Cost Plan Hash: -1873216041
(8 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 3100 and j < 9910 and k > 50;

                                QUERY PLAN
---------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 3100))
```

```
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a4 tbl_a_4
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1553185667, Plan Hash: -993736942, Minimum Cost Plan Hash: -1873216041
(12 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 3100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a4_i_idx on tbl_a4 tbl_a_4
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1553185667, Plan Hash: -993736942, Minimum Cost Plan Hash: -1873216041
(14 rows)
```

パーティションごとにアクセス方法が異なる別の承認済みプランを考えてみましょう。

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 2100))
```

```
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Bitmap Heap Scan on tbl_a3 tbl_a_3
         Recheck Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a3_i_idx
               Index Cond: ((i >= 990) AND (i <= 2100))
 SQL Hash: 1553185667, Plan Hash: 2032136998
(12 rows)
```

この場合、2 つのパーティションからデータを読み込むプランは適用されません。承認されたプランの (アクセス方法、インデックス名) の組み合わせがすべて使用可能でない限り、プランを実施することはできません。例えば、以下のプランはプランのハッシュが異なり、承認されたプランはこのような場合には適用できません。

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1900 and j < 9910 and k > 50;

                              QUERY PLAN
-----------------------------------------------------------------------
 Append
   ->  Bitmap Heap Scan on tbl_a1 tbl_a_1
         Recheck Cond: ((i >= 990) AND (i <= 1900))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a1_i_idx
               Index Cond: ((i >= 990) AND (i <= 1900))
   ->  Bitmap Heap Scan on tbl_a2 tbl_a_2
         Recheck Cond: ((i >= 990) AND (i <= 1900))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a2_i_idx
               Index Cond: ((i >= 990) AND (i <= 1900))
  Note: This is not an Approved plan.  No usable Approved plan was found.
  SQL Hash: 1553185667, Plan Hash: -568647260
(13 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1900 and j < 9910 and k > 50;

                              QUERY PLAN
-----------------------------------------------------------------------
 Append
```

```
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 1900))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 1900) AND (j < 9910) AND (k > 50))
 Note: This is not an Approved plan.  No usable Approved plan was found.
 SQL Hash: 1553185667, Plan Hash: -496793743
(8 rows)
```

命名規則

QPM が宣言的パーティションテーブルでプランを適用するには、親テーブル、テーブルパーティション、インデックスに固有の命名規則に従う必要があります。

- 親テーブル名 - これらの名前は、数字だけでなく、アルファベットまたは特殊文字が異なっていなければなりません。例えば、tA、tB、tC は個別の親テーブルには使用できる名前ですが、t1、t2、t3 はそうではありません。
- 各パーティションテーブル名 - 同じ親を持つパーティション名は、数字のみが異なっている必要があります。例えば、tA に指定できるパーティション名は tA1、tA2、t1a、t2A、または複数桁の場合もあります。

  その他の違い (文字、特殊文字) があっても、プランの適用は保証されません。
- インデックス名 – パーティションテーブル階層で、すべてのインデックスに一意の名前があることを確認します。つまり、名前の数値以外の部分は異なっている必要があります。例えば、tA_col1_idx1 という名前のインデックスを持つ tA という名前のパーティションテーブルがある場合、tA_col1_idx2 という名前の別のインデックスを使用することはできません。ただし、tA_a_col1_idx2 というインデックスは、数値以外の部分が一意であるため使用できます。このルールは、親テーブルと個々のパーティションテーブルで作成されたインデックスに適用されます。

上記の命名規則に従わないと、承認されたプランの実施に失敗する可能性があります。次の例は、このような実施が失敗したことを示しています。

```
postgres=>create table t1(i int, j int, k int, l int, m int) partition by range(i);
CREATE TABLE
postgres=>create table t1a partition of t1 for values from (0) to (1000);
CREATE TABLE
postgres=>create table t1b partition of t1 for values from (1001) to (2000);
CREATE TABLE
```

```
postgres=>SET apg_plan_mgmt.capture_plan_baselines TO 'manual';
SET
postgres=>explain (hashes true, costs false) select count(*) from t1 where i > 0;

                            QUERY PLAN
------------------------------------------------------------------------------
 Aggregate
   ->  Append
         ->  Seq Scan on t1a t1_1
               Filter: (i > 0)
         ->  Seq Scan on t1b t1_2
               Filter: (i > 0)
 SQL Hash: -1720232281, Plan Hash: -1010664377
(7 rows)
```

```
postgres=>SET apg_plan_mgmt.use_plan_baselines TO 'on';
SET
postgres=>explain (hashes true, costs false) select count(*) from t1 where i > 1000;

                            QUERY PLAN
------------------------------------------------------------------------------
 Aggregate
   ->  Seq Scan on t1b t1
         Filter: (i > 1000)
 Note: This is not an Approved plan. No usable Approved plan was found.
 SQL Hash: -1720232281, Plan Hash: 335531806
(5 rows)
```

2 つのプランは同じように見えますが、子テーブルの名前によって Plan Hash 値は異なります。数値のみでなくアルファベット文字で名前が異なるテーブルでは、プランの実施が失敗します。

エクステンションと外部データラッパーの使用

Aurora PostgreSQL 互換エディション DB クラスターに機能を拡張するには、さまざまな PostgreSQL の拡張機能をインストールして使用できます。例えば、ユースケースが非常に大きなテーブル間で集中的なデータ入力を必要とする場合、pg_partman 拡張機能をインストールしてデータをパーティション分割し、ワークロードを分散できます。

> **Note**
>
> Aurora PostgreSQL 14.5 以降、Aurora PostgreSQL は Trusted Language Extensions for PostgreSQL をサポートしています。この機能は拡張機能 pg_tle として実装され、Aurora PostgreSQL に追加できます。この拡張を使用することで、開発者は安全な環境で独自の PostgreSQL 拡張を作成できるため、セットアップと設定の要件および新しい拡張機能の予備テストの多くが簡素化されます。詳細については、「Trusted Language Extensions for PostgreSQL を使用した操作」を参照してください。

拡張機能をインストールする代わりに、Aurora PostgreSQL DB クラスターのカスタム DB クラスターパラメータグループの shared_preload_libraries リストに特定のモジュールを追加することもできます。通常、デフォルトの DB クラスターパラメータグループでは、pg_stat_statements のみが読み込まれますが、リストに追加できるモジュールは他にもいくつかあります。例えば、PostgreSQL pg_cron エクステンションによるメンテナンスのスケジューリング で説明されているように、pg_cron モジュールを追加することでスケジュール機能を追加できます。別の例として、auto_explain モジュールをロードすることでクエリ実行計画を記録できます。詳細については、AWS ナレッジセンターの「クエリ実行計画のログ記録」をご覧ください。

外部データへのアクセスを可能にするエクステンションは、具体的には外部データラッパー (FDW) と呼ばれます。一例として、oracle_fdw 拡張機能を使用すると、Aurora PostgreSQL DB クラスターが Oracle データベースと連動できるようになります。

また、Aurora PostgreSQL DB インスタンスにインストール可能な拡張機能は、rds.allowed_extensions パラメータにリストアップして、正確に指定することができます。詳細については、「PostgreSQL 拡張機能のインストールを制限する」を参照してください。

Aurora PostgreSQL で使用可能ないくつかの拡張機能、モジュール、FDW の設定および使用方法についての詳細を以下に説明します。簡単にするために、これらはすべて「拡張機能」と呼ばれています。現在利用可能な Aurora PostgreSQL バージョンで使用できる拡張機能の一覧について

は、Aurora PostgreSQL リリースノートの「Aurora PostgreSQL 拡張機能バージョン」を参照してください。

- lo モジュールを使用したラージオブジェクトの管理
- PostGIS 拡張機能を使用した空間データの管理
- pg_partman エクステンションによる PostgreSQL パーティションの管理
- PostgreSQL pg_cron エクステンションによるメンテナンスのスケジューリング
- pgAudit を使用してデータベースのアクティビティを記録する
- pglogical を使用してインスタンス間でデータを同期する
- oracle_fdw 拡張機能による Oracle データベースの操作
- tds_fdw 拡張機能による SQL Server データベースの操作

PostgreSQL での Amazon Aurora 委任拡張機能サポートの使用

PostgreSQL に対して Amazon Aurora 委任拡張機能のサポートを使用すると、拡張機能管理を `rds_superuser` である必要のないユーザーに委任できます。この委任拡張機能のサポートにより、`rds_extension` という新しいロールが作成され、他の拡張機能を管理するには、これをユーザーに割り当てる必要があります。このロールは、拡張機能を作成、更新、削除できます。

Aurora PostgreSQL DB インスタンスにインストール可能な拡張機能は、`rds.allowed_extensions` パラメータにそれらをリストアップして指定することができます。詳細については、「Amazon RDS for PostgreSQL で PostgreSQL 拡張機能を使用する」を参照してください。

`rds.allowed_delegated_extensions` パラメータを使用して、`rds_extension` ロールでユーザーが管理できる拡張機能のリストを制限できます。

委任拡張機能のサポートは、次のバージョンで利用できます。

- すべての上位バージョン
- 15.5 以降の 15 バージョン
- 14.10 以降の 14 バージョン
- 13.13 以降の 13 バージョン
- 12.17 以降の 12 バージョン

トピック

ユーザーに対する委任拡張機能のサポートの有効化

ユーザーに対して委任拡張機能のサポートを有効にするには、以下を実行する必要があります。

1. ユーザーに **rds_extension** ロールを付与する – rds_superuser としてデータベースに接続し、次のコマンドを実行します。

   ```
   Postgres => grant rds_extension to user_name;
   ```

2. 委任されたユーザーが管理できる拡張機能のリストを設定する – rds.allowed_delegated_extensions では、DB クラスターパラメータで rds.allowed_extensions を使用して、使用可能な拡張機能のサブセットを指定できます。これは、次のいずれかのレベルで実行できます。

   - クラスターまたはインスタンスパラメータグループで、AWS Management Console または API を使用します。詳細については、「「パラメータグループを使用する」」を参照してください。
   - データベースレベルで次のコマンドを使用します。

     ```
     alter database database_name set rds.allowed_delegated_extensions =
      'extension_name_1,
                        extension_name_2,...extension_name_n';
     ```

   - ユーザーレベルで次のコマンドを使用します。

     ```
     alter user user_name set rds.allowed_delegated_extensions = 'extension_name_1,
     ```

```
extension_name_2,...extension_name_n';
```

> **Note**
>
> rds.allowed_delegated_extensions 動的パラメータを変更した後にデータベースを再起動する必要はありません。

3. 拡張機能の作成プロセス中に作成されたオブジェクトへのアクセスを委任されたユーザーに許可する – 特定の拡張機能では、rds_extension ロールを持つユーザーがオブジェクトにアクセスする前に、追加のアクセス許可を付与する必要があるオブジェクトが作成されます。rds_superuser は、それらのオブジェクトへのアクセス権を委任されたユーザーに付与する必要があります。オプションの 1 つは、イベントトリガーを使用して、委任されたユーザーにアクセス許可を自動的に付与することです。詳細については、「委任拡張機能のサポートの無効化」のイベントトリガーの例を参照してください。

PostgreSQL での Aurora 委任拡張機能サポートで使用される設定

設定名	説明	デフォルト値	注意	アクセス許可を変更または付与できるユーザー
rds.allowed_delegated_extensions	このパラメータは、rds_extension ロールがデータベースで管理できる拡張機能を制限します。rds.allowed_extensions のサブセットである必要があります。	空の文字列	• デフォルトでは、このパラメータは空の文字列です。つまり、rds_extension を持つユーザーに拡張機能が委任されていません。 • ユーザーにアクセス許可がある場合は、サポートされている拡張機能を追加で	rds_superuser

設定名	説明	デフォルト値	注意	アクセス許可を変更または付与できるユーザー
			きます。これを実行するためには、rds.allowed_delegated_extensions パラメータをカンマで区切った拡張子名の文字列に設定します。このパラメータに拡張機能のリストを追加すると、rds_extension ロールのあるユーザーがインストールできる拡張機能を明示的に特定できます。 • * に設定すると、rds_allowed_extensions にリストされているすべての拡張機能が rds_extension ロールを持つユーザーに委任されます。 このパラメータの設定の詳細については、「ユーザーに対す	

設定名	説明	デフォルト値	注意	アクセス許可を変更または付与できるユーザー
			る委任拡張機能のサポートの有効化」を参照してください。	
rds.allowed_extensions	このパラメータにより、カスタマーは Aurora PostgreSQL DB インスタンスにインストールできる拡張機能を制限できます。詳細については、「PostgreSQL 拡張機能のインストールの制限」を参照してください。	"*"	デフォルトでは、このパラメータは「*」に設定されています。つまり、RDS for PostgreSQL および Aurora PostgreSQL でサポートされているすべての拡張機能は、必要な権限を持つユーザーが作成できます。 空の場合、Aurora PostgreSQL DB インスタンスに拡張機能をインストールできないことを意味します。	管理者

設定名	説明	デフォルト値	注意	アクセス許可を変更または付与できるユーザー
rds-delegated_nsionow_dr ascac	このパラメータは、rds_extension を持つユーザーがカスケードオプションを使用して拡張機能を削除する機能を制御します。	off	デフォルトで、rds-delegated_extensionallow_drop_cascade は off に設定されています。つまり、rds_extension を持つユーザーは、カスケードオプションを使用して拡張機能を削除することはできません。 その機能を許可するには、rds.delegated_extensionallow_drop_cascade パラメータを on に設定する必要があります。	rds_superuser

委任拡張機能のサポートの無効化

部分的にオフにする

委任されたユーザーは、新しい拡張機能を作成することはできませんが、既存の拡張機能を更新することはできます。

- DB クラスターパラメータグループでデフォルト値に rds.allowed_delegated_extensions をリセットします。
- データベースレベルで次のコマンドを使用します。

```
alter database database_name reset rds.allowed_delegated_extensions;
```

- ユーザーレベルで次のコマンドを使用します。

```
alter user user_name reset rds.allowed_delegated_extensions;
```

すべてオフにする

ユーザーから rds_extension ロールを取り消すと、ユーザーは標準のアクセス許可に戻ります。ユーザーは拡張機能を作成、更新、削除できなくなります。

```
postgres => revoke rds_extension from user_name;
```

イベントトリガーの例

rds_extension を持つ委任ユーザーに、拡張機能の作成によって作成されたオブジェクトに対するアクセス許可の設定を必要とする拡張機能の使用を許可する場合は、次のイベントトリガーの例をカスタマイズし、委任されたユーザーに完全な機能へのアクセスを許可する拡張機能のみを追加できます。このイベントトリガーは template1 (デフォルトのテンプレート) で作成できるため、template1 から作成されたすべてのデータベースにそのイベントトリガーがあります。委任されたユーザーが拡張機能をインストールすると、このトリガーは拡張機能によって作成されたオブジェクトの所有権を自動的に付与します。

```
CREATE OR REPLACE FUNCTION create_ext()

  RETURNS event_trigger AS $$

DECLARE

  schemaname TEXT;
  databaseowner TEXT;

  r RECORD;

BEGIN
```

```
 IF tg_tag = 'CREATE EXTENSION' and current_user != 'rds_superuser' THEN
   RAISE NOTICE 'SECURITY INVOKER';
   RAISE NOTICE 'user: %', current_user;
   FOR r IN SELECT * FROM pg_event_trigger_ddl_commands()
   LOOP
       CONTINUE WHEN r.command_tag != 'CREATE EXTENSION' OR r.object_type !=
'extension';

       schemaname = (
           SELECT n.nspname
           FROM pg_catalog.pg_extension AS e
           INNER JOIN pg_catalog.pg_namespace AS n
           ON e.extnamespace = n.oid
           WHERE e.oid = r.objid
       );

       databaseowner = (
           SELECT pg_catalog.pg_get_userbyid(d.datdba)
           FROM pg_catalog.pg_database d
           WHERE d.datname = current_database()
       );
       RAISE NOTICE 'Record for event trigger %, objid: %,tag: %, current_user: %,
schema: %, database_owenr: %', r.object_identity, r.objid, tg_tag, current_user,
schemaname, databaseowner;
       IF r.object_identity = 'address_standardizer_data_us' THEN
           EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE %I.us_gaz TO
%I WITH GRANT OPTION;', schemaname, databaseowner);
           EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE %I.us_lex TO
%I WITH GRANT OPTION;', schemaname, databaseowner);
           EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE %I.us_rules
TO %I WITH GRANT OPTION;', schemaname, databaseowner);
       ELSIF r.object_identity = 'dict_int' THEN
           EXECUTE format('ALTER TEXT SEARCH DICTIONARY %I.intdict OWNER TO %I;',
schemaname, databaseowner);
       ELSIF r.object_identity = 'pg_partman' THEN
           EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE
%I.part_config TO %I WITH GRANT OPTION;', schemaname, databaseowner);
           EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE
%I.part_config_sub TO %I WITH GRANT OPTION;', schemaname, databaseowner);
           EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE
%I.custom_time_partitions TO %I WITH GRANT OPTION;', schemaname, databaseowner);
       ELSIF r.object_identity = 'postgis_topology' THEN
```

```
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON ALL TABLES IN
 SCHEMA topology TO %I WITH GRANT OPTION;', databaseowner);
            EXECUTE format('GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA topology TO
 %I WITH GRANT OPTION;', databaseowner);
            EXECUTE format('GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA topology TO %I
 WITH GRANT OPTION;', databaseowner);
            EXECUTE format('GRANT USAGE ON SCHEMA topology TO %I WITH GRANT OPTION;',
 databaseowner);
        END IF;
    END LOOP;
  END IF;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE EVENT TRIGGER log_create_ext ON ddl_command_end EXECUTE PROCEDURE create_ext();
```

Amazon Aurora 委任拡張機能サポートの使用のメリット

PostgreSQL に対して Amazon Aurora 委任拡張機能のサポートを使用すると、拡張機能管理を `rds_superuser` ロールを持たないユーザーに対してセキュアに委任できます。この機能には次の利点があります。

- 選択したユーザーに拡張機能管理を簡単に委任できます。
- これには `rds_superuser` ロールは不要です。
- 同じ DB クラスター内の異なるデータベースに対して、異なる拡張機能セットをサポートする機能を提供します。

PostgreSQL での Aurora 委任拡張機能サポートの制限

- 拡張機能の作成プロセス中に作成されたオブジェクトは、拡張機能が正しく機能するために追加の権限が必要になる場合があります。

特定の拡張機能に必要なアクセス許可

次の拡張機能を作成、使用、または更新するには、委任されたユーザーに次の関数、テーブル、スキーマに対する必要な権限が必要です。

所有権またはアクセス許可が必要な拡張機能	機能	テーブル	Schema	テキスト検索ディクショナリ	コメント
addre tanda er_da s		us_gaz、us _lex、us_l ex、l.us_rules			
amch	bt_index_ check、bt_ index_par ent_check				
dict_i				intdict	
pg_p n		custom_ti me_partit ions、part			

所有権またはアクセス許可が必要な拡張機能	機能	テーブル	Schema	テキスト検索ディクショナリ	コメント
		_config、part_config_sub			
pg_statem					
Postg	st_tileenvelope	spatial_ref_sys			
postgaster					
postgopolo		トポロジ、レイヤー	トポロジ		委任されたユーザーはデータベース所有者であること

所有権またはアクセス許可が必要な拡張機能	機能	テーブル	Schema	テキスト検索ディクショナリ	コメント
log_f	create_foreign_table_for_log_file				
rds_t	role_password_encryption_type				
postg iger_ oder		geocode_settings_default, geocode_settings	tiger		
pg_fr acem	pg_freespace				

所有権またはアクセス許可が必要な拡張機能	機能	テーブル	Schema	テキスト検索ディクショナリ	コメント
pg_visibility	pg_visibility				

セキュリティに関する考慮事項

rds_extension ロールを持つユーザーは、接続権限を持つすべてのデータベースの拡張機能を管理できることに注意してください。委任されたユーザーが 1 つのデータベースの拡張機能を管理する場合は、各データベースのパブリックからすべての権限を取り消し、その特定のデータベースの接続権限を委任されたユーザーに明示的に付与することをお勧めします。

ユーザーが複数のデータベースから情報にアクセスできる拡張機能がいくつかあります。これらの拡張機能を rds.allowed_delegated_extensions に追加する前に、rds_extension を付与するユーザーにクロスデータベース機能があることを確認してください。例えば、postgres_fdw と dblink は、同じインスタンスまたはリモートインスタンス上のデータベース間でクエリを実行する機能を提供します。log_fdw は postgres エンジンのログファイルを読み取ります。これらは、イ

ンスタンス内のすべてのデータベース用であり、複数のデータベースからのスロークエリやエラーメッセージが含まれている可能性があります。pg_cron は、DB インスタンスでスケジュールされたバックグラウンドジョブの実行を有効にし、別のデータベースで実行するようにジョブを設定できます。

DROP EXTENSION CASCADE を無効化

rds_extension ロールを持つユーザーがカスケードオプションを使用して拡張機能を削除する機能は、rds.delegated_extension_allow_drop_cascade パラメータによって制御されます。デフォルトで、rds-delegated_extension_allow_drop_cascade は off に設定されています。つまり、rds_extension ロールを持つユーザーは、以下のクエリに示すように、カスケードオプションを使用して拡張機能を削除することはできません。

```
DROP EXTENSION CASCADE;
```

これにより、拡張機能に依存するオブジェクト、およびそれらのオブジェクトに依存するすべてのオブジェクトが自動的に削除されます。カスケードオプションを使用しようとすると、エラーが発生します。

その機能を許可するには、rds.delegated_extension_allow_drop_cascade パラメータを on に設定する必要があります。

rds.delegated_extension_allow_drop_cascade 動的パラメータを変更しても、データベースを再起動する必要はありません。これは、次のいずれかのレベルで実行できます。

- クラスターまたはインスタンスパラメータグループで、AWS Management Console または API を使用します。
- データベースレベルで次のコマンドを使用する。

  ```
  alter database database_name set rds.delegated_extension_allow_drop_cascade = 'on';
  ```

- ユーザーレベルで次のコマンドを使用する。

  ```
  alter role tenant_user set rds.delegated_extension_allow_drop_cascade = 'on';
  ```

委任拡張機能サポートを使用して追加できる拡張機能の例

- rds_tools

```
extension_test_db=> create extension rds_tools;
CREATE EXTENSION
extension_test_db=> SELECT * from rds_tools.role_password_encryption_type() where
 rolname = 'pg_read_server_files';
ERROR: permission denied for function role_password_encryption_type
```

- amcheck

```
extension_test_db=> CREATE TABLE amcheck_test (id int);
CREATE TABLE
extension_test_db=> INSERT INTO amcheck_test VALUES (generate_series(1,100000));
INSERT 0 100000
extension_test_db=> CREATE INDEX amcheck_test_btree_idx ON amcheck_test USING btree
 (id);
CREATE INDEX
extension_test_db=> create extension amcheck;
CREATE EXTENSION
extension_test_db=> SELECT bt_index_check('amcheck_test_btree_idx'::regclass);
ERROR: permission denied for function bt_index_check
extension_test_db=> SELECT bt_index_parent_check('amcheck_test_btree_idx'::regclass);
ERROR: permission denied for function bt_index_parent_check
```

- pg_freespacemap

```
extension_test_db=> create extension pg_freespacemap;
CREATE EXTENSION
extension_test_db=> SELECT * FROM pg_freespace('pg_authid');
ERROR: permission denied for function pg_freespace
extension_test_db=> SELECT * FROM pg_freespace('pg_authid',0);
ERROR: permission denied for function pg_freespace
```

- pg_visibility

```
extension_test_db=> create extension pg_visibility;
CREATE EXTENSION
extension_test_db=> select * from pg_visibility('pg_database'::regclass);
ERROR: permission denied for function pg_visibility
```

- postgres_fdw

```
extension_test_db=> create extension postgres_fdw;
CREATE EXTENSION
```

```
extension_test_db=> create server myserver foreign data wrapper postgres_fdw options
 (host 'foo', dbname 'foodb', port '5432');
ERROR: permission denied for foreign-data wrapper postgres_fdw
```

lo モジュールを使用したラージオブジェクトの管理

lo モジュール (拡張機能) は、JDBC または ODBC ドライバを介して PostgreSQL データベースを操作するデータベースユーザーおよび開発者向けです。JDBC と ODBC はどちらも、ラージオブジェクトの参照が変更されたときに、データベースがラージオブジェクトの削除を処理することを想定しています。ただし、PostgreSQL はそのように動作しません。PostgreSQL では、オブジェクトの参照が変更されたときにオブジェクトを削除する必要があるとは想定していません。その結果、オブジェクトはディスク上に残り、参照されません。lo 拡張機能には、必要に応じてオブジェクトを削除するための参照変更時にトリガーするために使用する関数が含まれています。

> **Tip**
>
> データベースが lo 拡張機能の恩恵を受けるかどうかを判断するには、vacuumlo ユーティリティを使用して、孤立したラージオブジェクトをチェックします。アクションを実行せずに孤立したラージオブジェクトのカウントを取得するには、-n オプション (no-op) を使ってユーティリティを実行します。この方法については、下記の「vacuumlo utility」を参照してください。

Lo モジュールは Aurora PostgreSQL 13.7、12.11、11.16、10.21 以降のマイナーバージョンで利用できます。

モジュール (拡張機能) をインストールするには、rds_superuser 権限が必要です。lo 拡張機能をインストールすると、データベースに次のものが追加されます。

- lo — これは、バイナリラージオブジェクト (BLOB) やその他のラージオブジェクトに使用できるラージオブジェクト (lo) データ型です。lo データ型は、oid データ型のドメインです。つまり、オプションの制約を持つオブジェクト識別子です。詳細については、PostgreSQL ドキュメントの「オブジェクト識別子」を参照してください。簡潔に言うと、lo データ型を使用して、ラージオブジェクト参照を保持するデータベース列を他のオブジェクト識別子 (OID) と区別できます。
- lo_manage — これは、ラージオブジェクト参照を含むテーブル列のトリガーで使用できる関数です。ラージオブジェクトを参照する値を削除または変更すると、トリガーによってリファレンスか

らオブジェクト (lo_unlink) のリンクが解除されます。列がラージオブジェクトへの唯一のデータベース参照である場合にのみ、列でトリガーを使用します。

ラージオブジェクトモジュールの詳細については、PostgreSQL ドキュメントの「lo」を参照してください。

lo 拡張機能のインストール

lo 拡張機能をインストールする前に、rds_superuser 権限があることを確認してください。

lo 拡張機能をインストールするには

1. psql を使用して、Aurora PostgreSQL DB クラスターのプライマリ DB インスタンスに接続します。

   ```
   psql --host=your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password
   ```

   プロンプトが表示されたら、パスワードを入力します。psql クライアントが接続し、プロンプトとしてデフォルトの管理用接続データベースである postgres=> を表示します。

2. 次のように 拡張機能をインストールします。

   ```
   postgres=> CREATE EXTENSION lo;
   CREATE EXTENSION
   ```

lo データ型を使用して、テーブルの列を定義できるようになりました。例えば、ラスターイメージデータが含まれるテーブルを作成できます (images)。次の例に示すように、列 raster の lo データ型を使って、テーブルを作成できます。

```
postgres=> CREATE TABLE images (image_name text, raster lo);
```

lo_manage トリガー関数を使用してオブジェクトを削除する

lo または他のラージオブジェクト列の lo_manage 関数を使って、lo が更新または削除されたときにクリーンアップ (および孤立したオブジェクトを防ぐ) ことができます。

ラージオブジェクトを参照する列にトリガーを設定するには

- 次のいずれかを実行します。
  - 引数に列名を使用して、ラージオブジェクトへの一意の参照を含むように、各列に BEFORE UPDATE トリガーまたは BEFORE DELETE トリガーを作成します。

```
postgres=> CREATE TRIGGER t_raster BEFORE UPDATE OR DELETE ON images
    FOR EACH ROW EXECUTE FUNCTION lo_manage(raster);
```

  - トリガーは、列が更新されている場合にのみ適用します。

```
postgres=> CREATE TRIGGER t_raster BEFORE UPDATE OF images
    FOR EACH ROW EXECUTE FUNCTION lo_manage(raster);
```

lo_manage トリガー関数は、トリガーの定義方法に応じて、列データの挿入または削除のコンテキストでのみ機能します。データベースで DROP または TRUNCATE 式を実行しても効果はありません。つまり、孤立したオブジェクトを作成しないようにするには、削除する前にテーブルからオブジェクト列を削除する必要があります。

例えば、images テーブルを含むデータベースを削除するとします。列は、次のように削除します。

```
postgres=> DELETE FROM images COLUMN raster
```

lo_manage 関数がその列で削除を処理するために定義されていると仮定すると、テーブルを安全に削除できます。

vacuumlo ユーティリティの使用

vacuumlo ユーティリティは、孤立したラージオブジェクトを識別し、データベースから削除できます。このユーティリティは PostgreSQL 9.1.24 以降で使用可能です。データベースユーザーがラージオブジェクトを日常的に操作する場合は、vacuumlo をときどき実行して、孤立したラージオブジェクトをクリーンアップすることをお勧めします。

lo 拡張機能をインストールする前に、vacuumlo を使用して Aurora PostgreSQL DB クラスターにメリットがあるかどうかを評価できます。これを行うには、-n オプション (no-op) を使用して vacuumlo を実行し、以下に示すように、削除される内容を表示します。

```
$ vacuumlo -v -n -h your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com -
p 5433 -U postgres docs-lab-spatial-db
```

```
Password:*****
Connected to database "docs-lab-spatial-db"
Test run: no large objects will be removed!
Would remove 0 large objects from database "docs-lab-spatial-db".
```

出力が示すように、孤立したラージオブジェクトは、この特定のデータベースでは問題になりません。

pg_upgrade ユーティリティの詳細については、PostgreSQL ドキュメントの「vacuumlo」を参照してください。

PostGIS 拡張機能を使用した空間データの管理

PostGIS は PostgreSQL の拡張機能であり、空間情報の保存と管理に使用します。PostGIS の詳細については、「Postgis.net」を参照してください。

バージョン 10.5 以降の PostgreSQL では、PostGIS がマップボックスのベクトルタイルデータを操作するために使用する libprotobuf 1.3.0 ライブラリがサポートされています。

PostGIS 拡張機能のセットアップには、`rds_superuser` 権限が必要です。PostGIS 拡張機能と空間データを管理するためのユーザー (ロール) を作成することをお勧めします。PostGIS 拡張機能とその関連コンポーネントは PostgreSQL に数千もの関数を追加します。ユースケースに適している場合は、PostGIS エクステンションを独自のスキーマで作成することを検討してください。次の例は、拡張機能を独自のデータベースにインストールする方法を示していますが、これは必須ではありません。

トピック

- ステップ 1: PostGIS 拡張機能を管理するユーザー (ロール) を作成する
- ステップ 2: PostGIS エクステンションを読み込む
- ステップ 3: ロールに拡張機能の所有権を移転します。
- ステップ 4: PostGIS オブジェクトの所有権を移転する
- ステップ 5: エクステンションをテストする
- ステップ 6: PostGIS 拡張機能を更新する
- PostGIS 拡張バージョン
- PostGIS 2 から PostGIS 3 へのアップグレード

ステップ 1: PostGIS 拡張機能を管理するユーザー (ロール) を作成する

まず、rds_superuser 権限があるユーザーとして RDS for PostgreSQL DB インスタンスに接続します。インスタンスの設定時にデフォルトの名前を保持している場合は、次のように postgres として接続します。

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --username=postgres
 --password
```

PostGIS 拡張機能を管理する別のロール (ユーザー) を作成します。

```
postgres=>  CREATE ROLE gis_admin LOGIN PASSWORD 'change_me';
CREATE ROLE
```

このロールに rds_superuser 権限を付与して、ロールが拡張機能をインストールできるようにします。

```
postgres=> GRANT rds_superuser TO gis_admin;
GRANT
```

PostGIS アーティファクトに使用するデータベースを作成します。この手順は省略可能です。または、ユーザーデータベースに PostGIS 拡張機能用のスキーマを作成することもできますが、これも必須ではありません。

```
postgres=> CREATE DATABASE lab_gis;
CREATE DATABASE
```

gis_admin に lab_gis データベース上のすべての特権を付与します。

```
postgres=> GRANT ALL PRIVILEGES ON DATABASE lab_gis TO gis_admin;
GRANT
```

セッションを終了し、RDS for PostgreSQL DB インスタンスに gis_admin として再接続します。

```
postgres=> psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=gis_admin --password --dbname=lab_gis
Password for user gis_admin:...
lab_gis=>
```

次の手順の説明に従って、拡張機能のセットアップを続けます。

ステップ 2: PostGIS エクステンションを読み込む

PostGIS 拡張機能には複数の関連する拡張機能があり、それらが連携することで地理空間機能を提供しています。ユースケースによっては、このステップで作成した拡張機能の一部が必要ない場合があります。

CREATE EXTENSION ステートメントを使用して PostGIS エクステンションをロードします。

```
CREATE EXTENSION postgis;
CREATE EXTENSION
CREATE EXTENSION postgis_raster;
CREATE EXTENSION
CREATE EXTENSION fuzzystrmatch;
CREATE EXTENSION
CREATE EXTENSION postgis_tiger_geocoder;
CREATE EXTENSION
CREATE EXTENSION postgis_topology;
CREATE EXTENSION
CREATE EXTENSION address_standardizer_data_us;
CREATE EXTENSION
```

次の例に示されている SQL クエリを実行すると、拡張子とその所有者がリストアップされ、結果を確認することができます。

```
SELECT n.nspname AS "Name",
  pg_catalog.pg_get_userbyid(n.nspowner) AS "Owner"
  FROM pg_catalog.pg_namespace n
  WHERE n.nspname !~ '^pg_' AND n.nspname <> 'information_schema'
  ORDER BY 1;
List of schemas
     Name     |   Owner
--------------+-----------
 public       | postgres
 tiger        | rdsadmin
 tiger_data   | rdsadmin
 topology     | rdsadmin
(4 rows)
```

ステップ 3: ロールに拡張機能の所有権を移転します。

ALTER SCHEMA ステートメント使用して、gis_admin ロールにスキーマの所有権を移転します。

```
ALTER SCHEMA tiger OWNER TO gis_admin;
ALTER SCHEMA
ALTER SCHEMA tiger_data OWNER TO gis_admin;
ALTER SCHEMA
ALTER SCHEMA topology OWNER TO gis_admin;
ALTER SCHEMA
```

次の SQL クエリを実行して、所有権の変更を確認できます。または、psql コマンドラインの \dn メタコマンドを使用します。

```
SELECT n.nspname AS "Name",
  pg_catalog.pg_get_userbyid(n.nspowner) AS "Owner"
  FROM pg_catalog.pg_namespace n
  WHERE n.nspname !~ '^pg_' AND n.nspname <> 'information_schema'
  ORDER BY 1;

       List of schemas
     Name     |     Owner
--------------+---------------
 public       | postgres
 tiger        | gis_admin
 tiger_data   | gis_admin
 topology     | gis_admin
(4 rows)
```

ステップ 4: PostGIS オブジェクトの所有権を移転する

次の関数を使用して、gis_admin ロールに PostGIS オブジェクトの所有権を移転します。psql プロンプトから次のステートメントを実行して関数を作成します。

```
CREATE FUNCTION exec(text) returns text language plpgsql volatile AS $f$ BEGIN EXECUTE
 $1; RETURN $1; END; $f$;
CREATE FUNCTION
```

続いて、次のクエリを実行して exec 関数を実行すると、ステートメントが実行されてアクセス許可が変更されます。

```
SELECT exec('ALTER TABLE ' || quote_ident(s.nspname) || '.' || quote_ident(s.relname)
 || ' OWNER TO gis_admin;')
  FROM (
    SELECT nspname, relname
```

```
    FROM pg_class c JOIN pg_namespace n ON (c.relnamespace = n.oid)
    WHERE nspname in ('tiger','topology') AND
    relkind IN ('r','S','v') ORDER BY relkind = 'S')
s;
```

ステップ 5: エクステンションをテストする

スキーマ名の指定を不要とするには、次のコマンドを使用して検索パスに tiger スキーマを追加します。

```
SET search_path=public,tiger;
SET
```

次の SELECT ステートメントを使用して、tiger スキーマをテストします。

```
SELECT address, streetname, streettypeabbrev, zip
 FROM normalize_address('1 Devonshire Place, Boston, MA 02109') AS na;
address | streetname | streettypeabbrev |  zip
---------+------------+------------------+-------
       1 | Devonshire | Pl               | 02109
(1 row)
```

この拡張機能の詳細については、PostGIS ドキュメントの「Tiger Geocoder」を参照してください。

次の topology ステートメントを使用して SELECT スキーマへのアクセスをテストします。これにより、createtopology 関数を呼び出して、指定された空間参照識別子 (26986) とデフォルトの許容誤差 (0.5) を持つ新しいトポロジーオブジェクト (my_new_topo) を登録します。詳細については、PostGIS ドキュメントの「CreateTopology」を参照してください。

```
SELECT topology.createtopology('my_new_topo',26986,0.5);
 createtopology
----------------
              1
(1 row)
```

ステップ 6: PostGIS 拡張機能を更新する

PostgreSQL の新しいリリースでは、それぞれのリリースと互換性のある 1 つまたは複数のバージョンの PostGIS 拡張機能をサポートしています。PostgreSQL エンジンを新しいバージョンにアップグレードしても、PostGIS 拡張機能は自動的にアップグレードされません。PostgreSQL エンジン

をアップグレードする前に、通常 PostGIS を現在の PostgreSQL バージョンで使用可能な最新バージョンにアップグレードします。詳細については、「PostGIS 拡張バージョン」を参照してください。

PostgreSQL エンジンのアップグレード後、PostGIS 拡張機能を再度アップグレードして、新しくアップグレードした PostgreSQL エンジンバージョンでサポートされているバージョンにアップグレードします。PostgreSQL のアップグレードの詳細については、「本番稼働用の DB クラスターの新しいメジャーバージョンへのアップグレードをテストする 」を参照してください。

Aurora PostgreSQL DB クラスター では、利用可能な PostGIS 拡張機能のバージョンアップを常時確認できます。そうするには、以下のコマンドを実行します。この関数は、PostGIS 2.5.0 以降のバージョンで使用できます。

```
SELECT postGIS_extensions_upgrade();
```

アプリケーションが最新で PostGIS バージョンがサポートされていない場合でも、次のように、メジャーバージョンで使用できる古いバージョンの PostGIS をインストールできます。

```
CREATE EXTENSION postgis VERSION "2.5.5";
```

古いバージョンから特定の PostGIS バージョンにアップグレードする場合は、次のコマンドも使用できます。

```
ALTER EXTENSION postgis UPDATE TO "2.5.5";
```

アップグレード前のバージョンによっては、この関数をもう一度実行する必要があります。初期に関数を実行した結果によって、追加のアップグレード関数が必要かどうかが決まります。例えば、PostGIS 2 から PostGIS 3 にアップグレードする場合がこれに該当します。(詳しくは、「PostGIS 2 から PostGIS 3 へのアップグレード」を参照してください。)

PostgreSQL エンジンのメジャーアップグレードの準備のためにこの拡張機能をアップグレードした場合は、他の準備作業を継続できます。詳細については、「本番稼働用の DB クラスターの新しいメジャーバージョンへのアップグレードをテストする」を参照してください。

PostGIS 拡張バージョン

「Aurora PostgreSQL リリースノート」の「Aurora PostgreSQL 互換エディションの拡張機能バージョン」に記載されている PostGIS など、すべての拡張機能バージョンをインストールすることを

お勧めします。「リリースで利用可能なバージョンのリストを取得するには、次のコマンドを使用します。

```
SELECT * FROM pg_available_extension_versions WHERE name='postgis';
```

バージョン情報は、Aurora PostgreSQL リリースノートの次のセクションで確認できます。

- Aurora PostgreSQL 14 の拡張機能バージョン
- Aurora PostgreSQL - 互換エディション 13 の拡張機能バージョン
- Aurora PostgreSQL - 互換エディション 12 の拡張機能バージョン
- Aurora PostgreSQL - 互換エディション 11 の拡張機能バージョン
- Aurora PostgreSQL - 互換エディション 10 の拡張機能バージョン
- Aurora PostgreSQL - 互換エディション 9.6 の拡張機能バージョン

PostGIS 2 から PostGIS 3 へのアップグレード

バージョン 3.0 以降、PostGIS ラスター機能は別の postgis_raster という拡張機能になりました。この拡張機能には、独自のインストールとアップグレードパスがあります。これにより、ラスター画像処理に必要な多くの関数、データ型などのアーティファクトがコア postgis 拡張機能から削除されます。つまり、ユースケースにラスター処理が必要ない場合は、postgis_raster 拡張機能をインストールする必要はありません。

次のアップグレード例では、最初のアップグレードコマンドは、ラスター機能を postgis_raster 拡張機能に展開します。次に、postgres_raster を新しいバージョンにアップグレードするには 2 つ目のアップグレードコマンドが必要です。

PostGIS 2 から PostGIS 3 にアップグレードするには

1. お使いの Aurora PostgreSQL DB クラスターの PostgreSQL バージョンで利用可能な PostGIS のデフォルトバージョンを確認します。確認するために、以下のクエリを実行します。

```
SELECT * FROM pg_available_extensions
    WHERE default_version > installed_version;
  name   | default_version | installed_version |                                comment
---------+-----------------+-------------------
+-------------------------------------------------------------
 postgis | 3.1.4           | 2.3.7             | PostGIS geometry and geography
 spatial types and functions
```

```
(1 row)
```

2. Aurora PostgreSQL DB クラスターのライターインスタンス、の各データベースにインストールされている PostGIS のバージョンを確認します。つまり、各ユーザーデータベースを次のようにクエリします。

```
SELECT
    e.extname AS "Name",
    e.extversion AS "Version",
    n.nspname AS "Schema",
    c.description AS "Description"
FROM
    pg_catalog.pg_extension e
    LEFT JOIN pg_catalog.pg_namespace n ON n.oid = e.extnamespace
    LEFT JOIN pg_catalog.pg_description c ON c.objoid = e.oid
    AND c.classoid = 'pg_catalog.pg_extension'::pg_catalog.regclass
WHERE
    e.extname LIKE '%postgis%'
ORDER BY
    1;
  Name   | Version | Schema |                            Description
---------+---------+--------
+-------------------------------------------------------------------------
 postgis | 2.3.7   | public | PostGIS geometry, geography, and raster spatial types
 and functions
(1 row)
```

このようにデフォルトバージョン (PostGIS 3.1.4) とインストールされているバージョン (PostGIS 2.3.7) が一致しない場合は、PostGIS 拡張機能をアップグレードする必要があることになります。

```
ALTER EXTENSION postgis UPDATE;
ALTER EXTENSION
WARNING: unpackaging raster
WARNING: PostGIS Raster functionality has been unpackaged
```

3. 次のクエリを実行して、ラスター機能が独自のパッケージに組み込まれていることを確認します。

```
SELECT
    probin,
    count(*)
```

```
FROM
    pg_proc
WHERE
    probin LIKE '%postgis%'
GROUP BY
    probin;
          probin          | count
--------------------------+-------
 $libdir/rtpostgis-2.3    | 107
 $libdir/postgis-3        | 487
(2 rows)
```

出力を確認すれば、バージョンの間にまだ差があることがわかります。PostGIS 関数はバージョン 3 (postgis-3) で、ラスター関数 (rtpostgis) はバージョン 2 (rtpostgis-2.3) です。アップグレードを完了するには、次のようにアップグレードコマンドを再度実行します。

```
postgres=> SELECT postgis_extensions_upgrade();
```

警告メッセージは無視しても問題ありません。次のクエリを再度実行して、アップグレードが完了していることを確認します。PostGIS と関連するすべての拡張機能に対してアップグレードが必要と表示されていなければ、アップグレードは完了です。

```
SELECT postgis_full_version();
```

4. 次のクエリを使用して、完了したアップグレードプロセスと個別にパッケージ化された拡張機能を確認し、それぞれのバージョンが一致していることを確認します。

```
SELECT
    e.extname AS "Name",
    e.extversion AS "Version",
    n.nspname AS "Schema",
    c.description AS "Description"
FROM
    pg_catalog.pg_extension e
    LEFT JOIN pg_catalog.pg_namespace n ON n.oid = e.extnamespace
    LEFT JOIN pg_catalog.pg_description c ON c.objoid = e.oid
        AND c.classoid = 'pg_catalog.pg_extension'::pg_catalog.regclass
WHERE
    e.extname LIKE '%postgis%'
ORDER BY
    1;
```

```
      Name      | Version | Schema |                                Description
----------------+---------+--------
+-------------------------------------------------------------------------
 postgis        | 3.1.5   | public | PostGIS geometry, geography, and raster
 spatial types and functions
 postgis_raster | 3.1.5   | public | PostGIS raster types and functions
(2 rows)
```

出力には、PostGIS 2 拡張機能が PostGIS 3 にアップグレードされ、postgis と現在は分離された postgis_raster 拡張機能の両方がバージョン 3.1.5 であることが表示されます。

このアップグレード完了後にラスター機能を使用する予定がない場合は、次のように拡張機能を削除できます。

```
DROP EXTENSION postgis_raster;
```

pg_partman エクステンションによる PostgreSQL パーティションの管理

PostgreSQL テーブルパーティションは、データ入力とレポートにおける高性能な処理のためのフレームワークを提供します。大量のデータを非常に速く入力する必要があるデータベースには、パーティションを使用します。またパーティションは、大きなテーブルでより高速のクエリを提供します。パーティションは、必要とする I/O リソースが少ないため、データベースインスタンスに影響を与えずにデータを維持するのに役立ちます。

パーティションを使用すると、データをカスタムサイズのチャンクに分割して処理することができます。例えば、時間単位、日単位、週単位、月単位、四半期単位、年単位、カスタム、またはこれらの組み合わせなどの範囲のパーティションの時系列データを選択できます。時系列データの例では、テーブルを時間単位でパーティション化した場合、各パーティションには 1 時間のデータが含まれます。時系列テーブルを日単位でパーティション化した場合、パーティションには 1 日分のデータが保持されます。パーティションキーは、パーティションのサイズを制御します。

パーティション化されたテーブルで INSERT SQL コマンドまたは UPDATE SQL コマンドを使用すると、データベースエンジンはデータを適切なパーティションにルーティングします。データを格納する PostgreSQL テーブルパーティションは、メインテーブルの子テーブルです。

データベースクエリの読み取り中、PostgreSQL オプティマイザはクエリの WHERE 句を調べ、可能な場合、関連するパーティションだけにデータベーススキャンを行うよう指示します。

バージョン 10 以降、PostgreSQL は宣言的なパーティショニングを使用してテーブルパーティションを実装します。これは、ネイティブ PostgreSQL パーティションとも言います。PostgreSQL バージョン 10 より前は、トリガーを使用してパーティションを実装していました。

PostgreSQL テーブルパーティションは、次の機能を提供します。

- 新しいパーティションの作成がいつでも可能。
- 可変のパーティション範囲。
- データ定義言語 (DDL) ステートメントを使用して、取り外し可能かつ再接続可能なパーティション。

  例えば、デタッチ可能なパーティションは、メインパーティションから履歴データを削除し、履歴データを分析用に保持する場合に便利です。
- 新しいパーティションは、親のデータベーステーブルの以下のプロパティを継承します。
  - インデックス
  - プライマリキー (パーティションキー列を含める必要があります)
  - 外部キー
  - 検査制約
  - 参照
- フルテーブルまたは各特定のパーティションのインデックスの作成。

個々のパーティションのスキーマを変更することはできません。ただし、パーティションに伝播される親テーブル (新しい列の追加など) は変更できます。

トピック

PostgreSQL pg_partman エクステンションの概要

PostgreSQL `pg_partman` エクステンションを使用すると、テーブルパーティションの作成とメンテナンスを自動化できます。一般的な情報については、`pg_partman` ドキュメントの「PG Partition Manager」を参照してください。

> **Note**
>
> pg_partman エクステンションは、Aurora PostgreSQL のエンジンバージョン 12.6 以降でサポートされています。

各パーティションを手動で作成する代わりに、次の設定で pg_partman を設定します。

- パーティション化するテーブル
- パーティションタイプ
- パーティションキー
- パーティションの粒度
- パーティションの事前作成および管理オプション

PostgreSQL のパーティション化されたテーブルの作成後、create_parent 関数を呼び出して、そのテーブルを pg_partman に登録します。これにより、関数に渡すパラメータに基づいて、必要なパーティションを作成します。

pg_partman エクステンションには、設定したスケジュールに基づいて呼び出しを行うことでパーティションを自動的に管理できる run_maintenance_proc 関数も用意されています。必要に応じて適切なパーティションが作成されるようにするには、この関数を定期的に (時間単位など) 実行するようにスケジュールします。また、パーティションが自動的に削除されるようにすることもできます。

pg_partman エクステンションの有効化

パーティションを管理する同じ PostgreSQL DB インスタンス内に複数のデータベースがある場合は、データベースごとに pg_partman エクステンションを有効にします。特定のデータベースで pg_partman エクステンションを有効にするには、パーティションメンテナンススキーマを作成した上で、次のように pg_partman エクステンションを作成します。

```
CREATE SCHEMA partman;
CREATE EXTENSION pg_partman WITH SCHEMA partman;
```

> **Note**
>
> pg_partman エクステンションを作成するには、rds_superuser 権限が必要です。

次のようなエラーが表示された場合は、アカウントに rds_superuser 権限を付与するか、スーパーユーザーアカウントを使用します。

```
ERROR: permission denied to create extension "pg_partman"
HINT: Must be superuser to create this extension.
```

rds_superuser 権限を付与するには、スーパーユーザーアカウントを使用して接続し、以下のコマンドを実行します。

```
GRANT rds_superuser TO user-or-role;
```

pg_partman エクステンションの使用方法を示す例では、次のサンプルのデータベーステーブルとパーティションを使用します。このデータベースでは、タイムスタンプに基づいてパーティション化されたテーブルを使用します。スキーマ data_mart には、events という列を持つ created_at という名前のテーブルが含まれています。この events テーブルには、次の設定が含まれています。

- プライマリキー event_id および created_at。パーティションのガイドに使用される列を含める必要があります。
- ck_valid_operation テーブル列に値を適用するための検査制約 operation。
- 2 つの外部キー。1 つ (fk_orga_membership)) は外部テーブル organization で、もう 1 つ (fk_parent_event_id) は自己参照外部キーです。
- 2 つのインデックス。1 つ (idx_org_id) は外部キー用で、もう 1 つ (idx_event_type) はイベントタイプ用です。

次の DDL ステートメントは、各パーティションに自動的に含まれるこれらのオブジェクトを作成します。

```
CREATE SCHEMA data_mart;
CREATE TABLE data_mart.organization ( org_id BIGSERIAL,
        org_name TEXT,
        CONSTRAINT pk_organization PRIMARY KEY (org_id)
    );

CREATE TABLE data_mart.events(
        event_id        BIGSERIAL,
        operation       CHAR(1),
        value           FLOAT(24),
```

```
        parent_event_id BIGINT,
        event_type      VARCHAR(25),
        org_id          BIGSERIAL,
        created_at      timestamp,
        CONSTRAINT pk_data_mart_event PRIMARY KEY (event_id, created_at),
        CONSTRAINT ck_valid_operation CHECK (operation = 'C' OR operation = 'D'),
        CONSTRAINT fk_orga_membership
            FOREIGN KEY(org_id)
            REFERENCES data_mart.organization (org_id),
        CONSTRAINT fk_parent_event_id
            FOREIGN KEY(parent_event_id, created_at)
            REFERENCES data_mart.events (event_id,created_at)
    ) PARTITION BY RANGE (created_at);

CREATE INDEX idx_org_id     ON  data_mart.events(org_id);
CREATE INDEX idx_event_type ON  data_mart.events(event_type);
```

create_parent 関数を使用したパーティションの設定

pg_partman エクステンションを有効にした後、create_parent 関数を使用して、パーティションメンテナンススキーマ内でパーティションの設定を行います。以下の例では、events で作成される pg_partman エクステンションの有効化 テーブルの例を使用します。create_parent 関数を次のように呼び出します。

```
SELECT partman.create_parent( p_parent_table => 'data_mart.events',
 p_control => 'created_at',
 p_type => 'native',
 p_interval=> 'daily',
 p_premake => 30);
```

パラメータは次のとおりです。

- p_parent_table - 親パーティションテーブル。このテーブルは既に存在しており、スキーマを含めて完全修飾である必要があります。
- p_control - パーティションのベースとなる列。データタイプは、整数または時間ベースである必要があります。
- p_type - タイプは 'native' または 'partman' です。通常、パフォーマンスの向上と柔軟性のためには、native タイプを使用します。partman タイプは継承により変化します。
- p_interval - 各パーティションの時間間隔または整数の範囲。この値の例としては、daily、時間単位その他があります。

- p_premake - 新しい挿入をサポートするために事前に作成するパーティションの数。

create_parent 関数の詳細については、pg_partman ドキュメントの「Creation Functions (関数の作成)」を参照してください。

run_maintenance_proc 関数を使用したパーティションのメンテナンス設定

パーティションのメンテナンスオペレーションを実行して、自動的に新しいパーティションの作成、パーティションのデタッチ、または古いパーティションの削除ができます。パーティションのメンテナンスは、内部のスケジューラをスタートする pg_partman および pg_cron エクステンション の run_maintenance_proc 関数により異なります。pg_cron スケジューラは、データベースで定義された SQL ステートメント、関数、および手順を自動的に実行します。

次の例では、events で作成した pg_partman エクステンションの有効化 テーブルの例を使用して、パーティションのメンテナンスオペレーションを自動的に実行するように設定します。前提条件にあるように、DB インスタンスのパラメータグループで、shared_preload_libraries パラメータに pg_cron を追加します。

```
CREATE EXTENSION pg_cron;

UPDATE partman.part_config
SET infinite_time_partitions = true,
    retention = '3 months',
    retention_keep_table=true
WHERE parent_table = 'data_mart.events';
SELECT cron.schedule('@hourly', $$CALL partman.run_maintenance_proc()$$);
```

その後、前の例についてのステップバイステップの説明を確認できます。

1. DB インスタンスに関連付けられているパラメータグループを変更して、pg_cron を shared_preload_libraries パラメータ値に追加します。この変更を有効にするには、DB インスタンスの再起動が必要です。詳細については、「DB パラメータグループのパラメータの変更」を参照してください。
2. CREATE EXTENSION pg_cron; のアクセス許可を持つアカウントを使用して、コマンド rds_superuser を実行します。これにより、pg_cron エクステンションが有効になります。詳細については、「PostgreSQL pg_cron エクステンションによるメンテナンスのスケジューリング」を参照してください。
3. UPDATE partman.part_config コマンドを実行して、data_mart.events テーブルの pg_partman 設定を調整します。

4. SET . . .コマンドを実行して、以下の句を使用しながら、data_mart.events テーブルを設定します。

   a. infinite_time_partitions = true, - 制限なしで新しいパーティションを自動的に作成できるようにテーブルを設定します。

   b. retention = '3 months', - テーブルの最大保持期間を 3 か月に設定します。

   c. retention_keep_table=true - 保存期間の期限が過ぎてもテーブルが自動的に削除されないように、テーブルを構成します。代わりに、保持期間より古いパーティションは、親テーブルからのみデタッチされます。

5. SELECT cron.schedule . . .コマンドを実行して、pg_cron 関数を呼び出します。この呼び出は、pg_partman メンテナンスプロシージャの partman.run_maintenance_proc が、スケジューラにより実行される頻度を定義します。この例では、プロシージャは 1 時間ごとに実行されます。

run_maintenance_proc 関数の詳細については、pg_partman ドキュメントの「Maintenance Functions (メンテナンス機能)」を参照してください。

PostgreSQL pg_cron エクステンションによるメンテナンスのスケジューリング

PostgreSQL pg_cron エクステンションを使用すると、PostgreSQL データベース内でメンテナンスコマンドのスケジュールを組むことができます。拡張機能の詳細については、pg_cron ドキュメントの 「What is pg_cron?」 を参照してください。

pg_cron エクステンションは、Aurora PostgreSQL エンジンのバージョン 12.6 以降のバージョンでサポートされています。

pg_cron の使用の詳細については、「RDS for PostgreSQL または Aurora PostgreSQL 互換エディションのデータベースで pg_cron を使用してジョブをスケジュールする」を参照してください

トピック

- pg_cron 拡張機能のセットアップ
- データベースユーザーに pg_cron を使用する権限を付与する
- pg_cron ジョブのスケジューリング
- pg_cron 拡張機能のリファレンス

pg_cron 拡張機能のセットアップ

次のように pg_cron 拡張機能をセットアップします。

1. shared_preload_libraries パラメータ値に pg_cron を追加して、PostgreSQL DB インスタンスに関連付けられているカスタムパラメータグループを変更します。

   静的パラメータグループの変更を反映するために PostgreSQL DB インスタンスを再起動します。パラメータグループを使用する方法の詳細については、 Amazon Aurora PostgreSQL のパラメータ を参照してください。
2. PostgreSQL DB インスタンスが再起動したら、rds_superuser の許可を持つアカウントを使用して以下のコマンドを実行します。例えば、Aurora PostgreSQL DB クラスターの作成時にデフォルト設定を使用した場合は、ユーザー postgres として接続し拡張機能を作成します。

   ```
   CREATE EXTENSION pg_cron;
   ```

   pg_cron スケジューラは、postgres という名前のデフォルトの PostgreSQL データベースに設定されます。pg_cron オブジェクトはこの postgres データベースに作成され、すべてのスケジューリングアクションがこのデータベースで実行されます。
3. デフォルト設定を使用することも、ジョブをスケジュールして、PostgreSQL DB インスタンス内の他のデータベースで実行させることもできます。PostgreSQL DB インスタンス内の他のデータベースでジョブをスケジュールするには、デフォルトのデータベース以外のデータベースでの cron ジョブのスケジューリング の例を参照してください。

データベースユーザーに pg_cron を使用する権限を付与する

pg_cron 拡張機能をインストールするには、rds_superuser 権限が必要です。ただし、pg_cron の使用権限は (rds_superuser グループ/ロールのメンバーによって) 他のデータベースユーザーに付与して、各ユーザーが自分のジョブをスケジュールできるようにすることができます。本番環境での運用が改善される場合にのみ、cron スキーマへのアクセス許可を付与することをお勧めします。

cron スキーマでデータベースユーザー権限を付与するには、以下のコマンドを実行します。

```
postgres=> GRANT USAGE ON SCHEMA cron TO db-user;
```

これにより、アクセス権限のあるオブジェクトの cron ジョブをスケジュールするための cron スキーマへの *db-user* アクセス許可が付与されます。データベースユーザーに権限がない場合、以下

に示すように、エラーメッセージを postgresql.log ファイルに投稿した後にジョブは失敗します。

```
2020-12-08 16:41:00 UTC::@:[30647]:ERROR: permission denied for table table-name
2020-12-08 16:41:00 UTC::@:[27071]:LOG: background worker "pg_cron" (PID 30647) exited
 with exit code 1
```

つまり、cron スキーマへの権限を付与されているデータベースユーザーに、スケジュールを設定する予定のオブジェクト (テーブル、スキーマなど) に対する権限も付与されていることを確認します。

この cron ジョブの詳細と、その成功または失敗も cron.job_run_details テーブルにキャプチャされます。詳細については、「ジョブのスケジュール設定とステータス取得用のテーブル」を参照してください。

pg_cron ジョブのスケジューリング

次のセクションでは、pg_cron ジョブを使用してさまざまな管理タスクをスケジュールする方法について説明します。

> **Note**
>
> pg_cron ジョブの作成時は、max_worker_processes 設定が cron.max_running_jobs の数より大きいことを確認します。バックグラウンドのワーカープロセスを使い切ると、pg_cron ジョブは失敗します。pg_cron ジョブのデフォルト数は 5 です。詳しくは、「pg_cron 拡張機能の管理用パラメータ」を参照してください。

トピック

- テーブルのバキューム処理
- pg_cron の履歴テーブルの除去
- エラーのログを postgresql.log ファイルにのみ記録する
- デフォルトのデータベース以外のデータベースでの cron ジョブのスケジューリング

テーブルのバキューム処理

Autovacuum は、ほとんどの場合、バキュームのメンテナンスを実行します。ただし、特定のテーブルのバキューム処理を、選択した特定の時点にスケジュールしたい、というケースも考えられます。

以下は、cron.schedule 関数を使用して、毎日 22:00 (GMT) に特定のテーブルで VACUUM FREEZE を使用するようにジョブをセットアップする例です。

```
SELECT cron.schedule('manual vacuum', '0 22 * * *', 'VACUUM FREEZE pgbench_accounts');
 schedule
----------
1
(1 row)
```

上記の例を実行した後、次のように cron.job_run_details テーブル内の履歴を確認できます。

```
postgres=> SELECT * FROM cron.job_run_details;
jobid  | runid | job_pid | database | username | command                        |
 status    | return_message | start_time                    | end_time
-------+-------+---------+----------+----------+--------------------------------
+-----------+----------------+--------------------------------
+--------------------------------
 1     | 1     | 3395    | postgres | adminuser| vacuum freeze pgbench_accounts
 | succeeded | VACUUM         | 2020-12-04 21:10:00.050386+00 | 2020-12-04
 21:10:00.072028+00
(1 row)
```

失敗したジョブを確認するための cron.job_run_details テーブルのクエリは、次のとおりです。

```
postgres=> SELECT * FROM cron.job_run_details WHERE status = 'failed';
jobid | runid | job_pid | database | username | command                        | status
 | return_message                                  | start_time                    |
 end_time
------+-------+---------+----------+----------+--------------------------------+--------
+-------------------------------------------------+--------------------------------
+--------------------------------
 5    | 4     | 30339   | postgres | adminuser| vacuum freeze pgbench_account | failed
 | ERROR: relation "pgbench_account" does not exist | 2020-12-04 21:48:00.015145+00 |
 2020-12-04 21:48:00.029567+00
(1 row)
```

詳細については、「ジョブのスケジュール設定とステータス取得用のテーブル」を参照してください。

pg_cron の履歴テーブルの除去

cron.job_run_details テーブルには、時間の経過とともに非常に大きくなる可能性がある cron ジョブの履歴が含まれています。そのため、このテーブルをクリアにするジョブをスケジュールすることをお勧めします。例えば、トラブルシューティングの目的では、1 週間分のエントリを保持するだけで十分です。

次の例では、cron.schedule 関数を使用して、cron.job_run_details テーブルをクリアにするよう、毎日午前 0 時に実行されるジョブをスケジュールします。このジョブは過去 7 日間しか残せません。rds_superuser アカウントを使用して、以下のようなジョブをスケジュールできます。

```
SELECT cron.schedule('0 0 * * *', $$DELETE
    FROM cron.job_run_details
    WHERE end_time < now() - interval '7 days'$$);
```

(詳しくは、「ジョブのスケジュール設定とステータス取得用のテーブル」を参照してください。)

エラーのログを postgresql.log ファイルにのみ記録する

cron.job_run_details テーブルへの書き込みをしないようにするには、PostgreSQL DB インスタンスに関連付けられているパラメータグループを変更し、cron.log_run パラメータをオフに設定します。pg_cron 拡張機能によって対象のテーブルには書き込まなくなり、エラーは postgresql.log ファイルのみに記録されるようになります。詳細については、「DB パラメータグループのパラメータの変更」を参照してください。

cron.log_run パラメータの値を確認するには、次のコマンドを使用します。

```
postgres=> SHOW cron.log_run;
```

詳細については、「pg_cron 拡張機能の管理用パラメータ」を参照してください。

デフォルトのデータベース以外のデータベースでの cron ジョブのスケジューリング

pg_cron のメタデータはすべて、postgres という名前の PostgreSQL のデフォルトのデータベースに保持されます。メンテナンスの cron ジョブの実行にはバックグラウンドワーカーが使用されるため、PostgreSQL DB インスタンス内の任意のデータベースでジョブのスケジューリングが可能です。

1. cron データベースで、cron.schedule を使用して通常どおりにジョブをスケジュールします。

```
postgres=> SELECT cron.schedule('database1 manual vacuum', '29 03 * * *', 'vacuum
 freeze test_table');
```

2. 作成したジョブのデータベース列を、rds_superuser ロールを持つユーザーとして更新し、そのジョブを PostgreSQL DB インスタンス内の別のデータベースで実行できるようにします。

```
postgres=> UPDATE cron.job SET database = 'database1' WHERE jobid = 106;
```

3. cron.job テーブルのクエリを実行して確認します。

```
postgres=> SELECT * FROM cron.job;
jobid | schedule    | command                        | nodename  | nodeport |
 database | username  | active | jobname
------+-------------+--------------------------------+-----------+----------
+----------+-----------+--------+------------------------
106   | 29 03 * * * | vacuum freeze test_table       | localhost | 8192     |
 database1| adminuser | t      | database1 manual vacuum
  1   | 59 23 * * * | vacuum freeze pgbench_accounts | localhost | 8192     |
 postgres | adminuser | t      | manual vacuum
(2 rows)
```

> **Note**
>
> 状況によっては、別のデータベースで実行する cron ジョブを追加することがあります。このような場合、ジョブは、正しいデータベース列を更新する前に、デフォルトのデータベース (postgres) で実行しようとする可能性があります。ユーザー名に権限がある場合、デフォルトのデータベースでジョブが正常に実行されます。

pg_cron 拡張機能のリファレンス

pg_cron エクステンションでは、次のパラメータ、関数、およびテーブルを使用できます。詳細については、pg_cron ドキュメントの What is pg_cron? を参照してください。

トピック

- pg_cron 拡張機能の管理用パラメータ
- 関数リファレンス: cron.schedule
- 関数リファレンス: cron.unschedule

- ジョブのスケジュール設定とステータス取得用のテーブル

pg_cron 拡張機能の管理用パラメータ

pg_cron エクステンションの動作を制御するパラメータの一覧を次に示します。

Parameter	説明
cron.database_name	pg_cron メタデータが保持されるデータベース。
cron.host	PostgresSQL に接続するためのホスト名。この値は変更できません。
cron.log_run	job_run_details テーブルで実行されるすべてのジョブを口グに記録します。有効な値は on または off です。詳細については、「ジョブのスケジュール設定とステータス取得用のテーブル」を参照してください。
cron.log_statement	実行する前に、すべての cron ステートメントを記録します。有効な値は on または off です。
cron.max_running_jobs	同時に実行できるジョブの最大数。
cron.use_background_workers	クライアントセッションの代わりにバックグラウンドワーカーを使用します。この値は変更できません。

次の SQL コマンドを使用して、これらのパラメータとその値を表示します。

```
postgres=> SELECT name, setting, short_desc FROM pg_settings WHERE name LIKE 'cron.%'
 ORDER BY name;
```

関数リファレンス: cron.schedule

この関数は、cron ジョブをスケジュールします。このジョブは、デフォルトの `postgres` データベースで初期にスケジュールされます。この関数は、ジョブ識別子を表す `bigint` の値を返します。PostgreSQL DB インスタンス内の他のデータベースで実行するようにジョブをスケジュールするには、デフォルトのデータベース以外のデータベースでの cron ジョブのスケジューリング の例を参照してください。

この関数には、2 つの構文形式があります。

構文

```
cron.schedule (job_name,
    schedule,
    command
);

cron.schedule (schedule,
    command
);
```

パラメータ

Parameter	説明
`job_name`	cron ジョブの名前。
`schedule`	cron ジョブのスケジュールを示すテキスト。形式はスタンダードの cron 形式です。
`command`	実行するコマンドのテキスト。

例

```
postgres=> SELECT cron.schedule ('test','0 10 * * *', 'VACUUM pgbench_history');
 schedule
----------
      145
(1 row)
```

```
postgres=> SELECT cron.schedule ('0 15 * * *', 'VACUUM pgbench_accounts');
 schedule
----------
      146
(1 row)
```

関数リファレンス: cron.unschedule

この関数は、cron ジョブを削除します。job_name または job_id を指定できます。ポリシーにより、ユーザーがジョブのスケジュールを削除する所有者であることが確認されます。この関数は、成功または失敗を示すブール値を返します。

関数の構文形式は以下のとおりです。

構文

```
cron.unschedule (job_id);

cron.unschedule (job_name);
```

パラメータ

Parameter	説明
job_id	cron ジョブがスケジュールされたときに cron.schedule 関数から返されたジョブ識別子。
job_name	cron.schedule 関数でスケジュールされた cron ジョブの名前。

例

```
postgres=> SELECT cron.unschedule(108);
 unschedule
------------
 t
(1 row)
```

```
postgres=> SELECT cron.unschedule('test');
 unschedule
------------
 t
(1 row)
```

ジョブのスケジュール設定とステータス取得用のテーブル

以下のテーブルは、cron ジョブのスケジューリングのためと、そのジョブがどのように完了したかを記録するために使用されます。

表	説明
`cron.job`	スケジュールされた各ジョブに関するメタデータが含まれます。このテーブルとのほとんどのやり取りは、`cron.schedule` 関数および `cron.unschedule` 関数を使用して行う必要があります。 **Important** 更新または挿入の権限をこのテーブルに直接与えないようにお勧めします。そうすることで、ユーザーは `username` 列を更新し、`rds-superuser` として実行できるようになります。
`cron.job_run_details`	ここには、過去にスケジュールされ実行されたジョブに関する履歴の情報が含まれます。これは、実行されたジョブのステータス、返されたメッセージ、およびスタートと終了の時間を調査する場合に便利です。 **Note** このテーブルが無期限に増えないようにするには、定期的に削除してください。例については、「pg_cron の履歴テーブルの除去」を参照してください。

pgAudit を使用してデータベースのアクティビティを記録する

金融機関、政府機関、および多くの業界では、規制要件を満たすために監査ログを保存する必要があります。Aurora PostgreSQL DB クラスター で PostgreSQL 監査拡張機能 (pgAudit) を使用することで、監査人が通常必要とする詳細なレコードや規制要件を満たすための詳細なレコードをキャプチャできます。例えば、pgAudit 拡張機能を設定して、特定のデータベースやテーブルに加えられた変更を追跡したり、変更を加えたユーザーやその他の多くの詳細を記録したりできます。

pgAudit 拡張機能は、ログメッセージをより詳細に拡張することにより、ネイティブの PostgreSQL ログ記録インフラストラクチャの機能に基づいて構築されています。つまり、監査ログは、他のログメッセージを表示するのと同じ方法を使用します。PostgreSQL ログ記録の詳細については、「Aurora PostgreSQL データベースログファイル」を参照してください。

pgAudit 拡張機能は、クリアテキストパスワードなどの機密データをログから編集します。Aurora PostgreSQL DB クラスター が、Aurora PostgreSQL DB クラスターのクエリログ記録をオンにする で説明されているようにデータ操作言語 (DML) ステートメントをログに記録するように設定されている場合は、PostgreSQL Audit 拡張機能を使用することでクリアテキストパスワードの問題を回避できます。

データベースインスタンスの監査は、きわめて詳細に構成できます。すべてのデータベースとすべてのユーザーを監査できます。また、特定のデータベース、ユーザー、その他のオブジェクトのみを監査することもできます。特定のユーザーやデータベースを監査対象から明示的に除外することもできます。詳細については、「監査ログからのユーザーまたはデータベースの除外」を参照してください。

キャプチャできる詳細の量を考慮すると、pgAudit を使用する場合はストレージ消費量を監視することをお勧めします。

pgAudit 拡張モジュールは、使用可能なすべての Aurora PostgreSQL バージョンでサポートされています。Aurora PostgreSQL のバージョンでサポートされている pgAudit のリストについては、Aurora PostgreSQL のリリースノートの「Amazon Aurora PostgreSQL の拡張機能バージョン」を参照してください。

トピック

- pgAudit 拡張機能のセットアップ
- データベースオブジェクトの監査
- 監査ログからのユーザーまたはデータベースの除外
- pgAudit 拡張機能のリファレンス

pgAudit 拡張機能のセットアップ

Aurora PostgreSQL DB クラスターに pgAudit 拡張機能を設定するには、まず RDS for PostgreSQL DB インスタンスのの共有ライブラリに pgAudit を追加します。Aurora PostgreSQL DB クラスター用のカスタム DB クラスターでパラメータグループ。カスタム DB クラスターパラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。次に、pgAudit 拡張機能をインストールします。最後に、監査するデータベースとオブジェクトを指定します。このセクションの手順で、方法を示します。AWS Management Console または AWS CLI を使用できます。

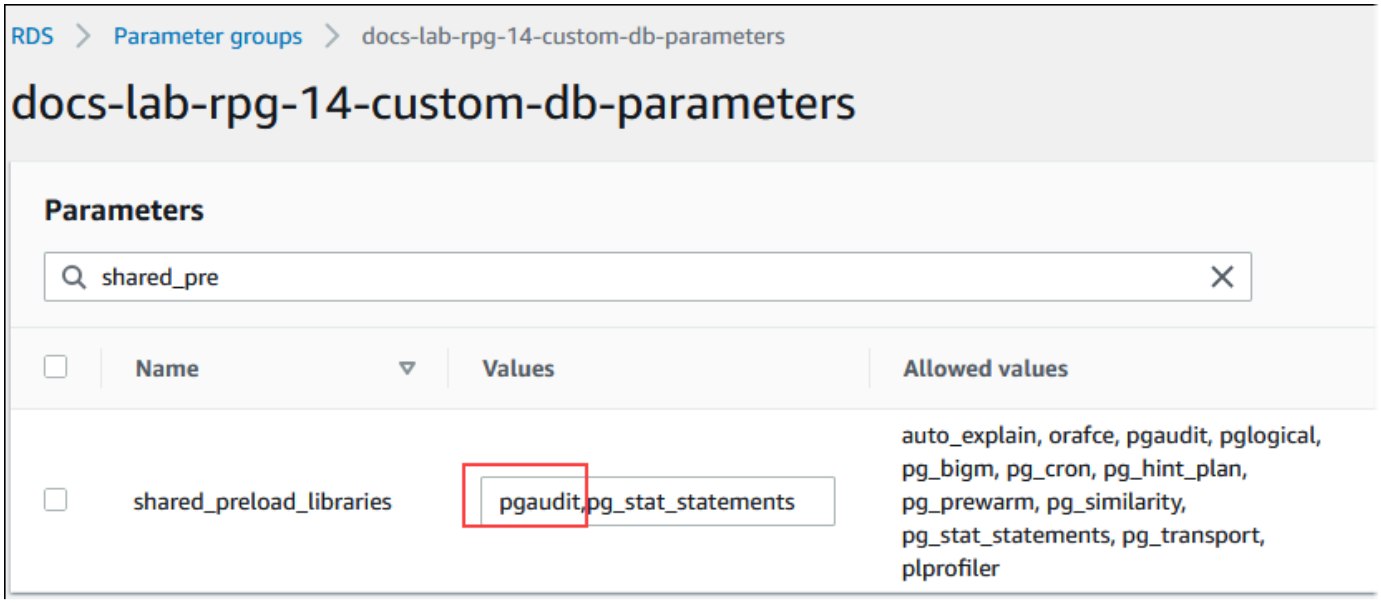
これらすべてのタスクを実行するには、`rds_superuser` ロールとして権限が必要です。

以下の手順では、Aurora PostgreSQL DB クラスター がカスタム DB クラスター パラメータグループに関連付けられていることを前提としています。

コンソール

pgAudit 拡張機能をセットアップするには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、Aurora PostgreSQL DB クラスターのライターインスタンス を選択します。
3. Aurora PostgreSQL DB クラスターライターインスタンスの [Configuration] (設定) タブを開きます。インスタンスの詳細の中から、パラメータグループのリンクを見つけてください。
4. リンクを選択して、Aurora PostgreSQL DB クラスターに関連するカスタムパラメータを開きます。
5. パラメータ検索フィールドに、`shared_pre` を入力して `shared_preload_libraries` パラメータを検索します。
6. プロパティ値にアクセスするには、[Edit parameters] (パラメータの編集) を選択します。
7. [Values] (値) フィールドのリストに `pgaudit` を追加します。値のリスト内の項目を区切るにはカンマを使用します。



8. Aurora PostgreSQL DB クラスターのライターインスタンス を再起動して、`shared_preload_libraries` パラメータの変更を有効にします。
9. インスタンスが使用可能になったら、pgAudit が初期化されていることを確認します。`psql` を使用して Aurora PostgreSQL DB クラスターのライターインスタンス、次のコマンドを実行します。

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pgaudit
(1 row)
```

10. pgAudit を初期化すると、拡張機能を作成できるようになりました。`pgaudit` 拡張機能はデータ定義言語 (DDL) ステートメントを監査するためのイベントトリガーをインストールするため、ライブラリを初期化した後に拡張機能を作成する必要があります。

```
CREATE EXTENSION pgaudit;
```

11. `psql` セッションを終了します。

```
labdb=> \q
```

12. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。

13. リストで pgaudit.log パラメータを検索し、ユースケースに応じた値に設定します。例えば、次の画像に示すように pgaudit.log パラメータを write に設定すると、ログへの挿入、更新、削除、およびその他のタイプの変更がキャプチャされます。

pgaudit.log パラメータには、次のいずれかの値を選択することもできます。

- none – これはデフォルトです。データベースの変更は記録されません。
- すべて — すべてをログに記録します (読み取り、書き込み、関数、ロール、DDL、その他)。
- ddl – ROLE クラスに含まれていない、すべてのデータ定義言語 (DDL) ステートメントのログ記録。
- function – 関数呼び出し、および DO ブロックのログ記録。
- misc – DISCARD、FETCH、CHECKPOINT、VACUUM、SET など、さまざまなコマンドのログ記録。
- read – SELECT および COPY のログ記録 (ソースがリレーション (テーブルなどの) またはクエリの場合)。
- role – GRANT、REVOKE、CREATE ROLE、ALTER ROLE、DROP ROLE など、ロールと権限に関連するステートメントのログ記録。
- write – INSERT、UPDATE、DELETE、TRUNCATE、および COPY のログ記録 (送信先がリレーション (テーブル) の場合)。

14. [Save changes] (変更の保存) をクリックします。
15. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
16. データベースリストから Aurora PostgreSQL DB クラスターのライターインスタンス を選択して選択し、アクションメニューから [Reboot] (再起動) を選択します。

AWS CLI

pgAudit をセットアップするには

AWS CLI を使用して pgAudit を設定するには、次の手順に示すように、modify-db-parameter-group オペレーションを呼び出してカスタムパラメータグループの監査ログパラメータを変更します。

1. 次の AWS CLI コマンドを使用してshared_preload_librariesパラメータに pgaudit を追加します。

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pgaudit,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. 次の AWS CLI コマンドを使用して Aurora PostgreSQL DB クラスターのライターインスタンスを再起動し、pgaudit ライブラリを初期化します。

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

3. インスタンスが使用可能になると、pgaudit が初期化されていることを確認できます。psql を使用して Aurora PostgreSQL DB クラスターのライターインスタンス、次のコマンドを実行します。

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pgaudit
(1 row)
```

pgAudit を初期化すると、拡張機能を作成できるようになりました。

```
CREATE EXTENSION pgaudit;
```

4. AWS CLI を使用できるように psql セッションを終了します。

```
labdb=> \q
```

5. 次の AWS CLI コマンドを使用して、セッション監査ログによって記録するステートメントのクラスを指定します。この例では、pgaudit.log パラメータを write に設定し、ログへの挿入、更新、削除をキャプチャします。

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=pgaudit.log,ParameterValue=write,ApplyMethod=pending-reboot" \
   --region aws-region
```

pgaudit.log パラメータには、次のいずれかの値を選択することもできます。

- none – これはデフォルトです。データベースの変更は記録されません。
- すべて — すべてをログに記録します (読み取り、書き込み、関数、ロール、DDL、その他)。
- ddl – ROLE クラスに含まれていない、すべてのデータ定義言語 (DDL) ステートメントのログ記録。
- function – 関数呼び出し、および DO ブロックのログ記録。
- misc – DISCARD、FETCH、CHECKPOINT、VACUUM、SET など、さまざまなコマンドのログ記録。
- read – SELECT および COPY のログ記録 (ソースがリレーション (テーブルなどの) またはクエリの場合)。
- role – GRANT、REVOKE、CREATE ROLE、ALTER ROLE、DROP ROLE など、ロールと権限に関連するステートメントのログ記録。
- write – INSERT、UPDATE、DELETE、TRUNCATE、および COPY のログ記録 (送信先がリレーション (テーブル) の場合)。

次の AWS CLI コマンドを使用して、Aurora PostgreSQL DB クラスターのライターインスタンス を再起動します。

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

データベースオブジェクトの監査

Aurora PostgreSQL DB クラスター に pgAudit をセットアップし、要件に合わせて設定すると、より詳細な情報が PostgreSQL ログに取得されます。例えば、デフォルトの PostgreSQL ログ設定はデータベーステーブルに変更が加えられた日付と時刻を識別しますが、pgAudit 拡張機能では、拡張機能のパラメータの設定方法に応じて、スキーマ、変更を行ったユーザー、その他の詳細をログエントリに含めることができます。監査を設定して、次の方法で変更を追跡できます。

- セッションごとに、ユーザー別。セッションレベルでは、完全修飾コマンドテキストをキャプチャできます。
- オブジェクトごとに、ユーザー別、データベース別。

オブジェクト監査機能は、システムで rds_pgaudit ロールを作成し、そのロールをカスタムパラメータグループの pgaudit.role パラメータに追加したときに有効になります。デフォルトでは、pgaudit.role パラメータは設定されておらず、許容される値は rds_pgaudit だけです。以下の手順は、pgaudit が初期化され、pgAudit 拡張機能のセットアップ の手順に従って pgaudit 拡張機能を作成したことを前提としています。

```
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:LOG: statement: SELECT feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:LOG: AUDIT: SESSION,2,1,READ,SELECT,TABLE,public.support,"SELECT
feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;",<none>
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:LOG: QUERY STATISTICS
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:DETAIL: ! system usage stats:
! 0.009494 s user, 0.007442 s system, 0.141985 s elapsed
! [0.022327 s user, 0.007442 s system total]
```

この例に示すように、「LOG: AUDIT: SESSION」行には、テーブルとそのスキーマなどの詳細情報が表示されます。

オブジェクト監査をセットアップするには

1. psql を使用して Aurora PostgreSQL DB クラスターのライターインスタンス、

   ```
   psql --host=your-instance-name.aws-region.rds.amazonaws.com --port=5432 --
   username=postgrespostgres --password --dbname=labdb
   ```

2. 次のコマンドを使用して、rds_pgaudit というデータベースロールを作成します。

   ```
   labdb=> CREATE ROLE rds_pgaudit;
   CREATE ROLE
   ```

```
labdb=>
```

3. psql セッションを終了します。

```
labdb=> \q
```

次のステップでは、AWS CLI を使用してカスタムパラメータグループの監査ログパラメータを変更します。

4. 次の AWS CLI コマンドを使用して、pgaudit.role パラメータを rds_pgaudit に設定します。デフォルトでは、このパラメータは空で、rds_pgaudit は、唯一の許容値です。

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=pgaudit.role,ParameterValue=rds_pgaudit,ApplyMethod=pending-reboot"
 \
   --region aws-region
```

5. 次の AWS CLI コマンドを使用して Aurora PostgreSQL DB クラスターのライターインスタンスを再起動し、パラメータの変更を有効にします。

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

6. 次のコマンドを実行して、pgaudit.role が rds_pgaudit に設定されたことを確認します。

```
SHOW pgaudit.role;
pgaudit.role
------------------
rds_pgaudit
```

pgAudit ログ記録をテストするには、監査するサンプルコマンドをいくつか実行します。例えば、次のコマンドを実行します。

```
CREATE TABLE t1 (id int);
GRANT SELECT ON t1 TO rds_pgaudit;
SELECT * FROM t1;
id
----
```

```
(0 rows)
```

データベースログには、次のようなエントリが含まれます。

```
...
2017-06-12 19:09:49 UTC:...:rds_test@postgres:[11701]:LOG: AUDIT:
OBJECT,1,1,READ,SELECT,TABLE,public.t1,select * from t1;
...
```

ログの表示方法については、「Amazon Aurora ログファイルのモニタリング」を参照してください。

pgAudit 拡張機能の詳細については、GitHub で「pgAudit」を参照してください。

監査ログからのユーザーまたはデータベースの除外

Aurora PostgreSQL データベースログファイル で説明したように、PostgreSQL ログはストレージ容量を使用します。pgAudit 拡張機能を使用すると、追跡する変更に応じて、ログに収集されるデータの量が、程度の差はありますが、増加します。Aurora PostgreSQL DB クラスター内のすべてのユーザーまたはデータベースを監査する必要はないかもしれません。

ストレージへの影響を最小限に抑え、監査記録を不必要にキャプチャしないようにするには、ユーザーとデータベースを監査対象から除外できます。特定のセッション内の口ギングを変更することもできます。次の例は、その方法を示しています。

> **Note**
>
> セッションレベルのパラメータ設定は、Aurora PostgreSQL DB クラスターのライターインスタンスのカスタム DB クラスターパラメータグループ の設定よりも優先されます。データベースユーザーに監査口グ設定の設定をバイパスさせたくない場合は、必ず権限を変更してください。

Aurora PostgreSQL DB クラスター が、すべてのユーザーとデータベースについて同じレベルのアクティビティを監査するように設定されているとします。次に、myuser ユーザーを監査しないことにします。次の SQL コマンドを使用して、myuser の監査機能を無効にできます。

```
ALTER USER myuser SET pgaudit.log TO 'NONE';
```

次に、次のクエリを使用して pgaudit.log の user_specific_settings 列をチェックし、パラメータが NONE に設定されていることを確認できます。

```
SELECT
    usename AS user_name,
    useconfig AS user_specific_settings
FROM
    pg_user
WHERE
    usename = 'myuser';
```

次のような出力が表示されます。

```
 user_name | user_specific_settings
-----------+------------------------
 myuser    | {pgaudit.log=NONE}
(1 row)
```

次のコマンドを使用すると、データベースとのセッションの最中に、指定したユーザーのログをオフにできます。

```
ALTER USER myuser IN DATABASE mydatabase SET pgaudit.log TO 'none';
```

次のクエリを使用して、特定のユーザーとデータベースの組み合わせの pgaudit.log の設定列を確認します。

```
SELECT
    usename AS "user_name",
    datname AS "database_name",
    pg_catalog.array_to_string(setconfig, E'\n') AS "settings"
FROM
    pg_catalog.pg_db_role_setting s
    LEFT JOIN pg_catalog.pg_database d ON d.oid = setdatabase
    LEFT JOIN pg_catalog.pg_user r ON r.usesysid = setrole
WHERE
    usename = 'myuser'
    AND datname = 'mydatabase'
ORDER BY
    1,
    2;
```

以下のような出力が表示されます。

```
  user_name | database_name |     settings
-----------+---------------+-----------------
 myuser    | mydatabase    | pgaudit.log=none
(1 row)
```

myuser の監査を無効化した後に、mydatabase の変更を追跡しないことにしました。次のコマンドを使用して、その特定のデータベースの監査を無効化します。

```
ALTER DATABASE mydatabase SET pgaudit.log to 'NONE';
```

次に、以下のクエリで database_specific_settings 列を確認し、pgaudit.log が NONE に設定されていることを確認します。

```
SELECT
a.datname AS database_name,
b.setconfig AS database_specific_settings
FROM
pg_database a
FULL JOIN pg_db_role_setting b ON a.oid = b.setdatabase
WHERE
a.datname = 'mydatabase';
```

次のような出力が表示されます。

```
 database_name | database_specific_settings
---------------+----------------------------
 mydatabase    | {pgaudit.log=NONE}
(1 row)
```

myuser の設定をデフォルト設定に戻すには、次のコマンドを使用します。

```
ALTER USER myuser RESET pgaudit.log;
```

設定をデータベースのデフォルト設定に戻すには、次のコマンドを使用します。

```
ALTER DATABASE mydatabase RESET pgaudit.log;
```

ユーザーとデータベースをデフォルト設定にリセットするには、次のコマンドを使用します。

```
ALTER USER myuser IN DATABASE mydatabase RESET pgaudit.log;
```

また、pgaudit.log パラメータに pgaudit.log を他の許容値のいずれかに設定することで、特定のイベントをログに記録することもできます (詳しくは、「pgaudit.log パラメータの許容設定のリスト」を参照してください)。

```
ALTER USER myuser SET pgaudit.log TO 'read';
ALTER DATABASE mydatabase SET pgaudit.log TO 'function';
ALTER USER myuser IN DATABASE mydatabase SET pgaudit.log TO 'read,function'
```

pgAudit 拡張機能のリファレンス

このセクションにリストされている 1 つまたは複数のパラメータを変更することで、監査ログに必要な詳細レベルを指定できます。

pgAudit 動作の制御

監査ログは、次のテーブルに示す 1 つ以上のパラメータを変更することで制御できます。

パラメータ	説明
pgaudit.log	セッション監査ログ記録によってログに記録されるステートメントのクラスを指定します。許容値には、ddl、関数、その他、読み取り、ロール、書き込み、なし、すべてが含まれます。(詳しくは、「pgaudit.log パラメータの許容設定のリスト」を参照してください)。
pgaudit.log_catalog	オンにすると (1 に設定)、ステートメント内のすべてのリレーションが pg_catalog 内にある場合に、ステートメントを監査証跡に追加します。
pgaudit.log_level	ログエントリに使用されるログレベルを指定します。指定できる値は debug5、debug4、debug3、debug2、debug1、info、notice、warning、log です。
pgaudit.log_parameter	オン (1 に設定) すると、ステートメントとともに渡されたパラメータが監査ログに記録されます。

パラメータ	説明
pgaudit.log_relation	オンにすると (1 に設定)、セッションの監査ログで、SELECT ステートメントまたは DML ステートメントで参照されるリレーション (TABLE、VIEW など) ごとに個別のログエントリが作成されます。
pgaudit.log_statement_once	ログ記録に、ステートメントテキストとパラメータを、ステートメントとサブステートメントの組み合わせの最初のログエントリとともに含めるか、すべてのエントリとともに含めるかを指定します。
pgaudit.role	オブジェクト監査ログ記録に使用するマスターロールを指定します。唯一許容されるエントリは rds_pgaudit です。

pgaudit.log パラメータの許容設定のリスト

Value	説明
なし	これがデフォルトです。データベースの変更は記録されません。
すべて	すべてをログに記録します (読み取り、書き込み、関数、ロール、DDL、その他)。
ddl	ROLE クラスに含まれていない、すべてのデータ定義言語 (DDL) ステートメントのログ記録。
関数	関数呼び出し、および DO ブロックのログ記録。
misc	DISCARD、FETCH、CHECKPOINT 、VACUUM、SET など、さまざまなコマンドのログ記録。
read	SELECT および COPY のログ記録 (ソースがリレーション (テーブルなどの) またはクエリの場合)。
ロール	GRANT、REVOKE、CREATE ROLE、ALTER ROLE、DROP ROLE など、ロールと権限に関連するステートメントのログ記録。

Value	説明
書き込み	INSERT、UPDATE、DELETE、TRUNCATE、および COPY の口グ記録 (送信先がリレーションの場合)。

セッション監査で複数のイベントタイプを口グ記録するには、カンマ区切りリストを使用します。すべてのイベントタイプを口グ記録するには、pgaudit.log を ALL に設定します。DB インスタンスを再起動して、変更を適用します。

オブジェクト監査では、監査の口グ記録を絞り込み、特定のリレーションを操作できます。例えば、1 つまたは複数のテーブルで、READ オペレーションの口グ記録を監査するよう指定できます。

pglogical を使用してインスタンス間でデータを同期する

現在利用可能なすべての Aurora PostgreSQL バージョンは、pglogical 拡張機能をサポートしています。pglogical 拡張は、バージョン 10 で PostgreSQL により導入された機能的に類似した論理レプリケーション機能よりも前のものです。詳細については、「Aurora での PostgreSQL 論理的なレプリケーションの使用」を参照してください。

pglogical 拡張が、2 つ以上の Aurora PostgreSQL DB クラスター間の論理レプリケーションをサポートします。また、異なる PostgreSQL バージョン間のレプリケーション、および RDS for PostgreSQL DB と Aurora PostgreSQL DB クラスターで実行されているデータベース間のレプリケーションもサポートしています。pglogical 拡張は、公開/サブスクライブモデルを使用して、テーブルやその他のオブジェクト (シーケンスなど) への変更をパブリッシャーからサブスクライバーに複製します。パブリッシャーノードからサブスクライバーノードに変更が確実に同期されるようにするには、レプリケーションスロットを使用し、次のように定義されます。

- パブリッシャーノードは、他のノードにレプリケートされるデータのソースである Aurora PostgreSQL DB クラスターです。パブリッシャーノードは、パブリケーションセットでレプリケートするテーブルを定義します。
- サブスクライバーノードは、公開者から WAL の更新を受け取る Aurora PostgreSQL DB クラスターです。サブスクライバーは、パブリッシャーに接続してデコードされた WAL データを取得するためのサブスクリプションを作成します。サブスクライバーがサブスクリプションを作成すると、パブリッシャーノードに複製スロットが作成されます。

pglogical 拡張の設定についての情報は、以下を参照してください。

トピック

pglogical 拡張の要件と制限

Aurora PostgreSQL の現在利用可能なすべてのリリースが `pglogical` 拡張機能をサポートしています。

パブリッシャーノードとサブスクライバーノードの両方を論理レプリケーション用に設定する必要があります。

サブスクライバーからパブリッシャーにレプリケートするテーブルは、名前とスキーマが同じである必要があります。これらのテーブルにも同じ列が含まれている必要があり、列は同じデータ型を使用する必要があります。パブリッシャーテーブルとサブスクライバーテーブルの両方に同じプライマリキーが必要です。一意の制約事項としては PRIMARY KEY のみを使用することをお勧めします。

サブスクライバーノードのテーブルには、CHECK 制約と NOT NULL 制約について、パブリッシャーノードのテーブルよりも許可度が高い制約を設定できます。

`pglogical` 拡張は、PostgreSQL (バージョン 10 以降) に組み込まれている論理レプリケーション機能ではサポートされていない双方向レプリケーションなどの機能を提供します。詳細については、「PostgreSQL bi-directional replication using pglogical」(pglogical を使用した PostgreSQL の双方向レプリケーション) を参照してください。

pglogical 拡張のセットアップ

Aurora PostgreSQL DB クラスターに `pglogical` 拡張機能を設定するには、Aurora PostgreSQL DB クラスター用のカスタム DB クラスターのパラメータグループの共有ライブラリに `pglogical` を追加します。また、論理デコードをオンにするには、`rds.logical_replication` パラメータの値を `1` に設定する必要があります。最後に、データベースに拡張を作成します。これらのタスクには、AWS Management Console または AWS CLI を使用できます。

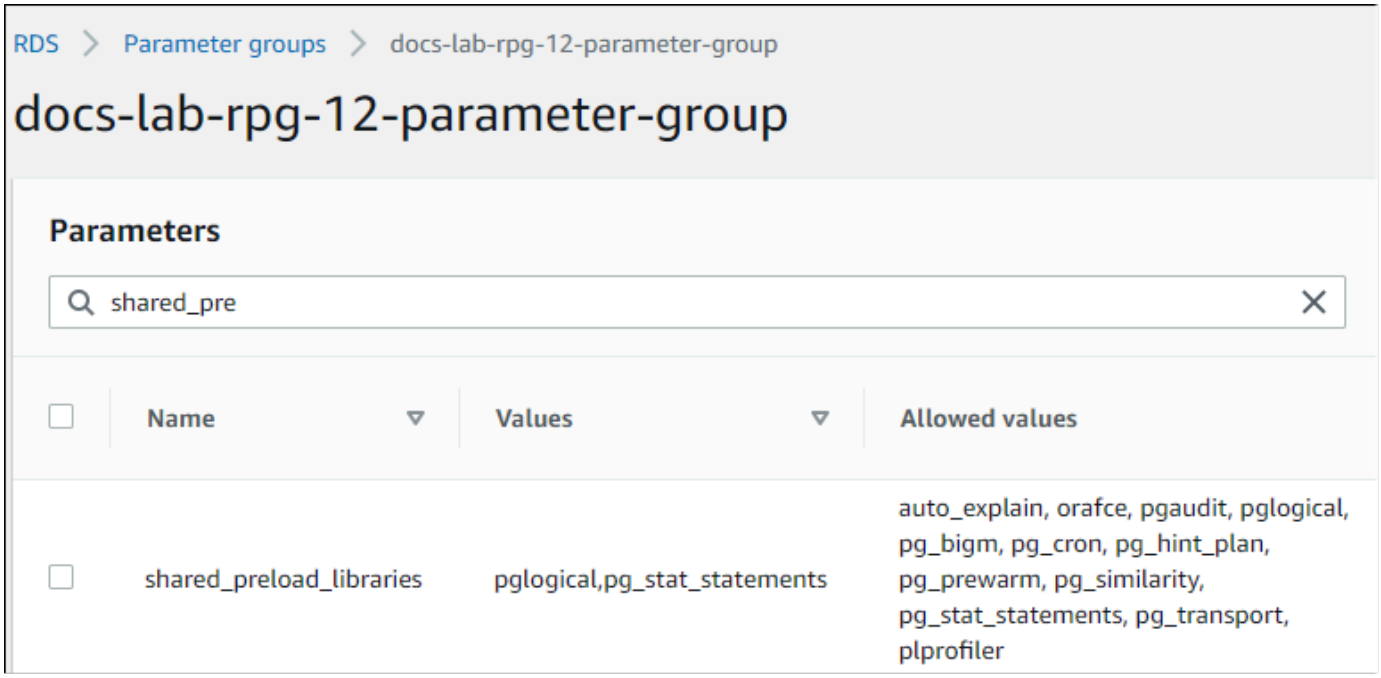
これらのタスクを実行するには、`rds_superuser` ロールとしてアクセス許可が必要です。

以下の手順では、Aurora PostgreSQL DB クラスター がカスタム DB クラスター パラメータグループに関連付けられていることを前提としています。カスタム DB クラスターパラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。

コンソール

pglogical 拡張をセットアップするには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、Aurora PostgreSQL DB クラスターのライターインスタンス を選択します。
3. Aurora PostgreSQL DB クラスターライターインスタンスの [Configuration] (設定) タブを開きます。インスタンスの詳細の中から、パラメータグループのリンクを見つけてください。
4. リンクを選択して、Aurora PostgreSQL DB クラスターに関連するカスタムパラメータを開きます。
5. パラメータ検索フィールドに、`shared_pre` を入力して `shared_preload_libraries` パラメータを検索します。
6. プロパティ値にアクセスするには、[Edit parameters] (パラメータの編集) を選択します。
7. [Values] (値) フィールドのリストに `pglogical` を追加します。値のリスト内の項目を区切るにはカンマを使用します。



8. rds.logical_replication パラメータを見つけて 1 に設定し、論理レプリケーションをオンにします。
9. Aurora PostgreSQL DB クラスターのライターインスタンス を再起動して、変更を有効にします。
10. インスタンスが使用可能になったら、psql (または pgAdmin) を使用して Aurora PostgreSQL DB クラスターのライターインスタンスに接続します。

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

11. pglogical が初期化されていることを確認するには、次のコマンドを実行します。

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pglogical
(1 row)
```

12. 次のように、論理デコードを有効にする設定を確認します。

```
SHOW wal_level;
wal_level
-----------
 logical
(1 row)
```

13. 次のように拡張を作成します。

```
CREATE EXTENSION pglogical;
EXTENSION CREATED
```

14. [Save changes] (変更を保存) をクリックします。
15. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
16. データベースリストから Aurora PostgreSQL DB クラスターのライターインスタンス を選択して選択し、アクションメニューから [Reboot] (再起動) を選択します。

AWS CLI

pglogical 拡張のセットアップするには

AWS CLI を使用して pglogical を設定するには、次の手順に示すように、modify-db-parameter-group オペレーションを呼び出してカスタムパラメータグループの特定のパラメータを変更します。

1. 次の AWS CLI コマンドを使用して shared_preload_libraries パラメータに pglogical を追加します。

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pglogical,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. 次の AWS CLI コマンドを使用して rds.logical_replication を 1 に設定し、Aurora PostgreSQL DB クラスターのライターインスタンスの論理デコード機能をオンにします。

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=rds.logical_replication,ParameterValue=1,ApplyMethod=pending-
reboot" \
   --region aws-region
```

3. 次の AWS CLI コマンドを使用して Aurora PostgreSQL DB クラスターのライターインスタンスを再起動し、pglogical ライブラリを初期化します。

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

4. インスタンスが使用可能になったら、psql を使用して Aurora PostgreSQL DB クラスターのライターインスタンスに接続します。

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

5. 次のように拡張を作成します。

```
CREATE EXTENSION pglogical;
EXTENSION CREATED
```

6. 次の AWS CLI コマンドを使用して、Aurora PostgreSQL DB クラスターのライターインスタンス を再起動します。

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

Aurora PostgreSQL DB クラスターに論理レプリケーションを設定する

以下の手順では、2 つの Aurora PostgreSQL DB クラスター間で論理レプリケーションを開始する方法を示しています。これらのステップでは、ソース (パブリッシャー) とターゲット (サブスクライバー) の両方に、pglogical 拡張のセットアップ で説明されているように pglogical 拡張が設定されていることを前提としています。

パブリッシャーノードを作成し、複製するテーブルを定義するには

これらのステップは、別のノードに複製する 1 つ以上のテーブルがあるデータベースを含むライターインスタンスが Aurora PostgreSQL DB クラスターにあることを前提としています。サブスクライバー上のパブリッシャーからテーブル構造を再作成する必要があるため、まず必要に応じてテーブル構造を取得します。そのためには、psql1 メタコマンド \d *tablename* を使用してサブスクライバーインスタンスに同じテーブルを作成します。次の手順では、デモンストレーションを目的として、パブリッシャー (ソース) でサンプルテーブルを作成します。

1. psql を使用して、サブスクライバーのソースとして使用したいテーブルがあるインスタンスに接続します。

```
psql --host=source-instance.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

複製する既存のテーブルがない場合は、次のようにサンプルテーブルを作成できます。

a. 次の SQL ステートメントを使用してサンプルテーブルを作成します。

```
CREATE TABLE docs_lab_table (a int PRIMARY KEY);
```

b. 次の SQL ステートメントを使用して、生成されたデータをテーブルに入力します。

```
INSERT INTO docs_lab_table VALUES (generate_series(1,5000));
INSERT 0 5000
```

c. 次の SQL ステートメントを使用して、テーブルにデータが存在することを確認します。

```
SELECT count(*) FROM docs_lab_table;
```

2. 次のように、この Aurora PostgreSQL DB クラスターをパブリッシャーノードとして指定します。

```
SELECT pglogical.create_node(
    node_name := 'docs_lab_provider',
    dsn := 'host=source-instance.aws-region.rds.amazonaws.com port=5432
 dbname=labdb');
 create_node
-------------
   3410995529
(1 row)
```

3. 複製するテーブルをデフォルトのレプリケーションセットに追加します。レプリケーションセットの詳細については、pglogical ドキュメントの「Replication sets」(レプリケーションセット)を参照してください。

```
SELECT pglogical.replication_set_add_table('default', 'docs_lab_table', 'true',
 NULL, NULL);
 replication_set_add_table
  --------------------------
  t
  (1 row)
```

パブリッシャーノードの設定が完了しました。これで、パブリッシャーから更新を受け取るようにサブスクライバーノードを設定できます。

サブスクライバーノードを設定し、更新を受信するサブスクリプションを作成するには

これらのステップは、Aurora PostgreSQL DB クラスターが pglogical 拡張機能を使用してセットアップされていることを前提としています。詳細については、「pglogical 拡張のセットアップ」を参照してください。

1. psql を使用して、パブリッシャーから更新を受け取るインスタンスに接続します。

```
psql --host=target-instance.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

2. サブスクライバーの Aurora PostgreSQL DB クラスターで、パブリッシャーに存在するのと同じテーブルを作成します。この例では、テーブルは docs_lab_table です。次に示すようにテーブルを作成できます。

```
CREATE TABLE docs_lab_table (a int PRIMARY KEY);
```

3. このテーブルが空であることを確認します。

```
SELECT count(*) FROM docs_lab_table;
 count
-------
  0
(1 row)
```

4. 次のように、この Aurora PostgreSQL DB クラスターをサブスクライバーノードとして指定します。

```
SELECT pglogical.create_node(
    node_name := 'docs_lab_target',
    dsn := 'host=target-instance.aws-region.rds.amazonaws.com port=5432
 sslmode=require dbname=labdb user=postgres password=********');
 create_node
-------------
   2182738256
(1 row)
```

5. サブスクリプションを作成します。

```
SELECT pglogical.create_subscription(
   subscription_name := 'docs_lab_subscription',
   provider_dsn := 'host=source-instance.aws-region.rds.amazonaws.com port=5432
 sslmode=require dbname=labdb user=postgres password=*******',
   replication_sets := ARRAY['default'],
   synchronize_data := true,
   forward_origins := '{}' );
 create_subscription
---------------------
```

```
1038357190
(1 row)
```

このステップを完了すると、パブリッシャーのテーブルのデータが、サブスクライバーのテーブルに作成されます。このことを確認するには、次の SQL クエリを使用します。

```
SELECT count(*) FROM docs_lab_table;
 count
-------
  5000
(1 row)
```

これ以降、パブリッシャーのテーブルに加えられた変更は、サブスクライバーのテーブルにレプリケートされます。

メジャーアップグレード後の論理レプリケーションの再確立

論理レプリケーションのパブリッシャーノードとして設定されている Aurora PostgreSQL DB クラスターのメジャーバージョンアップグレードを実行する前に、アクティブではないものを含め、すべてのレプリケーションスロットを削除する必要があります。パブリッシャーノードからデータベーストランザクションを一時的に迂回させ、レプリケーションスロットを削除し、Aurora PostgreSQL DB クラスターをアップグレードしてから、レプリケーションを再確立して再開することをお勧めします。

レプリケーションスロットはパブリッシャーノードでのみホストされます。論理レプリケーションシナリオの Aurora PostgreSQL サブスクライバーノードには、削除するスロットがありません。Aurora PostgreSQL のメジャーバージョンアップグレードプロセスでは、パブリッシャーノードとは関係なく、サブスクライバーを新しいメジャーバージョンの PostgreSQL にアップグレードできます。ただし、アップグレードプロセスによってレプリケーションプロセスが中断され、パブリッシャーノードとサブスクライバーノード間の WAL データの同期が妨げられます。パブリッシャーまたはサブスクライバー、あるいはその両方をアップグレードした後に、パブリッシャーとサブスクライバー間の論理レプリケーションを再確立する必要があります。以下の手順は、レプリケーションが中断されたことを確認する方法と問題を解決する方法を示しています。

論理レプリケーションが中断されたことの確認

次のように、パブリッシャーノードまたはサブスクライバーノードのいずれかにクエリを実行することで、レプリケーションプロセスが中断されたことを確認できます。

パブリッシャーノードを確認するには

- psql を使用してパブリッシャーノードに接続して、pg_replication_slots 関数をクエリします。active 列の値に注目します。通常は t (true) が返されます。これは、レプリケーションがアクティブであることを示します。クエリが f (false) を返す場合は、サブスクライバーへのレプリケーションが停止したことを示します。

```
SELECT slot_name,plugin,slot_type,active FROM pg_replication_slots;
                    slot_name              |      plugin      | slot_type | active
-------------------------------------------+------------------+-----------+--------
 pgl_labdb_docs_labcb4fa94_docs_lab3de412c | pglogical_output | logical   | f
(1 row)
```

サブスクライバーノードを確認するには

サブスクライバーノードでは、3 つの異なる方法でレプリケーションのステータスを確認できます。

- サブスクライバーノードの PostgreSQL ログを調べて、失敗のメッセージを見つけます。ログでは、次に示すように、終了コード 1 を含むメッセージで失敗が識別されます。

```
2022-07-06 16:17:03 UTC::@:[7361]:LOG: background worker "pglogical apply
 16404:2880255011" (PID 14610) exited with exit code 1
2022-07-06 16:19:44 UTC::@:[7361]:LOG: background worker "pglogical apply
 16404:2880255011" (PID 21783) exited with exit code 1
```

- pg_replication_origin 関数をクエリします。次のように、psql を使用してサブスクライバーノード上のデータベースに接続し、pg_replication_origin 関数をクエリします。

```
SELECT * FROM pg_replication_origin;
 roident | roname
---------+--------
(0 rows)
```

結果セットが空の場合は、レプリケーションが中断されたことを意味します。通常、次のような出力が表示されます。

```
  roident |                       roname
 ---------+-------------------------------------------------
        1 | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
```

```
  (1 row)
```

- 次の例に示すように、pglogical.show_subscription_status 関数をクエリします。

```
SELECT subscription_name,status,slot_name FROM pglogical.show_subscription_status();
     subscription_name | status |              slot_name
-------+----------------+--------+-----------------------------------
 docs_lab_subscription | down   | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
(1 row)
```

この出力は、レプリケーションが中断されたことを示しています。そのステータスは down です。通常、出力にはステータスが replicating として表示されます。

論理レプリケーションプロセスが中断された場合は、次のステップに従ってレプリケーションを再確立できます。

パブリッシャーノードとサブスクライバーノード間の論理レプリケーションを再確立するには

レプリケーションを再確立するには、以下のステップで説明するように、まずサブスクライバーをパブリッシャーノードから切断し、次にサブスクリプションを再確立します。

1. 次のように psql を使用してサブスクライバーノードに接続します。

```
psql --host=222222222222.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

2. pglogical.alter_subscription_disable 関数を使用してサブスクリプションを非アクティブ化します。

```
SELECT pglogical.alter_subscription_disable('docs_lab_subscription',true);
 alter_subscription_disable
----------------------------
 t
(1 row)
```

3. 以下のように、pg_replication_origin をクエリして、パブリッシャーノードの識別子を取得します。

```
SELECT * FROM pg_replication_origin;
 roident |               roname
---------+--------------------------------------
```

```
       1 | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
(1 row)
```

4. 前のステップからの応答を pg_replication_origin_create コマンドに使用して、サブスクリプションが再確立されたときに使用できる識別子を割り当てます。

```
SELECT pg_replication_origin_create('pgl_labdb_docs_labcb4fa94_docs_lab3de412c');
  pg_replication_origin_create
------------------------------
                             1
(1 row)
```

5. 次の例のように、ステータスを true にして名前を渡し、サブスクリプションを有効にします。

```
SELECT pglogical.alter_subscription_enable('docs_lab_subscription',true);
  alter_subscription_enable
---------------------------
 t
(1 row)
```

ノードのステータスを確認します。ステータスはこの例のように replicating として表示されているはずです。

```
SELECT subscription_name,status,slot_name
  FROM pglogical.show_subscription_status();
            subscription_name |   status    |              slot_name
------------------------------+-------------+-------------------------------------
 docs_lab_subscription         | replicating |
 pgl_labdb_docs_lab98f517b_docs_lab3de412c
(1 row)
```

パブリッシャーノード上のサブスクライバーのレプリケーションスロットのステータスを確認します。スロットの active 列は t (true) を返し、レプリケーションが再確立されたことを示します。

```
SELECT slot_name,plugin,slot_type,active
  FROM pg_replication_slots;
                   slot_name              |      plugin      | slot_type | active
------------------------------------------+------------------+-----------+--------
 pgl_labdb_docs_lab98f517b_docs_lab3de412c | pglogical_output | logical   | t
(1 row)
```

Aurora PostgreSQL 用ロジカルレプリケーションスロットの管理

論理レプリケーションシナリオでパブリッシャーノードとして機能している Aurora PostgreSQL DB クラスターのライターインスタンスでメジャーバージョンアップグレードを実行する前に、インスタンスのレプリケーションスロットを削除する必要があります。メジャーバージョンアップグレードの事前確認プロセスにより、スロットが削除されるまでアップグレードを続行できないことが通知されます。

pglogical 拡張を使用して作成されたレプリケーションスロットを特定するには、各データベースにログインしてノードの名前を取得します。サブスクライバーノードにクエリを実行すると、次の例に示すように、パブリッシャーノードとサブスクライバーノードの両方が出力されます。

```
SELECT * FROM pglogical.node;
node_id   |     node_name
------------+-------------------
 2182738256 | docs_lab_target
 3410995529 | docs_lab_provider
(2 rows)
```

次のクエリで、サブスクリプションの詳細を取得できます。

```
SELECT sub_name,sub_slot_name,sub_target
  FROM pglogical.subscription;
 sub_name |         sub_slot_name          | sub_target
----------+--------------------------------+------------
  docs_lab_subscription     | pgl_labdb_docs_labcb4fa94_docs_lab3de412c | 2182738256
(1 row)
```

これで、次のようにサブスクリプションを削除できます。

```
SELECT pglogical.drop_subscription(subscription_name := 'docs_lab_subscription');
 drop_subscription
-------------------
                 1
(1 row)
```

サブスクリプションを削除すると、ノードを削除できます。

```
SELECT pglogical.drop_node(node_name := 'docs-lab-subscriber');
 drop_node
-----------
```

```
 t
(1 row)
```

次のように、ノードが存在しないことを確認できます。

```
SELECT * FROM pglogical.node;
 node_id | node_name
---------+-----------
(0 rows)
```

pglogical 拡張のパラメータリファレンス

表には、pglogical 拡張に関連するパラメータがあります。pglogical.conflict_log_level や pglogical.conflict_resolution などのパラメータは、更新の競合を処理するために使用されます。パブリッシャーから変更をサブスクライブしているテーブルにローカルで変更を加えると、競合が発生する可能性があります。これ以外にも、競合は、双方向のレプリケーションや、複数のサブスクライバーが同じパブリッシャーからレプリケートする場合など、さまざまなシナリオで発生する可能性があります。詳細については、「PostgreSQL bi-directional replication using pglogical」(pglogical を使用した PostgreSQL の双方向レプリケーション) を参照してください。

パラメータ	説明
pglogical.batch_inserts	可能であれば、バッチ挿入。デフォルトでは設定されていません。オンにする場合は「1」に、オフにする場合は「0」に変更します。
pglogical.conflict_log_level	解決された競合のログ記録に使用するログレベルを設定します。サポートされている文字列値は、debug5、debug4、debug3、debug2、debug1、info、notice、warning、error、log、fatal、panic です。
pglogical.conflict_resolution	競合が解決可能な場合に競合を解決するために使用するメソッドを設定します。サポートされている文字列値は、error、apply_remote、keep_local、last_update_wins、first_update_wins です。
pglogical.extra_connection_options	すべてのピアノード接続に追加する接続オプション。

パラメータ	説明
pglogical.synchronous_commit	pglogical 固有の同期コミット値
pglogical.use_spi	低レベル API の代わりに SPI (サーバープログラミングインターフェイス) を使用して変更を適用します。オンにする場合は「1」に、オフにする場合は「0」に設定します。SPI の詳細については、PostgreSQL ドキュメントの「サーバープログラミングインターフェイス」を参照してください。

Amazon Aurora PostgreSQL でサポートされている外部データラッパーを使用する

外部データラッパー (FDW) は、外部データへのアクセスを提供する特定のタイプの拡張機能です。例えば、`oracle_fdw` 拡張機能を使用すると、Aurora PostgreSQL DB インスタンス が Oracle データベースと連動できるようになります。

以下で、PostgreSQL でサポートされている、いくつかの 外部データラッパーについての情報を確認できます。

トピック

- SQL を使用した DB ログのアクセスのための log_fdw 拡張機能の使用
- 外部データへのアクセスのための postgres_fdw 拡張機能の使用
- mysql_fdw 拡張機能による MySQL データベースの操作
- oracle_fdw 拡張機能による Oracle データベースの操作
- tds_fdw 拡張機能による SQL Server データベースの操作

SQL を使用した DB ログのアクセスのための log_fdw 拡張機能の使用

Aurora PostgreSQL DB クラスター は、SQL インターフェイスを通じてデータベースエンジンのログにアクセスする際に使用できる、`log_fdw` 拡張機能をサポートしています。`log_fdw` エクステンションは、データベースログ用の外部テーブルの作成を容易にする 2 つの関数を提供します。

- `list_postgres_log_files` - データベースログディレクトリのファイルとファイルサイズ (バイト単位) を一覧表示します。

- `create_foreign_table_for_log_file(table_name text, server_name text, log_file_name text)` - 現在のデータベースで指定されたファイルの外部テーブルを構築します。

`log_fdw` によって作成されたすべての関数は、`rds_superuser` によって所有されます。`rds_superuser` ロールのメンバーは、これらの関数へのアクセス権限を他のデータベースユーザーに付与することができます。

デフォルトでは、ログファイルは、`log_destination` パラメータで指定されたように、Amazon Aurora によって `stderr` (標準エラー) 形式で生成されます。このパラメータには、`stderr` と `csvlog` (カンマ区切り値、CSV) の 2 つのオプションしかありません。パラメータに `csvlog` オプションを追加すると、Amazon Aurora は `stderr` と `csvlog` 両方のログを生成します。これは DB クラスターのストレージ容量に影響を与える可能性があるため、ログ処理に影響を与える他のパラメータに注意する必要があります。詳細については、「ログの送信先の設定 (stderr、csvlog)」を参照してください。

`csvlog` ログを生成すること 1 つの利点は、`log_fdw` 拡張機能により、データが複数の列にきちんと分割された外部テーブルを構築できることです。これを行うには、インスタンスをカスタム DB パラメータグループに関連付けて、`log_destination` の設定を変更できるようにする必要があります。これを行う方法については、「「パラメータグループを使用する」」を参照してください。

次の例では、`log_destination` パラメータに `cvslog` が含まれることを前提としています。

log_fdw 拡張を使用するには

1. `log_fdw` 拡張機能をインストールします。

```
postgres=> CREATE EXTENSION log_fdw;
CREATE EXTENSION
```

2. 外部データラッパーとしてログサーバーを作成します。

```
postgres=> CREATE SERVER log_server FOREIGN DATA WRAPPER log_fdw;
CREATE SERVER
```

3. ログファイルのリストからすべてを選択します。

```
postgres=> SELECT * FROM list_postgres_log_files() ORDER BY 1;
```

レスポンスの例を次に示します。

```
          file_name           | file_size_bytes
------------------------------+-----------------
 postgresql.log.2023-08-09-22.csv |            1111
 postgresql.log.2023-08-09-23.csv |            1172
 postgresql.log.2023-08-10-00.csv |            1744
 postgresql.log.2023-08-10-01.csv |            1102
(4 rows)
```

4. 選択したファイルの、1 つの 'log_entry' 列でテーブルを作成します。

```
postgres=> SELECT create_foreign_table_for_log_file('my_postgres_error_log',
     'log_server', 'postgresql.log.2023-08-09-22.csv');
```

レスポンスでは、テーブルが存在しているということ以外の詳細を返しません。

```
-----------------------------------
(1 row)
```

5. ログファイルのサンプルを選択します。次のコードは、ログの時間とエラーメッセージの説明を取得します。

```
postgres=> SELECT log_time, message FROM my_postgres_error_log ORDER BY 1;
```

レスポンスの例を次に示します。

```
            log_time             |                                 message
---------------------------------
+--------------------------------------------------------------------------------
Tue Aug 09 15:45:18.172 2023 PDT | ending log output to stderr
Tue Aug 09 15:45:18.175 2023 PDT | database system was interrupted; last known up
 at 2023-08-09 22:43:34 UTC
Tue Aug 09 15:45:18.223 2023 PDT | checkpoint record is at 0/90002E0
Tue Aug 09 15:45:18.223 2023 PDT | redo record is at 0/90002A8; shutdown FALSE
Tue Aug 09 15:45:18.223 2023 PDT | next transaction ID: 0/1879; next OID: 24578
Tue Aug 09 15:45:18.223 2023 PDT | next MultiXactId: 1; next MultiXactOffset: 0
Tue Aug 09 15:45:18.223 2023 PDT | oldest unfrozen transaction ID: 1822, in
 database 1
(7 rows)
```

外部データへのアクセスのための postgres_fdw 拡張機能の使用

postgres_fdw 拡張を使用してリモートデータベースサーバーにあるテーブルのデータにアクセスできます。PostgreSQL DB インスタンスからリモート接続を設定すると、リードレプリカにもアクセスできます。

postgres_fdw を使用してリモートデータベースサーバーにアクセスするには

1. postgres_fdw 拡張をインストールします。

```
CREATE EXTENSION postgres_fdw;
```

2. CREATE SERVER を使用して外部データサーバーを作成します。

```
CREATE SERVER foreign_server
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host 'xxx.xx.xxx.xx', port '5432', dbname 'foreign_db');
```

3. リモートサーバーで使用するロールを識別するためのユーザーマッピングを作成します。

```
CREATE USER MAPPING FOR local_user
SERVER foreign_server
OPTIONS (user 'foreign_user', password 'password');
```

4. リモートサーバーのテーブルにマッピングするテーブルを作成します。

```
CREATE FOREIGN TABLE foreign_table (
        id integer NOT NULL,
        data text)
SERVER foreign_server
OPTIONS (schema_name 'some_schema', table_name 'some_table');
```

mysql_fdw 拡張機能による MySQL データベースの操作

Aurora PostgreSQL DB クラスター から MySQL 互換データベースにアクセスするには、mysql_fdw 拡張機能をインストールしそれを使用します。この外部データラッパーを使用すると、RDS for MySQL、Aurora MySQL、MariaDB、その他の MySQL 互換データベースを操作できます。Aurora PostgreSQL DB クラスター から MySQL データベースへの接続は、クライアントとサーバーの設定に応じて、ベストエフォートベースで暗号化されます。ただし、必要に応じて暗号化を強制できます。詳細については、「拡張機能で転送中の暗号化を使用する」を参照してください。

mysql_fdw 拡張機能は、Amazon Aurora PostgreSQL バージョン 15.4、14.9、13.12、12.16、 以降のリリースでサポートされています。MySQL 互換データベースインスタンス上のテーブルに対する RDS for PostgreSQL DB での選択、挿入、更新、および削除をサポートします。

トピック

- mysql_fdw 拡張機能を使用するように Aurora PostgreSQL DB をセットアップする
- 例: Aurora PostgreSQL から Aurora MySQL データベースを操作する
- 拡張機能で転送中の暗号化を使用する

mysql_fdw 拡張機能を使用するように Aurora PostgreSQL DB をセットアップする

Aurora PostgreSQL DB クラスター での mysql_fdw 拡張機能のセットアップには、DB クラスターでの拡張機能のロードと、MySQL DB インスタンスへの接続ポイントの作成が関係しています。このタスクでは、MySQL DB インスタンスに関する次の詳細が必要です。

- ホスト名またはエンドポイント。Aurora MySQL DB クラスター の場合、コンソールを使用してエンドポイントを見つけることができます。[接続とセキュリティ] タブを選択し、[エンドポイントとポート] セクションを確認します。
- ポート番号。MySQL のデフォルトポート番号は 3306 です。
- データベースの名前 DB 識別子。

また、MySQL ポート 3306 のセキュリティグループまたはアクセスコントロールリスト (ACL) へのアクセスを提供する必要があります。Aurora PostgreSQL DB クラスターと Aurora MySQL DB クラスター の両方がポート 3306 にアクセスする必要があります。アクセスが正しく設定されていない場合、MySQL 互換テーブルに接続しようとすると、次のようなエラーメッセージが表示されます。

```
ERROR: failed to connect to MySQL: Can't connect to MySQL server on 'hostname.aws-
region.rds.amazonaws.com:3306' (110)
```

次の手順では、ユーザーが (rds_superuser アカウントとして) 外部サーバーを作成します。次に、外部サーバーへのアクセスを特定のユーザーに付与します。その後、これらのユーザーは、MySQL DB インスタンスを操作するための適切な MySQL ユーザーアカウントへの独自のマッピングを作成します。

mysql_fdw を使用して MySQL データベースサーバーにアクセスするには

1. rds_superuser ロールがあるアカウントを使用して PostgreSQL DB インスタンスを接続します。Aurora PostgreSQL DB クラスター の作成時にデフォルトを受け入れた場合、ユーザー名は postgres であり、psql コマンドラインツールを使用して次のように接続できます。

```
psql --host=your-DB-instance.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password
```

2. 次のように mysql_fdw 拡張機能をインストールします。

```
postgres=> CREATE EXTENSION mysql_fdw;
CREATE EXTENSION
```

拡張機能が Aurora PostgreSQL DB クラスターにインストールされたら、MySQL データベースへの接続を提供する外部サーバーをセットアップします。

外部サーバーを作成するには

Aurora PostgreSQL DB クラスター でこれらのタスクを実行します。このステップは、rds_superuser特権 (postgres など) があるユーザーとして接続していることを前提としています。

1. Aurora PostgreSQL DB クラスターで外部サーバーを作成します。

```
postgres=> CREATE SERVER mysql-db FOREIGN DATA WRAPPER mysql_fdw OPTIONS (host 'db-
name.111122223333.aws-region.rds.amazonaws.com', port '3306');
CREATE SERVER
```

2. 適切なユーザーに外部サーバーへのアクセスを付与します。これらは、管理者以外のユーザー、つまり、rds_superuser ロールのないユーザーである必要があります。

```
postgres=> GRANT USAGE ON FOREIGN SERVER mysql-db to user1;
GRANT
```

PostgreSQL ユーザーは、外部サーバーを介して MySQL データベースへの独自の接続を作成し、管理します。

例: Aurora PostgreSQL から Aurora MySQL データベースを操作する

Aurora PostgreSQL DB インスタンスにシンプルなテーブルがあると仮定します。Aurora PostgreSQL ユーザーが、そのテーブルで (SELECT)、INSERT、UPDATE、DELETE の項目をクエリしたいと思っています。mysql_fdw 拡張機能は、前の手順で詳述されているように、RDS for PostgreSQL DB インスタンスで作成された、と仮定します。rds_superuser 権限のあるユーザーとして RDS for PostgreSQL DB インスタンスに接続した後、次の手順に進むことができます。

1. Aurora PostgreSQL DB インスタンスで外部サーバーを作成します。

```
test=> CREATE SERVER mysqldb FOREIGN DATA WRAPPER mysql_fdw OPTIONS (host 'your-
DB.aws-region.rds.amazonaws.com', port '3306');
CREATE SERVER
```

2. rds_superuser の許可を持たないユーザーに、(例えば user1 として) 使用を許可します。

```
test=> GRANT USAGE ON FOREIGN SERVER mysqldb TO user1;
GRANT
```

3. *user1* として接続し、MySQL ユーザーへのマッピングを作成します。

```
test=> CREATE USER MAPPING FOR user1 SERVER mysqldb OPTIONS (username 'myuser',
 password 'mypassword');
CREATE USER MAPPING
```

4. MySQL テーブルにリンクされた外部テーブルを作成します。

```
test=> CREATE FOREIGN TABLE mytab (a int, b text) SERVER mysqldb OPTIONS (dbname
 'test', table_name '');
CREATE FOREIGN TABLE
```

5. 外部テーブルに対して単純なクエリを実行します。

```
test=> SELECT * FROM mytab;
a |   b
---+-------
1 | apple
(1 row)
```

6. MySQL テーブルでのデータの追加、変更、削除を行うことができます。例:

```
test=> INSERT INTO mytab values (2, 'mango');
```

```
INSERT 0 1
```

SELECT クエリをもう一度実行して、結果を確認します。

```
test=> SELECT * FROM mytab ORDER BY 1;
 a |   b
---+-------
1 | apple
2 | mango
(2 rows)
```

拡張機能で転送中の暗号化を使用する

Aurora PostgreSQL から MySQL への接続は、デフォルトで転送中の暗号化 (TLS/SSL) を使用します。ただし、クライアントとサーバーの設定が異なる場合、接続は暗号化されていない状態に戻ります。RDS for MySQL ユーザアカウントの REQUIRE SSL オプションを指定して、すべての発信接続に対して暗号化を適用できます。この同じアプローチは MariaDB および Aurora MySQL ユーザーアカウントでも機能します。

REQUIRE SSL に構成された MySQL ユーザーアカウントの場合、安全な接続を確立できないと接続の試行は失敗します。

既存の MySQL データベースユーザーアカウントの暗号化を強制するには、ALTER USER コマンドを使用できます。次の表に示すとおり、構文は MySQL のバージョンによって異なります。詳細については、MySQL リファレンスマニュアルの ALTER USER を参照してください。

MySQL 5.7、MySQL 8.0	MySQL 5.6
`ALTER USER 'user'@'%' REQUIRE SSL;`	`GRANT USAGE ON *.* to 'user'@'%' REQUIRE SSL;`

mysql_fdw 拡張機能の詳細については、mysql_fdw ドキュメントをご覧ください。

oracle_fdw 拡張機能による Oracle データベースの操作

Aurora PostgreSQL DB クラスター から Oracle データベースにアクセスするには、oracle_fdw 拡張機能をインストールして、使用します。この拡張機能は、Oracle データベース用の外部データラッパーです。この拡張機能の詳細については、oracle_fdw のドキュメントを参照してください。

oracle_fdw 拡張機能は、PostgreSQL 12.7 (Amazon Aurora release 4.2) 以上のバージョンでサポートされています。

トピック

oracle_fdw 拡張機能の有効化

oracle_fdw 拡張機能を使用するには、以下の手順を実行します。

oracle_fdw 拡張機能を有効化するには

- rds_superuser のアクセス許可を持つアカウントを使用して、次のコマンドを実行します。

```
CREATE EXTENSION oracle_fdw;
```

例: Amazon RDS for Oracle Database にリンクされた外部サーバーの使用

以下は、Amazon RDS for Oracle のデータベースにリンクされた外部サーバーの使用例です。

RDS for Oracle データベースにリンクされた外部サーバーを作成するには

1. RDS for Oracle DB インスタンスの以下の点を書き留めます。
   - エンドポイント
   - ポート
   - データベース名
2. 外部サーバーを作成します。

```
test=> CREATE SERVER oradb FOREIGN DATA WRAPPER oracle_fdw OPTIONS (dbserver
 '//endpoint:port/DB_name');
CREATE SERVER
```

3. rds_superuser の権限を持たないユーザーに、(例えば user1 として) 使用を許可します。

```
test=> GRANT USAGE ON FOREIGN SERVER oradb TO user1;
GRANT
```

4. user1 として接続し、Oracle ユーザーへのマッピングを作成します。

```
test=> CREATE USER MAPPING FOR user1 SERVER oradb OPTIONS (user 'oracleuser',
 password 'mypassword');
CREATE USER MAPPING
```

5. Oracle テーブルにリンクされた外部テーブルを作成します。

```
test=> CREATE FOREIGN TABLE mytab (a int) SERVER oradb OPTIONS (table 'MYTABLE');
CREATE FOREIGN TABLE
```

6. 外部テーブルに対しクエリを実行します。

```
test=>  SELECT * FROM mytab;
a
---
1
(1 row)
```

クエリで次のエラーが報告された場合は、セキュリティグループとアクセスコントロールリストをチェックして、両方のインスタンス間で通信が可能なことを確認します。

```
ERROR: connection for foreign table "mytab" cannot be established
DETAIL: ORA-12170: TNS:Connect timeout occurred
```

転送時の暗号化の使用

PostgreSQL から Oracle への転送時における暗号化は、クライアントとサーバーの設定パラメータの組み合わせに基づき構成されます。Oracle 21c の使用例については、Oracle ドキュメントの「About the Values for Negotiating Encryption and Integrity」を参照してください。Amazon RDS で oracle_fdw 用に使用されるクライアントは、ACCEPTED に設定されています。つまり、暗号化は Oracle データベースサーバーの設定に依存します。

データベースが RDS for Oracle 上にある場合の暗号化の設定については、「Oracle ネイティブネットワーク暗号化」を参照してください。

pg_user_mappings のビューおよび許可を理解する

PostgreSQL カタログ pg_user_mapping は、Aurora PostgreSQL ユーザーからのマッピングを外部データ (リモート) サーバー上のユーザーに保存します。カタログへのアクセスは制限されていますが、pg_user_mappings ビューをクリックすると、マッピングが表示されます。以下に、Oracle データベースの例で許可がどのように適用されるかを示す例がありますが、この情報は一般的に外部データラッパーに適用されます。

次の出力では、ロールとアクセス許可が、3 つの異なるサンプルユーザーにマップされていることが示されています。ここで、ユーザー rdssu1 と rdssu2 は rds_superuser ロールのメンバーであり、user1 はメンバーではありません。この例では、psql メタコマンド \du を使用して、既存のロールを一覧表示します。

```
test=>  \du
                                                          List of roles
    Role name    |                          Attributes                          |
                 Member of
-----------------+------------------------------------------------------------
+-------------------------------------------------------
 rdssu1          |                                                              |
 {rds_superuser}
 rdssu2          |                                                              |
 {rds_superuser}
 user1           |                                                              | {}
```

すべてのユーザー (rds_superuser 権限を持っているユーザーを含む) は、pg_user_mappings テーブルで独自のユーザーマッピング (umoptions) を表示することが許可されています。次の例に示すように、rdssu1 がすべてのユーザーマッピングを取得しようとすると、rdssu1rds_superuser 権限があっても、次のエラーが発生します。

```
test=> SELECT * FROM pg_user_mapping;
ERROR: permission denied for table pg_user_mapping
```

次に例をいくつか示します。

```
test=> SET SESSION AUTHORIZATION rdssu1;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |            umoptions
-------+-------+---------+--------+------------+---------------------------------
```

```
 16414 | 16411 | oradb   |  16412 | user1      |
 16423 | 16411 | oradb   |  16421 | rdssu1     | {user=oracleuser,password=mypwd}
 16424 | 16411 | oradb   |  16422 | rdssu2     |
 (3 rows)

test=> SET SESSION AUTHORIZATION rdssu2;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |            umoptions
-------+-------+---------+--------+------------+---------------------------------
 16414 | 16411 | oradb   |  16412 | user1      |
 16423 | 16411 | oradb   |  16421 | rdssu1     |
 16424 | 16411 | oradb   |  16422 | rdssu2     | {user=oracleuser,password=mypwd}
 (3 rows)

test=> SET SESSION AUTHORIZATION user1;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |           umoptions
-------+-------+---------+--------+------------+-------------------------------
 16414 | 16411 | oradb   |  16412 | user1      | {user=oracleuser,password=mypwd}
 16423 | 16411 | oradb   |  16421 | rdssu1     |
 16424 | 16411 | oradb   |  16422 | rdssu2     |
 (3 rows)
```

information_schema._pg_user_mappings と pg_catalog.pg_user_mappings の間に実装上の違いがあるため、手動で作成された rds_superuser が pg_catalog.pg_user_mappings 内のパスワードを表示する場合には、追加のアクセス許可が必要となります。

rds_superuser が information_schema._pg_user_mappings 内のパスワードを表示する際には、追加のアクセス許可は必要ありません。

rds_superuser ロールを持たないユーザーの場合、以下の条件の下でのみ、pg_user_mappings 内のパスワードを表示できます。

- 現在のユーザーはマップされているユーザーであり、サーバーの所有者であるか、そのサーバーに対する USAGE 権限を保持しています。
- 現在のユーザーはサーバーの所有者であり、マッピングは PUBLIC となっています。

tds_fdw 拡張機能による SQL Server データベースの操作

PostgreSQL `tds_fdw` 拡張機能を使用して、Sybase や Microsoft SQL Server データベースなど、表形式データストリーム (TDS) プロトコルをサポートするデータベースにアクセスできます。この外部データラッパーを使用すると、 Aurora PostgreSQL DB クラスターを、Amazon RDS for Microsoft SQL Server を含む、TDS プロトコルを使用するデータベースに接続できます。詳細については、GitHub にある tds-fdw/tds_fdw に関するドキュメントを参照してください。

`tds_fdw` 拡張機能は、Amazon Aurora PostgreSQL バージョン 13.6 以降でサポートされています。

tds_fdw 拡張機能を使用するように Aurora PostgreSQL DB をセットアップする

次の手順では、`tds_fdw` をセットアップして、Aurora PostgreSQL DB クラスターと使用する例を示します。`tds_fdw` を使用して SQL Server データベースに接続する前に、インスタンスの次の詳細を取得する必要があります。

- ホスト名またはエンドポイント。RDS for SQL Server DB インスタンスの場合、コンソールを使用してエンドポイントを見つけることができます。[Connectivity & security] (接続とセキュリティ) タブを選択し、[Endpoint and port] (エンドポイントとポート) セクションを確認します。
- ポート番号。Microsoft SQL Server のデフォルトポート番号は 1433 です。
- データベースの名前 DB 識別子。

また、SQL Server ポート、1433 のセキュリティグループまたはアクセスコントロールリスト (ACL) でのアクセスを提供する必要があります。Aurora PostgreSQL DB クラスターと RDS for SQL Server DB インスタンスの両方が、ポート 1433 にアクセスする必要があります。アクセスが正しく設定されていない場合、Microsoft SQL Server をクエリしようとすると、次のエラーメッセージが表示されます。

```
ERROR: DB-Library error: DB #: 20009, DB Msg: Unable to connect:
Adaptive Server is unavailable or does not exist (mssql2019.aws-
region.rds.amazonaws.com), OS #: 0, OS Msg: Success, Level: 9
```

tds_fdw を使用して SQL Server データベースに接続するには

1. `rds_superuser` ロールがあるアカウントを使用して、Aurora PostgreSQL DB クラスターのプライマリインスタンスに接続します。

```
psql --host=your-cluster-name-instance-1.aws-region.rds.amazonaws.com --port=5432
 --username=test --password
```

2. tds_fdw 拡張機能をインストールします。

```
test=> CREATE EXTENSION tds_fdw;
CREATE EXTENSION
```

Aurora PostgreSQL DB クラスター に拡張機能をインストールした後、外部サーバーをセットアップします。

外部サーバーを作成するには

rds_superuser 権限があるアカウントを使用する Aurora PostgreSQL DB クラスターで次のタスクを実行します。

1. Aurora PostgreSQL DB クラスターで外部サーバーを作成します。

```
test=> CREATE SERVER sqlserverdb FOREIGN DATA WRAPPER tds_fdw OPTIONS
 (servername 'mssql2019.aws-region.rds.amazonaws.com', port '1433', database
 'tds_fdw_testing');
CREATE SERVER
```

   SQLServer 側で非 ASCII データにアクセスするには、Aurora PostgreSQL DB クラスターの character_set オプションを使用してサーバーリンクを作成します。

```
test=> CREATE SERVER sqlserverdb FOREIGN DATA WRAPPER tds_fdw OPTIONS (servername
 'mssql2019.aws-region.rds.amazonaws.com', port '1433', database 'tds_fdw_testing',
 character_set 'UTF-8');
CREATE SERVER
```

2. rds_superuser ロール権限を持たないユーザーに、(例えば user1 として) 許可を付与します。

```
test=> GRANT USAGE ON FOREIGN SERVER sqlserverdb TO user1;
```

3. user1 として接続し、SQL Server ユーザーへのマッピングを作成します。

```
test=> CREATE USER MAPPING FOR user1 SERVER sqlserverdb OPTIONS (username
 'sqlserveruser', password 'password');
CREATE USER MAPPING
```

4. SQL Server テーブルにリンクされた外部テーブルを作成します。

```
test=> CREATE FOREIGN TABLE mytab (a int) SERVER sqlserverdb OPTIONS (table
 'MYTABLE');
CREATE FOREIGN TABLE
```

5. 外部テーブルに対しクエリを実行します。

```
test=> SELECT * FROM mytab;
 a
---
 1
(1 row)
```

接続に転送中の暗号化を使用する

Aurora PostgreSQL から SQL Server への接続には、SQL Server のデータベース設定に応じて、転送中の暗号化 (TLS/SSL) を使用します。SQL Server が暗号化用に設定されていない場合、SQL Server データベースへの要求を行う RDS for PostgreSQL クライアントは、暗号化されていない状態に戻ります。

`rds.force_ssl` パラメータを設定して、RDS for SQL Server DB インスタンスへの接続に暗号化を強制できます。この方法については、「DB インスタンスへの接続に SSL を使用させる」を参照してください。RDS for SQL Server での SSL/TLS 設定の詳細については、「Microsoft SQL Server DB インスタンスでの SSL の使用」を参照してください。

Trusted Language Extensions for PostgreSQL を使用した操作

Trusted Language Extensions for PostgreSQL は PostgreSQL 拡張機能を構築するためのオープンソース開発キットです。これにより、高性能の PostgreSQL 拡張機能を構築し、それらを Aurora PostgreSQL DB クラスター上で安全に実行できます。PostgreSQL の Trusted Language Extensions (TLE) を使用することで、PostgreSQL の機能を拡張する文書化されたアプローチに従った PostgreSQL 拡張機能を作成できます。詳細については、PostgreSQL ドキュメントの「エクステンションへの関連オブジェクトのパッケージ化」を参照してください。

TLE の主な利点の 1 つは、PostgreSQL インスタンスの基盤となるファイルシステムへのアクセスを提供しない環境で使用できることです。以前は、新しい拡張機能をインストールするにはファイルシステムへのアクセスが必要でした。TLE ではこの制約がありません。Aurora PostgreSQL DB クラスターで実行されているものを含め、あらゆる PostgreSQL データベース用の新しい拡張機能を作成するための開発環境を提供します。

TLE は、TLE を使用して作成する拡張機能の危険なリソースへのアクセスを防ぐように設計されています。そのランタイム環境では、拡張機能の不具合による影響は 1 つのデータベース接続に限定されます。また、TLE では、データベース管理者が拡張機能をインストールできるユーザーをきめ細かく制御でき、拡張機能を実行するためのアクセス許可モデルも用意されています。

TLE は、Aurora PostgreSQL バージョン 14.5 以上のバージョンでサポートされています。

Trusted Language Extensions の開発環境とランタイムは、`pg_tle` PostgreSQL 拡張機能のバージョン 1.0.1 としてパッケージ化されています。JavaScript、Perl、PL/pgSQL、および SQL での拡張機能の作成をサポートしています。他の PostgreSQL 拡張機能をインストールするのと同じ方法で、Aurora PostgreSQL DB クラスター に `pg_tle` 拡張機能をインストールします。`pg_tle` をセットアップすると、開発者はこれを使用して TLE 拡張機能と呼ばれる新しい PostgreSQL 拡張機能を作成できます。

次のトピックでは、Trusted Language Extensions をセットアップする方法と、独自の TLE 拡張機能の作成を開始する方法について説明します。

トピック

- 用語
- Trusted Language Extensions for PostgreSQL を使用するための要件
- Aurora PostgreSQL DB クラスターに Trusted Language Extensions を設定する
- Trusted Language Extensions for PostgreSQL の概要

- Aurora PostgreSQL の TLE 拡張機能の作成
- TLE 拡張機能をデータベースから削除する
- Trusted Language Extensions for PostgreSQL のアンインストール
- TLE 拡張機能で PostgreSQL フックを使用する
- Trusted Language Extensions for PostgreSQL の関数リファレンス
- Trusted Language Extensions for PostgreSQL のフックリファレンス

用語

Trusted Language Extensions の理解を深めるために、このトピックで使用されている用語については、次の用語集を参照してください。

Trusted Language Extensions for PostgreSQL

Trusted Language Extensions for PostgreSQL は、`pg_tle` 拡張機能としてパッケージされているオープンソース開発キットの正式名称です。これは、どの PostgreSQL システムでも使用できます。詳細については、GitHub の「aws/pg_tle」を参照してください。

Trusted Language Extensions

Trusted Language Extensions は、Trusted Language Extensions for PostgreSQL の省略名です。このドキュメントでは、この短縮名とその略称 (TLE) も使用されています。

信頼できる言語

信頼できる言語とは、特定のセキュリティ属性を持つプログラミング言語またはスクリプト言語です。例えば、信頼できる言語は通常、ファイルシステムへのアクセスを制限し、指定されたネットワークプロパティの使用を制限します。TLE 開発キットは、信頼できる言語をサポートするように設計されています。PostgreSQL は、信頼できる、または信頼できない拡張機能を作成するために使用される複数の異なる言語をサポートしています。例については、PostgreSQL ドキュメントの「信頼できる PL/Perl と信頼できない PL/Perl」を参照してください。Trusted Language Extensions を使用して拡張機能を作成すると、その拡張機能は本質的に信頼できる言語メカニズムを使用します。

TLE 拡張機能

TLE 拡張機能は、Trusted Language Extensions (TLE) 開発キットを使用して作成された PostgreSQL 拡張機能です。

Trusted Language Extensions for PostgreSQL を使用するための要件

TLE 開発キットをセットアップして使用するための要件は次のとおりです。

- Aurora PostgreSQL バージョン – Trusted Language Extensions は、Aurora PostgreSQL バージョン 14.5 以降のバージョンでのみサポートされています。
  - Aurora PostgreSQL DB クラスター、をアップグレードする必要がある場合は、「Amazon Aurora PostgreSQL DB クラスターのアップグレード」「」を参照してください。
  - PostgreSQL を実行している Aurora DB クラスター をまだ持っていない場合は作成できます。詳細については、「Aurora PostgreSQL DB クラスターの作成と接続」を参照してください。
- **`rds_superuser`** 権限が必要です - `pg_tle` 拡張機能をセットアップおよび設定するには、データベースユーザーロールに `rds_superuser` ロールのアクセス許可が必要です。デフォルトでは、このロールは Aurora PostgreSQL DB クラスターを作成する `postgres` ユーザーに付与されます。
- カスタム DB パラメータグループが必要です – Aurora PostgreSQL DB クラスター には、カスタム DB パラメータグループを設定する必要があります。Aurora PostgreSQL DB クラスターのライターインスタンスに、カスタム DB パラメータグループを使用します。
  - Aurora PostgreSQL DB クラスター がカスタム DB パラメータグループで構成されていない場合は、カスタム DB パラメータグループを作成して Aurora PostgreSQL DB クラスターのライターインスタンスに関連付ける必要があります。ステップの簡単な概要については、「カスタム DB パラメータグループの作成と適用」を参照してください。
  - Aurora PostgreSQL DB クラスター が、カスタム DB パラメータグループを使用して既に設定されている場合は、Trusted Language Extensions をセットアップできます。詳細については、「Aurora PostgreSQL DB クラスターに Trusted Language Extensions を設定する」を参照してください。

カスタム DB パラメータグループの作成と適用

以下のステップを使用してカスタム DB パラメータグループを作成し、それを使用するように Aurora PostgreSQL DB クラスター を設定します。

コンソール

カスタム DB パラメータグループを作成して、Aurora PostgreSQL DB クラスターで使用するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。

2. Amazon RDS メニューから [Parameter groups] (パラメータグループ) を選択します。
3. [パラメータグループの作成]を選択します。
4. [Parameter group details] (パラメータグループの詳細) ページで、次の情報を入力します。
   - [Parameter group family] (パラメータグループファミリー) で、[aurora-postgresql14.] を選択します
   - [Type] (タイプ) で、[DB Parameter Group] (DB パラメータグループ) を選択します。
   - [Group name] (グループ名) には、パラメータグループに操作の内容に合ったわかりやすい名前を付けます。
   - [Description] (説明) には、チームの他のメンバーが簡単に見つけられるように、わかりやすい説明を入力します。
5. [Create] (作成) を選択します。カスタム DB パラメータグループは AWS リージョン で作成されます。次のステップに従って、Aurora PostgreSQL DB クラスター を使用するように変更できるようになりました。
6. Amazon RDS メニューから [Databases] (データベース) を選択します。
7. 一覧から TLE で使用する Aurora PostgreSQL DB クラスター を選択し、[Modify] (変更) を選択します。
8. DB クラスター設定の変更ページで、[Database options] (データベースオプション) を見つけ、セレクターを使用してカスタム DB パラメータグループを選択します。
9. [Continue] (続行) を選択して、変更を保存します。
10. [Apply immediately] (すぐに適用) を選択すると、引き続き Aurora PostgreSQL DB クラスターを TLE を使用するようにセットアップできます。

Trusted Language Extensions のシステム設定を継続するには、「Aurora PostgreSQL DB クラスターに Trusted Language Extensions を設定する」を参照してください。

DB クラスターと DB パラメータグループの操作の詳細については、「DB クラスターパラメータグループを使用する」を参照してください。

AWS CLI

AWS CLI をデフォルト AWS リージョン に設定することで、CLI コマンドを使用するときに `--region` 引数を指定しなくても済みます。詳細については、AWS Command Line Interface ユーザーガイドの「設定の基本」を参照してください。

カスタム DB パラメータグループを作成して、Aurora PostgreSQL DB クラスターで使用するには

1. create-db-parameter-group AWS CLI コマンドを使用して、AWS リージョン の aurora-postgresql14 をベースにしたカスタム DB パラメータグループを作成してください。このステップでは、Aurora PostgreSQL DB クラスターのライターインスタンスに適用する DB パラメータグループを作成することに注意してください。

   Linux、macOS、Unix の場合:

   ```
   aws rds create-db-parameter-group \
     --region aws-region \
     --db-parameter-group-name custom-params-for-pg-tle \
     --db-parameter-group-family aurora-postgresql14 \
     --description "My custom DB parameter group for Trusted Language Extensions"
   ```

   Windows の場合:

   ```
   aws rds create-db-parameter-group ^
     --region aws-region ^
     --db-parameter-group-name custom-params-for-pg-tle ^
     --db-parameter-group-family aurora-postgresql14 ^
     --description "My custom DB parameter group for Trusted Language Extensions"
   ```

   AWS リージョン でカスタム DB パラメータグループを使用できるため、Aurora PostgreSQL DB クラスター のライターインスタンスを変更してそれを使用できます。
2. modify-db-instance AWS CLI コマンドを使用して、カスタム DB パラメータグループを Aurora PostgreSQL DB クラスターのライターインスタンスに適用します。このコマンドは、アクティブなインスタンスを直ちに再起動します。

   Linux、macOS、Unix の場合:

   ```
   aws rds modify-db-instance \
     --region aws-region \
     --db-instance-identifier your-writer-instance-name \
     --db-parameter-group-name custom-params-for-pg-tle \
     --apply-immediately
   ```

   Windows の場合:

   ```
   aws rds modify-db-instance ^
   ```

```
  --region aws-region ^
  --db-instance-identifier your-writer-instance-name ^
  --db-parameter-group-name custom-params-for-pg-tle ^
  --apply-immediately
```

Trusted Language Extensions のシステム設定を継続するには、「Aurora PostgreSQL DB クラスターに Trusted Language Extensions を設定する」を参照してください。

詳細については、「DB インスタンスでの DB パラメータグループの使用」を参照してください。

Aurora PostgreSQL DB クラスターに Trusted Language Extensions を設定する

以下のステップでは、Aurora PostgreSQL DB クラスター がカスタム DB クラスター パラメータグループに関連付けられていることを前提としています。これらの手順には、AWS Management Console または AWS CLI を使用できます。

Aurora PostgreSQL DB クラスター で信頼できる Trusted Language Extensions をセットアップする場合、そのデータベースに対するアクセス許可を持つデータベースユーザーが使用できるように、特定のデータベースにインストールします。

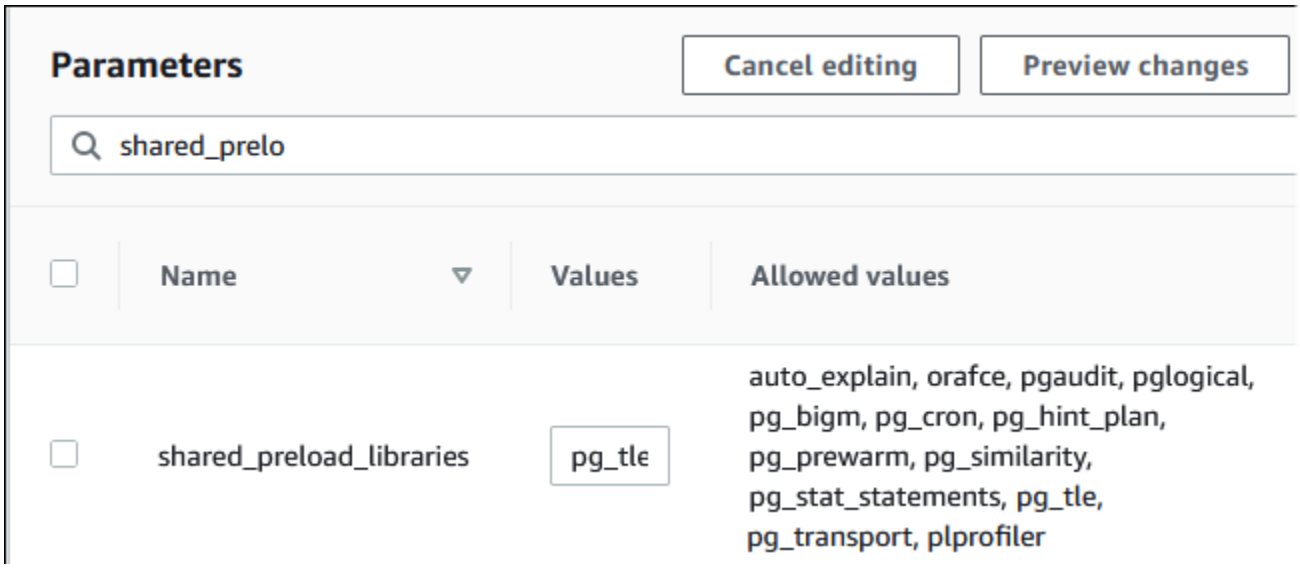
コンソール

Trusted Language Extensions をセットアップするには

`rds_superuser` グループ (ロール) のメンバーであるアカウントを使用して、次のステップを実行します。

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、Aurora PostgreSQL DB クラスターのライターインスタンス を選択します。
3. Aurora PostgreSQL DB クラスターライターインスタンスの [Configuration] (設定) タブを開きます。インスタンスの詳細の中から、パラメータグループのリンクを見つけてください。
4. リンクを選択して、Aurora PostgreSQL DB クラスターに関連するカスタムパラメータを開きます。
5. パラメータ検索フィールドに、`shared_pre` を入力して `shared_preload_libraries` パラメータを検索します。

6. プロパティ値にアクセスするには、[Edit parameters] (パラメータの編集) を選択します。
7. [Values] (値) フィールドのリストに pg_tle を追加します。値のリスト内の項目を区切るにはカンマを使用します。



8. Aurora PostgreSQL DB クラスターのライターインスタンス を再起動して、shared_preload_libraries パラメータの変更を有効にします。
9. インスタンスが使用可能になったら、pg_tle が初期化されていることを確認します。psql を使用して Aurora PostgreSQL DB クラスターのライターインスタンス、次のコマンドを実行します。

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_tle
(1 row)
```

10. pg_tle 拡張子を初期化すると、拡張機能を作成できるようになりました。

```
CREATE EXTENSION pg_tle;
```

以下の psql メタコマンドを使用して、拡張機能がインストールされていることを確認できます。

```
labdb=> \dx
                          List of installed extensions
  Name   | Version |   Schema   |                Description
---------+---------+------------+---------------------------------------------
```

```
 pg_tle  | 1.0.1   | pgtle      | Trusted-Language Extensions for PostgreSQL
 plpgsql | 1.0     | pg_catalog | PL/pgSQL procedural language
```

11. Aurora PostgreSQL DB クラスター のセットアップ時に作成したプライマリユーザー名に pgtle_admin ロールを付与します。デフォルトを受け入れた場合は、postgres です。

```
labdb=> GRANT pgtle_admin TO postgres;
GRANT ROLE
```

次の例に示すように、psql メタコマンドを使用して、付与されたことを確認できます。出力には pgtle_admin と postgres ロールのみが表示されます。詳細については、「PostgreSQL のロールとアクセス権限について」を参照してください。

```
labdb=> \du
                          List of roles
    Role name    |           Attributes            |              Member of
-----------------+---------------------------------
+-----------------------------------
pgtle_admin     | Cannot login                     | {}
postgres        | Create role, Create DB          +| {rds_superuser,pgtle_admin}
                | Password valid until infinity    |...
```

12. \q メタコマンドを使用して psql セッションを終了します。

```
\q
```

TLE 拡張機能の作成を開始するには、「例: SQL を使用した信頼できる言語拡張関数の作成」を参照してください。

AWS CLI

AWS CLI をデフォルト AWS リージョン に設定することで、CLI コマンドを使用するときに --region 引数を指定しなくても済みます。詳細については、AWS Command Line Interface ユーザーガイドの「設定の基本」を参照してください。

Trusted Language Extensions をセットアップするには

1. shared_preload_libraries パラメータに pg_tle を追加するには、modify-db-parameter-group AWS CLI コマンドを使用します。

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pg_tle,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. reboot-db-instance AWS CLI コマンドを使用して、Aurora PostgreSQL DB クラスター のライター インスタンスを再起動し、pg_tle ライブラリを初期化します。

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

3. インスタンスが使用可能になると、pg_tle が初期化されていることを確認できます。psql を使用して Aurora PostgreSQL DB クラスターのライターインスタンス、次のコマンドを実行します。

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_tle
(1 row)
```

pg_tle を初期化すると、拡張機能を作成できるようになりました。

```
CREATE EXTENSION pg_tle;
```

4. Aurora PostgreSQL DB クラスター のセットアップ時に作成したプライマリユーザー名に pgtle_admin ロールを付与します。デフォルトを受け入れた場合は、postgres です。

```
GRANT pgtle_admin TO postgres;
GRANT ROLE
```

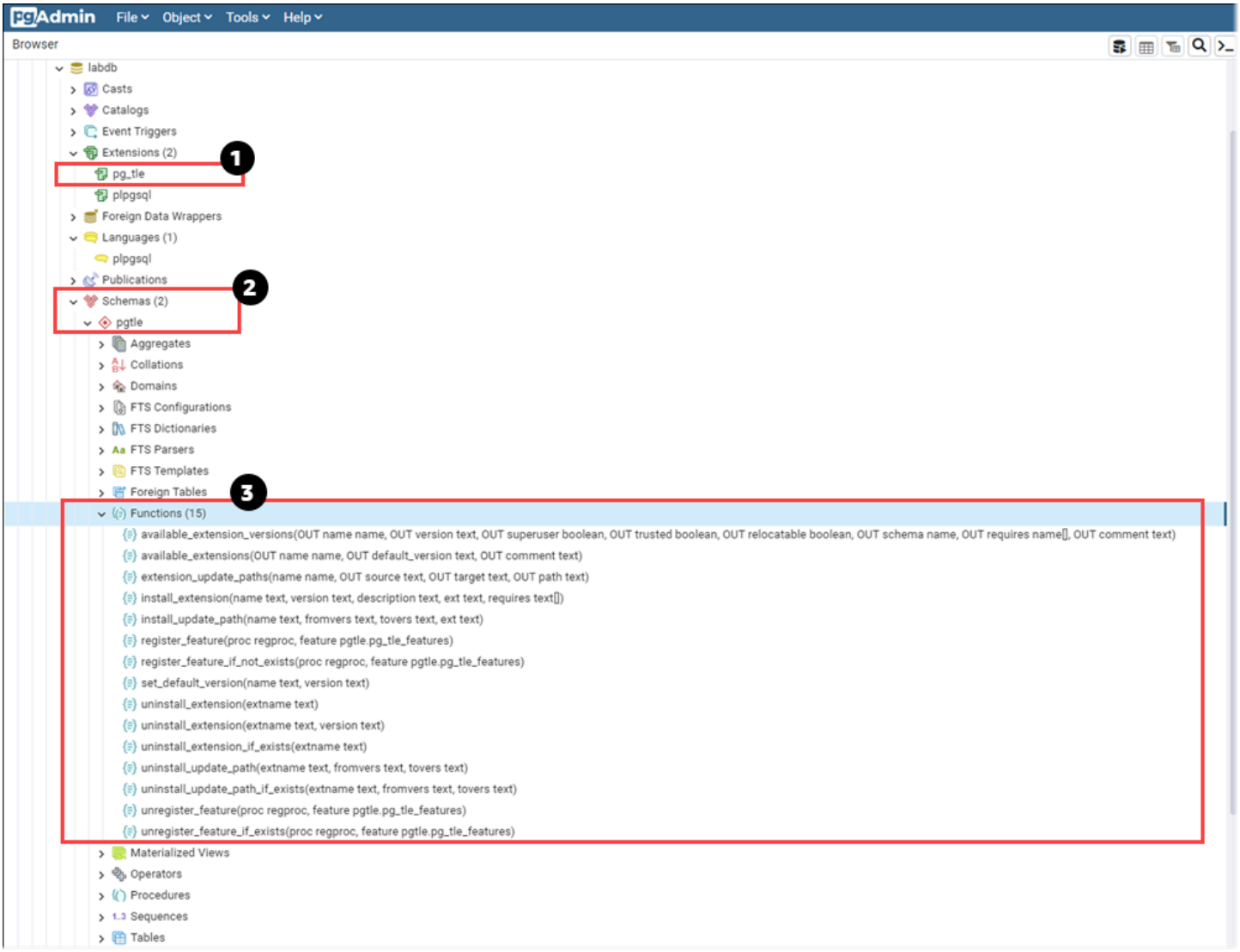
5. 以下のように psql セッションを終了します。

```
labdb=> \q
```

TLE 拡張機能の作成を開始するには、「例: SQL を使用した信頼できる言語拡張関数の作成」を参照してください。

Trusted Language Extensions for PostgreSQL の概要

Trusted Language Extensions for PostgreSQL は PostgreSQL 拡張機能で、他の PostgreSQL 拡張機能をセットアップするのと同じ方法で Aurora PostgreSQL DB クラスター にインストールします。pgAdmin クライアントツールのサンプルデータベースの次の画像では、pg_tle 拡張機能を構成するコンポーネントの一部を確認できます。



以下の詳細を表示できます。

1. Trusted Language Extensions (TLE) for PostgreSQL 開発キットは pg_tle 拡張機能としてパッケージ化されています。そのため、pg_tle がインストールされているデータベースで使用可能な拡張機能にそれが追加されます。
2. TLE には独自のスキーマ、pgtle があります。このスキーマには、作成した拡張機能をインストールおよび管理するためのヘルパー関数 (3) が含まれています。
3. TLE には、拡張機能のインストール、登録、管理のためのヘルパー関数が十数種類用意されています。これらの関数の詳細については、「Trusted Language Extensions for PostgreSQL の関数リファレンス」を参照してください。

pg_tle 拡張機能のその他のコンポーネントには、以下のものが含まれています。

- **pgtle_admin** ロール – pgtle_admin ロールは、pg_tle 拡張機能のインストール時に作成されます。この役職には特権があり、そのように扱われる必要があります。データベースユーザーに pgtle_admin ロールを付与する場合は、最小特権の原則に従うことを強くお勧めします。つまり、postgres のような新しい TLE 拡張機能の作成、インストール、管理が許可されているデータベースユーザーにのみ pgtle_admin ロールを付与します。
- **pgtle.feature_info** テーブル – pgtle.feature_info テーブルは保護されたテーブルで、TLE、フック、およびそれらが使用するカスタムストアドプロシージャと関数に関する情報が含まれています。pgtle_admin権限がある場合は、以下の Trusted Language Extensions の関数を使用して、テーブル内の情報を追加および更新します。
  - pgtle.register_feature
  - pgtle.register_feature_if_not_exists
  - pgtle.unregister_feature
  - pgtle.unregister_feature_if_exists

Aurora PostgreSQL の TLE 拡張機能の作成

TLE を使用して作成した拡張機能は、pg_tle 拡張機能がインストールされている任意の Aurora PostgreSQL DB クラスター にインストールできます。pg_tle 拡張機能のスコープは、インストールされている PostgreSQL データベースに限定されます。TLE を使用して作成した拡張機能は、同じデータベースを対象としています。

さまざまな pgtle 関数を使用して、TLE 拡張機能を構成するコードをインストールします。以下の Trusted Language Extensions 関数には pgtle_admin すべてロールが必要です。

- pgtle.install_extension

- pgtle.install_update_path
- pgtle.register_feature
- pgtle.register_feature_if_not_exists
- pgtle.set_default_version
- pgtle.uninstall_extension(name)
- pgtle.uninstall_extension(name, version)
- pgtle.uninstall_extension_if_exists
- pgtle.uninstall_update_path
- pgtle.uninstall_update_path_if_exists
- pgtle.unregister_feature
- pgtle.unregister_feature_if_exists

例: SQL を使用した信頼できる言語拡張関数の作成

次の例は、さまざまな式を使用して距離を計算するためのいくつかの SQL 関数を含む `pg_distance` という名前の TLE 拡張機能を作成する方法を示しています。リストには、マンハッタン距離を計算する関数とユークリッド距離を計算する関数があります。これらの式の違いの詳細については、Wikipedia の「Taxicab geometry」と「Euclidean geometry」を参照してください。

Aurora PostgreSQL DB クラスターに Trusted Language Extensions を設定する で説明されているように `pg_tle` 拡張機能をセットアップしていれば、この例を独自の Aurora PostgreSQL DB クラスター で使用できます。

> **Note**
>
> この手順を実行するには、`pgtle_admin` ロールの権限が必要です。

サンプルの TLE 拡張機能を作成するには

以下の手順では、`labdb` という名前のサンプルデータベースを使用します。このデータベースは `postgres` プライマリユーザーが所有しています。`postgres` ロールには、`pgtle_admin` ロールのアクセス許可もあります。

1. `psql` を使用して、Aurora PostgreSQL DB クラスターのライターインスタンスに接続します。

```
psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
--port=5432 --username=postgres --password --dbname=labdb
```

2. 次のコードをコピーして psql セッションコンソールに貼り付けて、pg_distance という名前の TLE 拡張機能を作成します。

```
SELECT pgtle.install_extension
(
 'pg_distance',
 '0.1',
  'Distance functions for two points',
$_pg_tle_$
    CREATE FUNCTION dist(x1 float8, y1 float8, x2 float8, y2 float8, norm int)
    RETURNS float8
    AS $$
      SELECT (abs(x2 - x1) ^ norm + abs(y2 - y1) ^ norm) ^ (1::float8 / norm);
    $$ LANGUAGE SQL;

    CREATE FUNCTION manhattan_dist(x1 float8, y1 float8, x2 float8, y2 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 1);
    $$ LANGUAGE SQL;

    CREATE FUNCTION euclidean_dist(x1 float8, y1 float8, x2 float8, y2 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 2);
    $$ LANGUAGE SQL;
$_pg_tle_$
);
```

次のような出力が表示されます。

```
install_extension
---------------
 t
(1 row)
```

これで、pg_distance 拡張機能を構成するアーティファクトがデータベースにインストールされました。これらのアーティファクトには、コントロールファイルと拡張機能のコードが含まれ

ます。これらは、CREATE EXTENSION コマンドを使用して拡張機能を作成するために必要となる項目です。つまり、データベースユーザーがその関数を利用できるようにするには、やはり拡張機能を作成する必要があります。

3. 拡張機能を作成するには、他の拡張機能と同じように CREATE EXTENSION コマンドを使用します。他の拡張機能と同様に、データベースユーザーにはデータベース内の CREATE アクセス許可が必要です。

```
CREATE EXTENSION pg_distance;
```

4. pg_distance TLE 拡張機能をテストするには、これを使用して 4 点間のマンハッタン距離を計算できます。

```
labdb=> SELECT manhattan_dist(1, 1, 5, 5);
8
```

同じ点群間のユークリッド距離を計算するには、以下を使用できます。

```
labdb=> SELECT euclidean_dist(1, 1, 5, 5);
5.656854249492381
```

pg_distance 拡張機能は関数をデータベースに読み込み、データベースに対するアクセス許可を持つすべてのユーザーがその関数を利用できるようにします。

TLE 拡張機能の変更

この TLE 拡張機能にパッケージされている関数のクエリパフォーマンスを向上させるには、次の 2 つの PostgreSQL 属性を仕様に追加してください。

- IMMUTABLE – IMMUTABLE 属性により、クエリオプティマイザが最適化を使用してクエリの応答時間を改善できるようになります。詳細については、PostgreSQL ドキュメントの「関数のボラティリティカテゴリ」を参照してください。
- PARALLEL SAFE – PARALLEL SAFE属性は、PostgreSQL が関数をパラレルモードで実行できるようにするもう 1 つの属性です。詳細については、PostgreSQL のドキュメントの「機能の作成」を参照してください。

次の例では、pgtle.install_update_path 関数を使用してこれらの属性を各関数に追加し、pg_distance TLE 拡張機能のバージョン 0.2 を作成する方法を確認できます。この関数

の詳細については、「pgtle.install_update_path」を参照してください。このタスクを実行するには、pgtle_admin ロールが必要です。

既存の TLE 拡張機能を更新してデフォルトバージョンを指定するには

1. psql または pgAdmin などの別のクライアントツールを使用して、Aurora PostgreSQL DB クラスター のライターインスタンスに接続します。

```
psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
--port=5432 --username=postgres --password --dbname=labdb
```

2. 次のコードをコピーして psql セッションコンソールに貼り付けることで、既存の TLE 拡張機能を変更します。

```
SELECT pgtle.install_update_path
(
 'pg_distance',
 '0.1',
 '0.2',
$_pg_tle_$
    CREATE OR REPLACE FUNCTION dist(x1 float8, y1 float8, x2 float8, y2 float8,
 norm int)
    RETURNS float8
    AS $$
      SELECT (abs(x2 - x1) ^ norm + abs(y2 - y1) ^ norm) ^ (1::float8 / norm);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;

    CREATE OR REPLACE FUNCTION manhattan_dist(x1 float8, y1 float8, x2 float8, y2
 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 1);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;

    CREATE OR REPLACE FUNCTION euclidean_dist(x1 float8, y1 float8, x2 float8, y2
 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 2);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;
$_pg_tle_$
);
```

次のようなレスポンスが表示されます。

```
install_update_path
--------------------
 t
(1 row)
```

このバージョンの拡張機能をデフォルトバージョンにすると、データベースユーザーがデータベースで拡張機能を作成または更新するときにバージョンを指定する必要がなくなります。

3. TLE 拡張機能の修正バージョン (バージョン 0.2) がデフォルトバージョンになるように指定するには、次の例に示す pgtle.set_default_version 関数を使用します。

```
SELECT pgtle.set_default_version('pg_distance', '0.2');
```

この関数の詳細については、「pgtle.set_default_version」を参照してください。

4. コードを配置したら、次に示すように、ALTER EXTENSION ... UPDATE コマンドを使用して、インストールされている TLE 拡張機能を通常の方法で更新できます。

```
ALTER EXTENSION pg_distance UPDATE;
```

TLE 拡張機能をデータベースから削除する

TLE 拡張機能は、他の PostgreSQL 拡張機能の場合と同じように DROP EXTENSION コマンドを使用して削除できます。拡張機能を削除しても、拡張機能を構成するインストールファイルは削除されないため、ユーザーは拡張機能を再作成できます。拡張機能とそのインストールファイルを削除するには、次の 2 段階のプロセスを実行します。

TLE 拡張機能とそのインストールファイルを削除するには

1. psql または別のクライアントツールを使用して Aurora PostgreSQL DB クラスターのライターインスタンス、

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=dbname
```

2. PostgreSQL 拡張機能と同様に、この拡張機能を削除してください。

```
DROP EXTENSION your-TLE-extension
```

例えば、例: SQL を使用した信頼できる言語拡張関数の作成 で詳細を説明しているように pg_distance という拡張機能を作成する場合は、次のように拡張機能を削除できます。

```
DROP EXTENSION pg_distance;
```

次のように、拡張機能が削除されたことを確認する出力が表示されます。

```
DROP EXTENSION
```

この時点で、拡張機能はデータベースでアクティブではなくなります。ただし、インストールファイルとコントロールファイルはデータベースにまだ残っているため、データベースユーザーは必要に応じて拡張機能を再作成できます。

- 拡張ファイルをそのまま残して、データベースユーザーが TLE 拡張機能を作成できるようにする場合は、ここで終了してください。
- 拡張機能を占めるすべてのファイルを削除する場合は、次のステップに進みます。

3. 拡張機能のインストールファイルをすべて削除するには、pgtle.uninstall_extension 関数を使用してください。この関数は、拡張機能のコードとコントロールファイルをすべて削除します。

```
SELECT pgtle.uninstall_extension('your-tle-extension-name');
```

例えば、すべての pg_distance インストールファイルを削除するには、次のコマンドを使用します。

```
SELECT pgtle.uninstall_extension('pg_distance');
 uninstall_extension
---------------------
 t
(1 row)
```

Trusted Language Extensions for PostgreSQL のアンインストール

TLE を使用して独自の TLE 拡張機能を作成する必要がなくなった場合は、pg_tle 拡張機能を削除してすべてのアーティファクトを削除できます。このアクションには、データベース内のすべての TLE 拡張機能の削除と pgtle スキーマの削除が含まれます。

pg_tle 拡張機能とそのスキーマをデータベースから削除するには

1. psql または別のクライアントツールを使用して Aurora PostgreSQL DB クラスターのライターインスタンス、

   ```
   psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password --dbname=dbname
   ```

2. pg_tle 拡張機能をデータベースから削除します。データベースに独自の TLE 拡張機能がまだデータベースで実行されている場合は、それらの拡張機能も削除する必要があります。そのためには、次に示すように、CASCADE キーワードを使用します。

   ```
   DROP EXTENSION pg_tle CASCADE;
   ```

   pg_tle 拡張機能がデータベースでまだ有効になっていない場合は、CASCADE キーワードを使用する必要はありません。

3. pgtle スキーマを削除します。このアクションにより、データベースからすべての管理関数が削除されます。

   ```
   DROP SCHEMA pgtle CASCADE;
   ```

   このコマンドは、プロセスが完了すると、以下を返します。

   ```
   DROP SCHEMA
   ```

   pg_tle 拡張機能、そのスキーマ、関数、およびすべてのアーティファクトが削除されます。TLE を使用して新しい拡張機能を作成するには、セットアッププロセスをもう一度実行してください。詳細については、「Aurora PostgreSQL DB クラスターに Trusted Language Extensions を設定する」を参照してください。

TLE 拡張機能で PostgreSQL フックを使用する

フックは PostgreSQL で利用できるコールバックメカニズムで、開発者は通常のデータベースオペレーション中にカスタム関数やその他のルーチンを呼び出すことができます。TLE 開発キットは PostgreSQL フックをサポートしているため、実行時にカスタム関数を PostgreSQL の動作と統合できます。例えば、フックを使用して認証プロセスを独自のカスタムコードに関連付けたり、特定のニーズに合わせてクエリの計画と実行プロセスを変更したりできます。

TLE 拡張機能にはフックを使用できます。フックの適用範囲がグローバルな場合、すべてのデータベースに適用されます。そのため、TLE 拡張機能がグローバルフックを使用している場合は、ユーザーがアクセスできるすべてのデータベースに TLE 拡張機能を作成する必要があります。

pg_tle 拡張機能を使用して独自の Trusted Language Extensions を構築する場合、SQL API の利用可能なフックを使用して拡張機能の関数を構築できます。すべてのフックを pg_tle に登録する必要があります。一部のフックでは、さまざまな設定パラメータを設定する必要がある場合もあります。例えば、passcode チェックフックをオン、オフ、または必須に設定できます。使用可能な pg_tle フックの特定の要件の詳細については、「Trusted Language Extensions for PostgreSQL のフックリファレンス」を参照してください。

例: PostgreSQL フックを使用する拡張機能の作成

このセクションで説明する例では、PostgreSQL フックを使用して特定の SQL のオペレーション中に入力されたパスワードをチェックし、データベースユーザーが自分のパスワードを password_check.bad_passwords テーブルに含まれるパスワードに設定できないようにします。この表には、一般的によく使用されているものの、簡単に破られてしまうパスワードの選択肢の上位 10 件が掲載されています。

この例を Aurora PostgreSQL DB クラスター、に設定するには、Trusted Language Extensions が既にインストールされている必要があります。詳細については、「Aurora PostgreSQL DB クラスターに Trusted Language Extensions を設定する」を参照してください。

パスワードチェックフックの例を設定するには

1. psql を使用して、Aurora PostgreSQL DB クラスターのライターインスタンスに接続します。

```
psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
--port=5432 --username=postgres --password --dbname=labdb
```

2. パスワードチェックフックコードリスト のコードをコピーし、データベースに貼り付けます。

```
SELECT pgtle.install_extension (
  'my_password_check_rules',
  '1.0',
  'Do not let users use the 10 most commonly used passwords',
$_pgtle_$
  CREATE SCHEMA password_check;
  REVOKE ALL ON SCHEMA password_check FROM PUBLIC;
  GRANT USAGE ON SCHEMA password_check TO PUBLIC;

  CREATE TABLE password_check.bad_passwords (plaintext) AS
  VALUES
    ('123456'),
    ('password'),
    ('12345678'),
    ('qwerty'),
    ('123456789'),
    ('12345'),
    ('1234'),
    ('111111'),
    ('1234567'),
    ('dragon');
  CREATE UNIQUE INDEX ON password_check.bad_passwords (plaintext);

  CREATE FUNCTION password_check.passcheck_hook(username text, password text,
 password_type pgtle.password_types, valid_until timestamptz, valid_null boolean)
  RETURNS void AS $$
    DECLARE
      invalid bool := false;
    BEGIN
      IF password_type = 'PASSWORD_TYPE_MD5' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE ('md5' || md5(bp.plaintext || username)) = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password
 dictionary';
        END IF;
      ELSIF password_type = 'PASSWORD_TYPE_PLAINTEXT' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
```

```
          WHERE bp.plaintext = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common common password
 dictionary';
        END IF;
      END IF;
    END
  $$ LANGUAGE plpgsql SECURITY DEFINER;

  GRANT EXECUTE ON FUNCTION password_check.passcheck_hook TO PUBLIC;

  SELECT pgtle.register_feature('password_check.passcheck_hook', 'passcheck');
$_pgtle_$
);
```

拡張機能がデータベースに読み込まれると、次のような出力が表示されます。

```
 install_extension
-------------------
 t
(1 row)
```

3. データベースに接続したままで、拡張機能を作成できるようになりました。

```
CREATE EXTENSION my_password_check_rules;
```

4. 次の psql メタコマンドを使用して、拡張機能がデータベースに作成されたことを確認できます。

```
\dx
                        List of installed extensions
          Name           | Version |   Schema   |
 Description
-------------------------+---------+------------
+-------------------------------------------------------------
 my_password_check_rules | 1.0     | public     | Prevent use of any of the top-ten
 most common bad passwords
 pg_tle                  | 1.0.1   | pgtle      | Trusted-Language Extensions for
 PostgreSQL
 plpgsql                 | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

5. 別のターミナルセッションを開いて、AWS CLI を操作します。パスワードチェックフックを有効にするには、カスタム DB パラメータグループを変更する必要があります。そのためには、以下の例に示すように modify-db-parameter-group CLI コマンドを使用します。

```
aws rds modify-db-parameter-group \
    --region aws-region \
    --db-parameter-group-name your-custom-parameter-group \
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

   パラメータグループ設定の変更が適用されるまでには数分かかる場合があります。ただし、このパラメータは動的であるため、設定を有効にするために Aurora PostgreSQL DB クラスター のライターインスタンスを再起動する必要はありません。

6. psql セッションを開き、データベースに問い合わせて password_check フックが有効になっていることを確認します。

```
labdb=> SHOW pgtle.enable_password_check;
pgtle.enable_password_check
----------------------------
on
(1 row)
```

パスワードチェックフックがアクティブになりました。次の例に示すように、新しいロールを作成し、不適切なパスワードを使用してテストできます。

```
CREATE ROLE test_role PASSWORD 'password';
ERROR:  Cannot use passwords from the common password dictionary
CONTEXT:  PL/pgSQL function
 password_check.passcheck_hook(text,text,pgtle.password_types,timestamp with time
 zone,boolean) line 21 at RAISE
SQL statement "SELECT password_check.passcheck_hook(
    $1::pg_catalog.text,
    $2::pg_catalog.text,
    $3::pgtle.password_types,
    $4::pg_catalog.timestamptz,
    $5::pg_catalog.bool)"
```

出力は、読みやすい形式にしてあります。

次の例では、pgsql インタラクティブメタコマンドの \password 動作が password_check フックの影響を受けることを示しています。

```
postgres=> SET password_encryption TO 'md5';
SET
postgres=> \password
Enter new password for user "postgres":*****
Enter it again:*****
ERROR:  Cannot use passwords from the common password dictionary
CONTEXT:  PL/pgSQL function
 password_check.passcheck_hook(text,text,pgtle.password_types,timestamp with time
 zone,boolean) line 12 at RAISE
SQL statement "SELECT password_check.passcheck_hook($1::pg_catalog.text,
 $2::pg_catalog.text, $3::pgtle.password_types, $4::pg_catalog.timestamptz,
 $5::pg_catalog.bool)"
```

この TLE 拡張機能モジュールを削除して、必要に応じてソースファイルをアンインストールできます。詳細については、「TLE 拡張機能をデータベースから削除する」を参照してください。

パスワードチェックフックコードリスト

ここに示すサンプルコードは、my_password_check_rules TLE 拡張機能の仕様を定義しています。このコードをコピーしてデータベースに貼り付けると、my_password_check_rules 拡張機能のコードがデータベースにロードされ、password_check フックが拡張機能で使用できるように登録されます。

```
SELECT pgtle.install_extension (
  'my_password_check_rules',
  '1.0',
  'Do not let users use the 10 most commonly used passwords',
$_pgtle_$
  CREATE SCHEMA password_check;
  REVOKE ALL ON SCHEMA password_check FROM PUBLIC;
  GRANT USAGE ON SCHEMA password_check TO PUBLIC;

  CREATE TABLE password_check.bad_passwords (plaintext) AS
  VALUES
    ('123456'),
    ('password'),
    ('12345678'),
    ('qwerty'),
    ('123456789'),
```

```
    ('12345'),
    ('1234'),
    ('111111'),
    ('1234567'),
    ('dragon');
  CREATE UNIQUE INDEX ON password_check.bad_passwords (plaintext);

  CREATE FUNCTION password_check.passcheck_hook(username text, password text,
 password_type pgtle.password_types, valid_until timestamptz, valid_null boolean)
  RETURNS void AS $$
    DECLARE
      invalid bool := false;
    BEGIN
      IF password_type = 'PASSWORD_TYPE_MD5' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE ('md5' || md5(bp.plaintext || username)) = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password dictionary';
        END IF;
      ELSIF password_type = 'PASSWORD_TYPE_PLAINTEXT' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE bp.plaintext = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common common password
 dictionary';
        END IF;
      END IF;
    END
  $$ LANGUAGE plpgsql SECURITY DEFINER;

  GRANT EXECUTE ON FUNCTION password_check.passcheck_hook TO PUBLIC;

  SELECT pgtle.register_feature('password_check.passcheck_hook', 'passcheck');
$_pgtle_$
);
```

Trusted Language Extensions for PostgreSQL の関数リファレンス

Trusted Language Extensions for PostgreSQL で利用できる関数については、以下のリファレンスドキュメントを参照してください。これらの関数を使用して、TLE 拡張機能、つまり Trusted Language Extensions 開発キットを使用して開発した PostgreSQL 拡張機能のインストール、登録、更新、管理を行います。

トピック

pgtle.available_extensions

`pgtle.available_extensions` 関数は、集合を返す関数です。データベース内の使用可能なすべての TLE 拡張機能を返します。返される各行には、1 つの TLE 拡張機能に関する情報が含まれています。

関数プロトタイプ

```
pgtle.available_extensions()
```

ロール

なし。

引数

なし。

出力

- name – TLE 拡張機能の名前。
- default_version – バージョンを指定せずに CREATE EXTENSION が呼び出されたときに使用する TLE 拡張機能のバージョン。
- description – TLE 拡張機能に関するさらに詳細な説明。

使用例

```
SELECT * FROM pgtle.available_extensions();
```

pgtle.available_extension_versions

available_extension_versions 関数は、集合を返す関数です。使用可能なすべての TLE 拡張機能とそのバージョンの一覧を返します。各行には、特定のロールが必要かどうかなど、特定の TLE 拡張機能のバージョンに関する情報が含まれています。

関数プロトタイプ

```
pgtle.available_extension_versions()
```

ロール

なし。

引数

なし。

出力

- name – TLE 拡張機能の名前。

- version – TLE 拡張機能のバージョン。
- superuser – この値は、TLE 拡張機能では常に false です。TLE 拡張機能の作成または更新に必要なアクセス許可は、特定のデータベースに他のオブジェクトを作成する場合と同じです。
- trusted – この値は、TLE 拡張機能では常に false です。
- relocatable – この値は、TLE 拡張機能では常に false です。
- schema – TLE 拡張機能がインストールされているスキーマの名前を指定します。
- requires – この TLE 拡張機能に必要な他の拡張機能の名前を含む配列。
- description – TLE 拡張機能の詳細な説明。

出力値の詳細については、PostgreSQL ドキュメントの [Extension > Extension Files] (拡張機能 > 拡張機能ファイル) にある「関連オブジェクトのパッケージ化」を参照してください。

使用例

```
SELECT * FROM pgtle.available_extension_versions();
```

pgtle.extension_update_paths

extension_update_paths 関数は、集合を返す関数です。TLE 拡張機能で使用可能なすべての更新パスのリストを返します。各行には、その TLE 拡張機能で使用可能なアップグレードまたはダウングレードが含まれています。

関数プロトタイプ

```
pgtle.extension_update_paths(name)
```

ロール

なし。

引数

name – アップグレードパスを取得する TLE 拡張機能の名前。

出力

- source – 更新のソースバージョン。

- target – 更新のターゲットバージョン。
- path – TLE 拡張機能を source バージョンから target にアップグレードするために使用されるアップグレードパス (例えば 0.1--0.2)。

使用例

```
SELECT * FROM pgtle.extension_update_paths('your-TLE');
```

pgtle.install_extension

この install_extension 関数を使用すると、TLE 拡張機能を構成するアーティファクトをデータベースにインストールし、その後、CREATE EXTENSION コマンドを使用して作成できます。

関数プロトタイプ

```
pgtle.install_extension(name text, version text, description text, ext text, requires
 text[] DEFAULT NULL::text[])
```

ロール

なし。

引数

- name – TLE 拡張機能の名前。この値は CREATE EXTENSION を呼び出すときに使用されます。
- version – TLE 拡張機能のバージョン。
- description – TLE 拡張機能に関する詳細な説明。この説明は、pgtle.available_extensions() の comment フィールドに表示されます。
- ext – TLE 拡張機能の内容。この値には、関数などのオブジェクトが含まれます。
- requires – この TLE 拡張機能の依存関係を指定するオプションパラメータ。pg_tle 拡張機能は、依存関係として自動的に追加されます。

これらの引数の多くは、PostgreSQL インスタンスのファイルシステムに PostgreSQL 拡張機能をインストールするための拡張制御ファイルに含まれている引数と同じです。詳細については、PostgreSQL ドキュメントの「拡張機能への関連オブジェクトのパッケージ化」にある「拡張ファイル」を参照してください。

出力

この関数は、正常時には OK を、エラー時には NULL を返します。

- OK – TLE 拡張機能がデータベースに正常にインストールされました。
- NULL – TLE 拡張機能がデータベースに正常にインストールされませんでした。

使用例

```
SELECT pgtle.install_extension(
 'pg_tle_test',
 '0.1',
 'My first pg_tle extension',
$_pgtle_$
  CREATE FUNCTION my_test()
  RETURNS INT
  AS $$
    SELECT 42;
  $$ LANGUAGE SQL IMMUTABLE;
$_pgtle_$
);
```

pgtle.install_update_path

この install_update_path 関数は、TLE 拡張機能の 2 つの異なるバージョン間の更新パスを提供します。この機能によって、TLE 拡張機能のユーザーは ALTER EXTENSION ... UPDATE 構文を使用してバージョンを更新できます。

関数プロトタイプ

```
pgtle.install_update_path(name text, fromvers text, tovers text, ext text)
```

ロール

pgtle_admin

引数

- name – TLE 拡張機能の名前。この値は CREATE EXTENSION を呼び出すときに使用されます。
- fromvers – アップグレードの TLE 拡張機能のソースバージョン。
- tovers – アップグレードする TLE 拡張機能のデスティネーションバージョン。

- ext – 更新の内容。この値には、関数などのオブジェクトが含まれます。

出力

なし。

使用例

```
SELECT pgtle.install_update_path('pg_tle_test', '0.1', '0.2',
  $_pgtle_$
    CREATE OR REPLACE FUNCTION my_test()
    RETURNS INT
    AS $$
      SELECT 21;
    $$ LANGUAGE SQL IMMUTABLE;
  $_pgtle_$
);
```

pgtle.register_feature

この register_feature 関数は、指定された内部 PostgreSQL 機能を pgtle.feature_info テーブルに追加します。PostgreSQL フックは、内部 PostgreSQL 機能の一例です。Trusted Language Extensions 開発キットは、PostgreSQL フックの使用をサポートしています。現在、この関数は次の機能をサポートしています。

- passcheck – PostgreSQL のパスワードチェック動作をカスタマイズするプロシージャまたは関数にパスワードチェックフックを登録します。

関数プロトタイプ

```
pgtle.register_feature(proc regproc, feature pg_tle_feature)
```

ロール

pgtle_admin

引数

- proc – その機能に使用するストアドプロシージャまた関数の名前。
- feature – 関数に登録する pg_tle 機能 (passcheck など) の名前。

出力

なし。

使用例

```
SELECT pgtle.register_feature('pw_hook', 'passcheck');
```

pgtle.register_feature_if_not_exists

この pgtle.register_feature_if_not_exists 関数は、指定した PostgreSQL 機能を pgtle.feature_info テーブルに追加し、その機能を使用する TLE 拡張機能またはその他のプロシージャまたは関数を識別します。フックと Trusted Language Extensions の詳細については、「TLE 拡張機能で PostgreSQL フックを使用する」を参照してください。

関数プロトタイプ

```
pgtle.register_feature_if_not_exists(proc regproc, feature pg_tle_feature)
```

ロール

pgtle_admin

引数

- proc – TLE 拡張機能として使用するロジック (コード) を含むストアドプロシージャまたは関数の名前。例えば、pw_hook コードです。
- feature – TLE 関数に登録する PostgreSQL 機能の名前。現在、使用できる機能は passcheck フックだけです。詳細については、「パスワードチェックフック (passcheck)」を参照してください。

出力

指定された拡張機能を登録した後、true を返します。機能が既に登録されていた場合は false を返します。

使用例

```
SELECT pgtle.register_feature_if_not_exists('pw_hook', 'passcheck');
```

pgtle.set_default_version

set_default_version 関数では、TLE 拡張機能に default_version を指定できます。この関数を使用してアップグレードパスを定義し、そのバージョンを TLE 拡張機能のデフォルトとして指定できます。データベースユーザーが CREATE EXTENSION および ALTER EXTENSION ... UPDATE コマンドで TLE 拡張機能を指定すると、そのユーザー用にそのバージョンの TLE 拡張機能がデータベースに作成されます。

この関数は成功すると true を返します。name 引数で指定された TLE 拡張機能が存在しない場合、関数はエラーを返します。同様に、TLE 拡張機能の version が存在しない場合、関数はエラーを返します。

関数プロトタイプ

```
pgtle.set_default_version(name text, version text)
```

ロール

pgtle_admin

引数

- name – TLE 拡張機能の名前。この値は CREATE EXTENSION を呼び出すときに使用されます。
- version – デフォルトに設定する TLE 拡張機能のバージョン。

出力

- true — デフォルトバージョンの設定が成功すると、関数は true を返します。
- ERROR – 指定された名前またはバージョンの TLE 拡張機能が存在しない場合は、エラーメッセージを返します。

使用例

```
SELECT * FROM pgtle.set_default_version('my-extension', '1.1');
```

pgtle.uninstall_extension(name)

uninstall_extension 関数は、TLE 拡張機能のすべてのバージョンをデータベースから削除します。この関数により、今後の CREATE EXTENSION の呼び出しで TLE 拡張機能がインストールされないようにします。TLE 拡張機能がデータベースに存在しない場合、エラーが発生します。

uninstall_extension 関数は、データベースで現在アクティブな TLE 拡張機能を削除しません。現在アクティブな TLE 拡張機能を削除するには、DROP EXTENSION を明示的に呼び出して削除する必要があります。

関数プロトタイプ

```
pgtle.uninstall_extension(extname text)
```

ロール

pgtle_admin

引数

- extname – アンインストールする TLE 拡張機能の名前。この名前は、特定のデータベースで使用する TLE 拡張機能をロードするために CREATE EXTENSION で使用される名前と同じです。

出力

なし。

使用例

```
SELECT * FROM pgtle.uninstall_extension('pg_tle_test');
```

pgtle.uninstall_extension(name, version)

uninstall_extension(name, version) 関数は、指定されたバージョンの TLE 拡張機能をデータベースから削除します。この関数は、CREATE EXTENSION および ALTER EXTENSION が TLE 拡張機能を指定されたバージョンにインストールまたは更新するのを防ぎます。この関数は、TLE 拡張機能の指定されたバージョンのすべての更新パスも削除します。この関数は、データベースで現在アクティブになっている TLE 拡張機能をアンインストールしません。TLE 拡張機能

を削除するには、明示的に DROP EXTENSION を呼び出す必要があります。TLE 拡張機能のすべてのバージョンをアンインストールするには、「pgtle.uninstall_extension(name)」を参照してください。

関数プロトタイプ

```
pgtle.uninstall_extension(extname text, version text)
```

ロール

pgtle_admin

引数

- extname – TLE 拡張機能の名前。この値は CREATE EXTENSION を呼び出すときに使用されます。
- version - データベースからアンインストールする TLE 拡張機能のバージョン。

出力

なし。

使用例

```
SELECT * FROM pgtle.uninstall_extension('pg_tle_test', '0.2');
```

pgtle.uninstall_extension_if_exists

uninstall_extension_if_exists 関数は、特定のデータベースから TLE 拡張機能のすべてのバージョンを削除します。TLE 拡張機能が存在しない場合、関数は何も返しません (エラーメッセージは発生しません)。指定された拡張機能がデータベース内で現在アクティブになっている場合、この関数はその拡張機能を削除しません。この関数を使用してアーティファクトをアンインストールする前に、DROP EXTENSION を明示的に呼び出して TLE 拡張機能を削除する必要があります。

関数プロトタイプ

```
pgtle.uninstall_extension_if_exists(extname text)
```

ロール

pgtle_admin

引数

- extname – TLE 拡張機能の名前。この値は CREATE EXTENSION を呼び出すときに使用されます。

出力

uninstall_extension_if_exists 関数は、指定された拡張機能をアンインストールした後に、true を返します。指定した拡張機能が存在しない場合、この関数は false を返します。

- true – TLE 拡張機能をアンインストールした後に true を返します。
- false – TLE 拡張機能がデータベースに存在しない場合に false を返します。

使用例

```
SELECT * FROM pgtle.uninstall_extension_if_exists('pg_tle_test');
```

pgtle.uninstall_update_path

uninstall_update_path 関数は TLE 拡張機能から指定された更新パスを削除します。これにより、ALTER EXTENSION ... UPDATE TO では、これを更新パスとして使用できなくなります。

TLE 拡張機能が、この更新パス上のいずれかのバージョンで現在使用されている場合、その拡張機能はデータベースに残ります。

指定された更新パスが存在しない場合、この関数はエラーを発生させます。

関数プロトタイプ

```
pgtle.uninstall_update_path(extname text, fromvers text, tovers text)
```

ロール

pgtle_admin

引数

- extname – TLE 拡張機能の名前。この値は CREATE EXTENSION を呼び出すときに使用されます。
- fromvers – 更新パスで使用されている TLE 拡張機能のソースバージョン。
- tovers – 更新パスで使用されている TLE 拡張機能の送信先バージョン。

出力

なし。

使用例

```
SELECT * FROM pgtle.uninstall_update_path('pg_tle_test', '0.1', '0.2');
```

pgtle.uninstall_update_path_if_exists

uninstall_update_path_if_exists 関数は、指定された更新パスを TLE 拡張機能から削除するという点で uninstall_update_path に似ています。ただし、更新パスが存在しない場合、この関数はエラーメッセージを表示しません。代わりに、関数は false を返します。

関数プロトタイプ

```
pgtle.uninstall_update_path_if_exists(extname text, fromvers text, tovers text)
```

ロール

pgtle_admin

引数

- extname – TLE 拡張機能の名前。この値は CREATE EXTENSION を呼び出すときに使用されます。
- fromvers – 更新パスで使用されている TLE 拡張機能のソースバージョン。
- tovers – 更新パスで使用されている TLE 拡張機能の送信先バージョン。

出力

- true – 関数は TLE 拡張機能のパスを正常に更新しました。

- false – この関数は TLE 拡張機能のパスを更新できませんでした。

使用例

```
SELECT * FROM pgtle.uninstall_update_path_if_exists('pg_tle_test', '0.1', '0.2');
```

pgtle.unregister_feature

unregister_feature 関数は、フックなどの pg_tle 機能を使用するために登録された関数を削除する方法を提供します。機能の登録については、「pgtle.register_feature」を参照してください。

関数プロトタイプ

```
pgtle.unregister_feature(proc regproc, feature pg_tle_features)
```

ロール

pgtle_admin

引数

- proc – pg_tle 機能に登録するストアド関数の名前。
- feature – 関数に登録する pg_tle 機能の名前。例えば、passcheck は、お客様が開発した、信頼できる言語拡張機能で使用するために登録できる機能です。詳細については、「パスワードチェックフック (passcheck)」を参照してください。

出力

なし。

使用例

```
SELECT * FROM pgtle.unregister_feature('pw_hook', 'passcheck');
```

pgtle.unregister_feature_if_exists

unregister_feature 関数は、フックなどの pg_tle 機能を使用するために登録された関数を削除する方法を提供します。詳細については、「TLE 拡張機能で PostgreSQL フックを使用する」を

参照してください。機能を正常に登録解除すると true を返します。機能が登録されていない場合は false を返します。

TLE 拡張機能の pg_tle 機能の登録の詳細については、「pgtle.register_feature」を参照してください。

関数プロトタイプ

```
pgtle.unregister_feature_if_exists('proc regproc', 'feature pg_tle_features')
```

ロール

pgtle_admin

引数

- proc – pg_tle 機能を含めるように登録されたストアド関数の名前。
- feature - 信頼できる言語拡張機能に登録された pg_tle 機能の名前。

出力

次のように false または true を返します。

- true - 関数は拡張機能から機能を正常に登録解除しました。
- false – 関数は TLE 拡張機能から機能を登録解除できませんでした。

使用例

```
SELECT * FROM pgtle.unregister_feature_if_exists('pw_hook', 'passcheck');
```

Trusted Language Extensions for PostgreSQL のフックリファレンス

Trusted Language Extensions for PostgreSQL は PostgreSQL フックをサポートしています。フックは、PostgreSQL のコア機能を拡張するために開発者が利用できる内部コールバックメカニズムです。フックを使用することで、デベロッパーはさまざまなデータベースオペレーション中に使用する独自の関数やプロシージャを実装できるため、PostgreSQL の動作を何らかの方法で変更できます。例えば、passcheck フックを使用して、ユーザー (ロール) のパスワードを作成または変更する際に提供されたパスワードを PostgreSQL がどのように処理するかをカスタマイズできます。

TLE 拡張機能で利用できるフックについては、次のドキュメントを参照してください。

トピック

- パスワードチェックフック (passcheck)

パスワードチェックフック (passcheck)

passcheck フックは、以下の SQL コマンドと psql メタコマンドのパスワードチェックプロセス時の PostgreSQL の動作をカスタマイズするために使用されます。

- CREATE ROLE *username* ...PASSWORD – 詳細については、PostgreSQL のドキュメントの「CREATE ROLE」を参照してください。
- ALTER ROLE *username*...PASSWORD – 詳細については、PostgreSQL のドキュメントの「ALTER ROLE」を参照してください。
- \password *username* – このインタラクティブな psql メタコマンドは、ALTER ROLE ... PASSWORD 構文を透過的に使用する前にパスワードをハッシュすることで、指定されたユーザーのパスワードを安全に変更します。このメタコマンドは ALTER ROLE ... PASSWORD コマンドの安全なラッパーであるため、フックは psql メタコマンドの動作に適用されます。

例については、「パスワードチェックフックコードリスト」を参照してください。

関数プロトタイプ

```
passcheck_hook(username text, password text, password_type pgtle.password_types,
 valid_until timestamptz, valid_null boolean)
```

引数

passcheck フック関数は、次の引数を取ります。

- username – パスワードを設定するロール (ユーザー名) の名前 (テキスト)。
- password – プレーンテキストまたはハッシュ化されたパスワード。入力するパスワードは、password_type で指定されたタイプと一致する必要があります。
- password_type – パスワードの pgtle.password_type 形式を指定します。この形式は、以下のいずれかのオプションになります。
  - PASSWORD_TYPE_PLAINTEXT – プレーンテキストのパスワード。

  - PASSWORD_TYPE_MD5 – MD5 (メッセージダイジェスト 5) アルゴリズムを使用してハッシュ化されたパスワード。
  - PASSWORD_TYPE_SCRAM_SHA_256 – SCRAM-SHA-256 アルゴリズムを使用してハッシュ化されたパスワード。
- valid_until – パスワードが無効になる時間を指定します。この引数はオプションです。この引数を使用する場合は、時間を timestamptz 値で指定します。
- valid_null – このブール値が true に設定されている場合、valid_until オプションは NULL に設定されます。

構成

この pgtle.enable_password_check 関数は、パスチェックフックが有効かどうかを制御します。パスチェックフックには 3 種類の設定があります。

- off – passcheck パスワードチェックフックをオフにします。これは、デフォルト値です。
- on – passcode パスワードチェックフックをオンにして、パスワードとテーブルを照合します。
- require – パスワードチェックフックを定義する必要があります。

使用に関する注意事項

passcheck フックをオンまたはオフにするには、Aurora PostgreSQL DB クラスターのライターインスタンス、のカスタム DB パラメータグループを変更する必要があります。

Linux、macOS、Unix の場合:

```
aws rds modify-db-parameter-group \
    --region aws-region \
    --db-parameter-group-name your-custom-parameter-group \
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

Windows の場合:

```
aws rds modify-db-parameter-group ^
    --region aws-region ^
    --db-parameter-group-name your-custom-parameter-group ^
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

Amazon Aurora PostgreSQL のリファレンス

トピック

EBCDIC やその他のメインフレーム移行のための Aurora PostgreSQL 照合順序

メインフレームアプリケーションを AWS などの新しいプラットフォームに移行する場合は、アプリケーションの動作が維持されることが理想的です。新しいプラットフォームでアプリケーションの動作をメインフレームと同じ状態に保つには、移行したデータを同じ照合順序ルールとソートルールを使用して照合する必要があります。例えば、多くの Db2 移行ソリューションでは、NULL 値を u0180 (Unicode の位置 0180) にシフトするため、これらの照合順序では u0180 が最初にソートされます。これは、照合順序がメインフレームソースとどのように異なるか、また元の EBCDIC 照合順序にうまく割り当てられる照合順序を選択する必要がある理由の一例です。

Aurora PostgreSQL 14.3 以降のバージョンでは、多くの ICU や EBCDIC 照合順序を提供しており、AWS Mainframe Modernization サービスを使用した AWS への移行などをサポートしています。このサービスの詳細については、「AWS Mainframe Modernization とは」を参照してください。

次の表は、Aurora PostgreSQL が提供する照合順序を示しています。これらの照合順序では EBCDIC の規則に準拠しており、メインフレームアプリケーションが AWS でもメインフレーム環境と同様に機能することを保証しています。照合順序名には、関連するコードページ (cpnnnn) が含まれているため、メインフレームソースに適した照合順序を選択できます。例えば、`en-US-cp037-x-icu` を使用して。コードページ 037 を使用するメインフレームアプリケーションから生成された EBCDIC データの照合動作を実現します。

EBCDIC 照合順序	AWS Blu Age 照合順序	AWS Micro Focus 照合順序
da-DK-cp1142-x-icu	da-DK-cp1142b-x-icu	da-DK-cp1142m-x-icu
da-DK-cp277-x-icu	da-DK-cp277b-x-icu	–

EBCDIC 照合順序	AWS Blu Age 照合順序	AWS Micro Focus 照合順序
de-DE-cp1141-x-icu	de-DE-cp1141b-x-icu	de-DE-cp1141m-x-icu
de-DE-cp273-x-icu	de-DE-cp273b-x-icu	–
en-GB-cp1146-x-icu	en-GB-cp1146b-x-icu	en-GB-cp1146m-x-icu
en-GB-cp285-x-icu	en-GB-cp285b-x-icu	–
en-US-cp037-x-icu	en-US-cp037b-x-icu	–
en-US-cp1140-x-icu	en-US-cp1140b-x-icu	en-US-cp1140m-x-icu
es-ES-cp1145-x-icu	es-ES-cp1145b-x-icu	es-ES-cp1145m-x-icu
es-ES-cp284-x-icu	es-ES-cp284b-x-icu	–
fi-FI-cp1143-x-icu	fi-FI-cp1143b-x-icu	fi-FI-cp1143m-x-icu
fi-FI-cp278-x-icu	fi-FI-cp278b-x-icu	–
fr-FR-cp1147-x-icu	fr-FR-cp1147b-x-icu	fr-FR-cp1147m-x-icu
fr-FR-cp297-x-icu	fr-FR-cp297b-x-icu	–
it-IT-cp1144-x-icu	it-IT-cp1144b-x-icu	it-IT-cp1144m-x-icu
it-IT-cp280-x-icu	it-IT-cp280b-x-icu	–
nl-BE-cp1148-x-icu	nl-BE-cp1148b-x-icu	nl-BE-cp1148m-x-icu
nl-BE-cp500-x-icu	nl-BE-cp500b-x-icu	–

AWS Blu Age の詳細については、「AWS Mainframe Modernization ユーザーガイド」の「チュートリアル: Managed Runtime for AWS Blu Age」を参照してください。

AWS Micro Focus 使用の詳細については、「AWS Mainframe Modernization ユーザーガイド」の「Tutorial: Managed Runtime for Micro Focus」を参照してください。

PostgreSQL における照合順序の管理の詳細については、PostgreSQL のドキュメントの「照合順序のサポート」を参照してください。

Aurora PostgreSQL でサポートされる照合

照合は、データベースに保存されている文字列をソートして比較する方法を決定する一連のルールです。照合は、コンピュータシステムにおいて基本的な役割を果たし、オペレーティングシステムの一部として組み込まれています。照合は、言語に新しい文字が追加されたり、順序規則が変更されたりすると、時間の経過とともに変化します。

照合ライブラリは、照合の特定のルールとアルゴリズムを定義します。PostgreSQL で使用される最も一般的な照合ライブラリは GNU C (glibc) と Unicode 用の国際化コンポーネント (ICU) です。デフォルトでは、Aurora PostgreSQL は、マルチバイト文字シーケンスの Unicode 文字ソート順序を含む glibc 照合を使用します。

新しい Aurora PostgreSQL DB クラスターを作成すると、オペレーティングシステムで使用可能な照合がチェックされます。`CREATE DATABASE` コマンド `LC_COLLATE` および `LC_CTYPE` の PostgreSQL パラメーターは、照合順序を指定するために使用され、そのデータベースのデフォルトの照合となります。または、`CREATE DATABASE` で `LOCALE` パラメータを使用して、これらのパラメータを設定することもできます。これにより、データベース内の文字列のデフォルトの照合と、文字を文字、数字、または記号として分類する規則が決まります。列、インデックス、またはクエリで使用する照合を選択することもできます。

Aurora PostgreSQL は、照合をサポートするためにオペレーティングシステムの glibc ライブラリに依存しています。Aurora PostgreSQL インスタンスは、オペレーティングシステムの最新バージョンで定期的に更新されます。これらのアップデートには glibc ライブラリの新しいバージョンが含まれることがあります。ごくまれに、新しいバージョンの glibc で一部の文字のソート順序や照合順序が変更されるため、データのソート方法が変わったり、無効なインデックスエントリが生成されることがあります。更新中に照合のソート順序の問題が見つかった場合は、インデックスの再構築が必要になることがあります。

glibc の更新による影響を減らすために、Aurora PostgreSQL に独立したデフォルトの照合ライブラリが含まれるようになりました。この照合ライブラリは、Aurora PostgreSQL 14.6、13.9、12.13、11.18、およびそれ以降のマイナーバージョンリリースで利用できます。glibc 2.26-59.amzn2 と互換性があり、誤ったクエリ結果を防ぐためにソート順序が安定しています。

Aurora PostgreSQL 関数のリファレンス

Aurora PostgreSQL 互換エディション DB エンジンを実行する Aurora DB クラスターで使用できる Aurora PostgreSQL 関数のリストを次に示します。これらの Aurora PostgreSQL 関数は、スタン

ダードの PostgreSQL 関数に追加されています。スタンダード PostgreSQL 関数の詳細については、「」を参照してください。PostgreSQL — 関数と演算子。

概要

Aurora PostgreSQL を実行する Amazon RDS DB インスタンスでは、次の関数を使用できます。

- aurora_db_instance_identifier
- aurora_ccm_status
- aurora_global_db_instance_status
- aurora_global_db_status
- aurora_list_builtins
- aurora_replica_status
- aurora_stat_activity
- aurora_stat_backend_waits
- aurora_stat_bgwriter
- aurora_stat_database
- aurora_stat_dml_activity
- aurora_stat_get_db_commit_latency
- aurora_stat_logical_wal_cache
- aurora_stat_memctx_usage
- aurora_stat_optimized_reads_cache
- aurora_stat_plans
- aurora_stat_reset_wal_cache
- aurora_stat_statements
- aurora_stat_system_waits
- aurora_stat_wait_event
- aurora_stat_wait_type
- aurora_version
- aurora_volume_logical_start_lsn
- aurora_wait_report

aurora_db_instance_identifier

接続している DB インスタンスの名前をレポートします。

構文

```
aurora_db_instance_identifier()
```

引数

なし

戻り型

VARCHAR 文字列

使用に関する注意事項

この関数によって、データベースクライアントまたはアプリケーション接続用の Aurora PostgreSQL 互換エディションクラスターの DB インスタンスの名前を表示します。

この関数は、Aurora PostgreSQL バージョン 13.7、12.11、11.16、10.21、およびそれ以降のすべてのバージョンのリリースで使用できます。

例

次の例は、`aurora_db_instance_identifier` 関数の呼び出し結果を示しています。

```
=> SELECT aurora_db_instance_identifier();
aurora_db_instance_identifier
------------------------------
 test-my-instance-name
```

この関数の結果を `aurora_replica_status` 関数と結合することで、接続先の DB インスタンスの詳細を取得できます。aurora_replica_status 単独では、どの DB インスタンスを使用しているかを表示できません。以下の例のように指定します。

```
=> SELECT *
    FROM aurora_replica_status() rt,
```

```
          aurora_db_instance_identifier() di
    WHERE rt.server_id = di;
-[ RECORD 1 ]----------------------+-----------------------
server_id                          | test-my-instance-name
session_id                         | MASTER_SESSION_ID
durable_lsn                        | 88492069
highest_lsn_rcvd                   |
current_read_lsn                   |
cur_replay_latency_in_usec         |
active_txns                        |
is_current                         | t
last_transport_error               | 0
last_error_timestamp               |
last_update_timestamp              | 2022-06-03 11:18:25+00
feedback_xmin                      |
feedback_epoch                     |
replica_lag_in_msec                |
log_stream_speed_in_kib_per_second | 0
log_buffer_sequence_number         | 0
oldest_read_view_trx_id            |
oldest_read_view_lsn               |
pending_read_ios                   | 819
```

aurora_ccm_status

クラスターキャッシュマネージャーのステータスを表示します。

構文

```
aurora_ccm_status()
```

引数

なし。

戻り型

次の列を含む SETOF レコード。

- buffers_sent_last_minute – 過去 1 分間に、指定された読み込みに送信されたバッファの数。

- buffers_found_last_minute – 過去 1 分間に識別されたアクセス頻度の高いバッファの数。
- buffers_sent_last_scan - バッファキャッシュの最後の完全スキャン中に、指定された読み取りに送信されたバッファの数。
- buffers_found_last_scan - バッファキャッシュの最後の完全スキャン中に送信されたアクセス頻度の高いバッファの数。指定された読み取りに既にキャッシュされているバッファは送信されません。
- buffers_sent_current_scan – 現在のスキャン中に送信されたバッファの数。
- buffers_found_current_scan – 現在のスキャンで識別されたアクセス頻度の高いバッファの数。
- current_scan_progress – 現在のスキャン中に、これまでに訪れたバッファの数。

使用に関する注意事項

この関数を使用して、クラスターキャッシュ管理 (CCM) 機能を確認、およびモニタリングできます。この関数は、Aurora PostgreSQL DB クラスターで CCM が有効な場合にのみ機能します。この関数を使用するには、Aurora PostgreSQL DB クラスターの Write DB インスタンスに接続します。

Aurora PostgreSQL DB クラスターの CCM をオンにするには、apg_ccm_enabled をクラスターのカスタム DB クラスターパラメータグループで 1 に設定します。この方法については、「クラスターキャッシュ管理の設定」を参照してください。

Aurora PostgreSQL DB クラスターでクラスターキャッシュ管理が有効になるのは、クラスターに Aurora Reader インスタンスが以下のように構成されている場合です。

- Aurora Reader インスタンスは、クラスターの Writer インスタンスと同じ DB インスタンスクラスタイプとサイズを使用します。
- Aurora Reader インスタンスは、クラスターの Tier-0 として構成されています。クラスターに複数の Reader がある場合、これは唯一の Tier-0 リーダーです。

複数のリーダーを Tier-0 に設定すると、CCM が無効になります。CCM が無効の場合、この関数を呼び出すと、次のエラーメッセージが返されます。

```
ERROR: Cluster Cache Manager is disabled
```

PostgreSQL pg_buffercache 拡張を使用して、バッファキャッシュを分析することもできます。詳細については、PostgreSQL ドキュメントの「pg_buffercache」を参照してください。

詳細については、「Aurora PostgreSQL クラスターキャッシュ管理の概要」を参照してください。

例

次の例は、aurora_ccm_status 関数呼び出しの結果を示しています。この最初の例は、CCM の統計情報を示しています。

```
=> SELECT * FROM aurora_ccm_status();
 buffers_sent_last_minute | buffers_found_last_minute | buffers_sent_last_scan |
 buffers_found_last_scan | buffers_sent_current_scan | buffers_found_current_scan |
 current_scan_progress
--------------------------+---------------------------+------------------------
+-------------------------+---------------------------+----------------------------
+------------------------
                   2242000 |                   2242003 |               17920442 |
         17923410 |                  14098000 |                  14100964 |
   15877443
```

詳細を確認するには、以下のように拡張表示を使用できます。

```
\x
Expanded display is on.
SELECT *  FROM aurora_ccm_status();
[ RECORD 1 ]-----------------------+---------
buffers_sent_last_minute           | 2242000
buffers_found_last_minute          | 2242003
buffers_sent_last_scan             | 17920442
buffers_found_last_scan            | 17923410
buffers_sent_current_scan          | 14098000
buffers_found_current_scan         | 14100964
current_scan_progress              | 15877443
```

この例では、ウォームレートとウォームの割合を確認する方法を示します。

```
=> SELECT buffers_sent_last_minute * 8/60 AS warm_rate_kbps,
100 * (1.0-buffers_sent_last_scan/buffers_found_last_scan) AS warm_percent
FROM aurora_ccm_status ();
 warm_rate_kbps | warm_percent
----------------+--------------
 16523 |        100.0
```

aurora_global_db_instance_status

Aurora グローバル DB クラスター内のレプリカを含む、すべての Aurora インスタンスのステータスを表示します。

構文

```
aurora_global_db_instance_status()
```

引数

なし

戻り型

次の列を含む SETOF レコード。

- `server_id` – DB インスタンスの ID。
- `session_id` – セッションの一意の識別子。`MASTER_SESSION_ID` の値は、Writer (プライマリ) DB インスタンスを識別します。
- `aws_region` – このグローバル DB インスタンスが実行される AWS リージョン。リージョンのリストについては、「利用可能なリージョン」を参照してください。
- `durable_lsn` – ストレージで耐久性のあるログシーケンス番号 (LSN)。ログシーケンス番号 (LSN) は、データベーストランザクションログ内のレコードを識別する一意の連続番号です。LSN は、より大きな LSN が後のトランザクションを表すように順序付けられます。
- `highest_lsn_rcvd` — Writer DB インスタンスから DB インスタンスが受信した最も高い LSN。
- `feedback_epoch` - DB インスタンスがホットスタンバイ情報を生成するときに使用するエポック。ホットスタンバイとは、プライマリ DB が復旧モードまたはスタンバイモードのときに、接続とクエリをサポートする DB インスタンスのことです。ホットスタンバイ情報には、ホットスタンバイとして使用されている DB インスタンスに関するエポック (時点) やその他の詳細が含まれます。詳細については、PostgreSQL ドキュメントの「ホットスタンバイ」を参照してください。
- `feedback_xmin` – DB インスタンスで使用される最小の (最も古い) アクティブトランザクション ID。
- `oldest_read_view_lsn` - ストレージから読み取るために DB インスタンスが使用した最も古い LSN。

- `visibility_lag_in_msec` – この DB インスタンスが Writer DB インスタンスからどれだけ遅れているか (ミリ秒単位)。

使用に関する注意事項

この関数は、Aurora DB クラスターのレプリケーションの統計を表示します。この関数は、クラスター内の各 Aurora PostgreSQL DB インスタンスについて、グローバルデータベース設定に任意のクロスリージョンレプリカを含むデータの行を表示します。

この関数は、Aurora PostgreSQL DB クラスターまたは Aurora PostgreSQL グローバルデータベースのどのインスタンスからでも実行できます。この関数は、すべてのレプリカインスタンスの遅延に関する詳細を返します。

この関数 (`aurora_global_db_instance_status`) または `aurora_global_db_status` を使用した遅延のモニタリングの詳細については、「Aurora PostgreSQL ベースのグローバルデータベースのモニタリング」を参照してください。

Aurora グローバルデータベースの詳細については、「Amazon Aurora Global Database の概要」を参照してください。

Aurora グローバルデータベースの開始方法については、「Amazon Aurora Global Database のスタート方法」または「Amazon Aurora よくある質問」を参照してください。

例

この例は、クロスリージョンインスタンスの統計を示しています。

```
=> SELECT *
  FROM aurora_global_db_instance_status();
               server_id                |              session_id              |
  aws_region  | durable_lsn | highest_lsn_rcvd | feedback_epoch | feedback_xmin |
 oldest_read_view_lsn | visibility_lag_in_msec
----------------------------------------+--------------------------------------
+--------------+-------------+------------------+----------------+---------------
+----------------------+------------------------
 db-119-001-instance-01                  | MASTER_SESSION_ID                    | eu-
west-1    |  2534560273 |           [NULL] |         [NULL] |        [NULL] |
     [NULL] |                [NULL]
 db-119-001-instance-02                  | 4ecff34d-d57c-409c-ba28-278b31d6fc40 | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560266 |                     6
```

```
 db-119-001-instance-03                   | 3e8a20fc-be86-43d5-95e5-bdf19d27ad6b | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560266 |                      6
 db-119-001-instance-04                   | fc1b0023-e8b4-4361-bede-2a7e926cead6 | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560254 |                     23
 db-119-001-instance-05                   | 30319b74-3f08-4e13-9728-e02aa1aa8649 | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560254 |                     23
 db-119-001-global-instance-1             | a331ffbb-d982-49ba-8973-527c96329c60 | eu-
central-1 |  2534560254 |       2534560266 |              0 |      19669196 |
 2534560247 |                    996
 db-119-001-global-instance-1             | e0955367-7082-43c4-b4db-70674064a9da | eu-
west-2    |  2534560254 |       2534560266 |              0 |      19669196 |
 2534560247 |                     14
 db-119-001-global-instance-1-eu-west-2a  | 1248dc12-d3a4-46f5-a9e2-85850491a897 | eu-
west-2    |  2534560254 |       2534560266 |              0 |      19669196 |
 2534560247 |                      0
```

この例は、グローバルレプリカの遅延をミリ秒単位で確認する方法を示します。

```
=> SELECT CASE
          WHEN 'MASTER_SESSION_ID' = session_id THEN 'Primary'
         ELSE 'Secondary'
      END AS global_role,
      aws_region,
      server_id,
      visibility_lag_in_msec
  FROM aurora_global_db_instance_status()
  ORDER BY 1, 2, 3;
   global_role |  aws_region  |               server_id                |
 visibility_lag_in_msec
-------------+-------------+----------------------------------------
+------------------------
 Primary     | eu-west-1    | db-119-001-instance-01                 |
 [NULL]
 Secondary   | eu-central-1 | db-119-001-global-instance-1           |
     13
 Secondary   | eu-west-1    | db-119-001-instance-02                 |
     10
 Secondary   | eu-west-1    | db-119-001-instance-03                 |
      9
```

```
 Secondary   | eu-west-1    | db-119-001-instance-04                  |
      2
 Secondary   | eu-west-1    | db-119-001-instance-05                  |
     18
 Secondary   | eu-west-2    | db-119-001-global-instance-1            |
     14
 Secondary   | eu-west-2    | db-119-001-global-instance-1-eu-west-2a |
     13
```

この例では、データベースのグローバル設定から、AWS リージョン ごとの最小、最大、平均の遅延時間を確認する方法を示しています。

```
=> SELECT 'Secondary' global_role,
       aws_region,
       min(visibility_lag_in_msec) min_lag_in_msec,
       max(visibility_lag_in_msec) max_lag_in_msec,
       round(avg(visibility_lag_in_msec),0) avg_lag_in_msec
  FROM aurora_global_db_instance_status()
 WHERE aws_region NOT IN (SELECT   aws_region
                            FROM aurora_global_db_instance_status()
                           WHERE session_id='MASTER_SESSION_ID')
                        GROUP BY aws_region
UNION ALL
SELECT  'Primary' global_role,
        aws_region,
        NULL,
        NULL,
        NULL
    FROM aurora_global_db_instance_status()
   WHERE session_id='MASTER_SESSION_ID'
ORDER BY 1, 5;
 global_role |  aws_region  | min_lag_in_msec | max_lag_in_msec | avg_lag_in_msec
-------------+--------------+-----------------+-----------------+-----------------
 Primary     | eu-west-1    |          [NULL] |          [NULL] |          [NULL]
 Secondary   | eu-central-1 |             133 |             133 |             133
 Secondary   | eu-west-2    |               0 |             495 |             248
```

aurora_global_db_status

Aurora グローバルデータベースの遅延のさまざまな側面に関する情報を表示します。具体的には、基盤となる Aurora ストレージの遅延 (いわゆる耐久性の遅延) と目標復旧時点 (RPO) 間の遅延などです。

構文

```
aurora_global_db_status()
```

引数

なし。

戻り型

次の列を含む SETOF レコード。

- aws_region – この DB クラスターがある AWS リージョン。エンジン別の AWS リージョン の一覧については、「リージョンとアベイラビリティーゾーン」を参照してください。
- highest_lsn_written – この DB クラスターに現在存在するログシーケンス番号 (LSN) の最大値。ログシーケンス番号 (LSN) は、データベーストランザクションログ内のレコードを識別する一意の連続番号です。LSN は、より大きな LSN が後のトランザクションを表すように順序付けられます。
- durability_lag_in_msec – セカンダリ DB クラスターの highest_lsn_written とプライマリ DB クラスターの highest_lsn_written とのタイムスタンプ値の差。-1 の値は、Aurora グローバルデータベースのプライマリ DB クラスターを識別します。
- rpo_lag_in_msec – 目標復旧時点 (RPO)の遅延。RPO 遅延とは、最近のユーザートランザクション COMMIT が、Aurora グローバルデータベースのプライマリ DB クラスターに保存された後、セカンダリ DB クラスターに保存されるまでにかかる時間です。-1 の値は、プライマリ DB クラスターを表します (したがって、遅延は関係ありません)。

  簡単に言えば、このメトリクスは、Aurora グローバルデータベース内の各 Aurora PostgreSQL DB クラスターの目標復旧時点、つまり、障害が発生した場合に失われる可能性のあるデータの量を計算します。遅延と同様に、RPO は時間単位で測定されます。
- last_lag_calculation_time – durability_lag_in_msec と rpo_lag_in_msec に対して値が最後に計算された時刻を指定するタイムスタンプ。1970-01-01 00:00:00+00 のような時間値は、これがプライマリ DB クラスターであることを意味します。
- feedback_epoch – セカンダリ DB クラスターがホットスタンバイ情報を生成するときに使用するエポック。ホットスタンバイとは、プライマリ DB が復旧モードまたはスタンバイモードのときに、接続とクエリをサポートする DB インスタンスのことです。ホットスタンバイ情報には、ホットスタンバイとして使用されている DB インスタンスに関するエポック (時点) やその他の詳細が

含まれます。詳細については、PostgreSQL ドキュメントの「ホットスタンバイ」を参照してください。

- `feedback_xmin` – セカンダリ DB クラスターで使用される最小 (最も古い) のアクティブトランザクション ID。

使用に関する注意事項

この関数は、Aurora グローバルデータベースのレプリケーションの統計を表示します。Aurora PostgreSQL グローバルデータベース内の各 DB クラスターに対して 1 つの行が表示されます。この関数は、Aurora PostgreSQL グローバルデータベースのどのインスタンスからでも実行できます。

目に見えるデータ遅延である Aurora グローバルデータベースレプリケーションの遅延を評価するには、「aurora_global_db_instance_status」を参照してください。

`aurora_global_db_status` や `aurora_global_db_instance_status` を使用して Aurora グローバルデータベースラグをモニタリングする方法の詳細については、「Aurora PostgreSQL ベースのグローバルデータベースのモニタリング」を参照してください。Aurora グローバルデータベースの詳細については、「Amazon Aurora Global Database の概要」を参照してください。

例

この例は、クロスリージョンストレージの統計情報を表示する方法を示します。

```
=> SELECT CASE
          WHEN '-1' = durability_lag_in_msec THEN 'Primary'
          ELSE 'Secondary'
       END AS global_role,
       *
  FROM aurora_global_db_status();
 global_role | aws_region | highest_lsn_written | durability_lag_in_msec |
 rpo_lag_in_msec | last_lag_calculation_time  | feedback_epoch | feedback_xmin
-------------+------------+---------------------+------------------------
+-----------------+-----------------------------+----------------+---------------
 Primary     | eu-west-1  |           131031557 |                     -1 |
 -1 | 1970-01-01 00:00:00+00     |             0 |              0
 Secondary   | eu-west-2  |           131031554 |                    410 |
  0 | 2021-06-01 18:59:36.124+00 |             0 |         12640
 Secondary   | eu-west-3  |           131031554 |                    410 |
  0 | 2021-06-01 18:59:36.124+00 |             0 |         12640
```

aurora_list_builtins

使用可能なすべての Aurora PostgreSQL 組み込み関数と、簡単な説明および関数の詳細を一覧表示します。

構文

```
aurora_list_builtins()
```

引数

なし

戻り型

SETOF レコード

例

次の例は、aurora_list_builtins 関数呼び出しの結果を示しています。

```
=> SELECT *
FROM aurora_list_builtins();

               Name                | Result data type |                    Argument data
 types                     | Type | Volatility |  Parallel  | Security |
             Description
-----------------------------------+------------------
+-----------------------------------------------------------+------+------------
+------------+----------
+-------------------------------------------------------------------------
 aurora_version                    | text             |
                           | func | stable     | safe       | invoker  | Amazon Aurora
 PostgreSQL-Compatible Edition version string
 aurora_stat_wait_type             | SETOF record     | OUT type_id smallint, OUT
 type_name text                 | func | volatile   | restricted | invoker  | Lists all
 supported wait types
 aurora_stat_wait_event            | SETOF record     | OUT type_id smallint, OUT
 event_id integer, OUT event_na.| func | volatile   | restricted | invoker  | Lists all
 supported wait events
                                   |                  |.me text
                           |      |            |            |          |
```

```
aurora_list_builtins              | SETOF record     | OUT "Name" text, OUT "Result
data type" text, OUT "Argum.| func | stable     | safe       | invoker  | Lists all
Aurora built-in functions
                                  |                  |.ent data types" text, OUT
"Type" text, OUT "Volatility" .|      |            |            |          |
                                  |                  |.text, OUT "Parallel" text, OUT
"Security" text, OUT "Des.|      |            |            |          |
                                  |                  |.cription" text
                         |      |            |            |          |
.
.
.
aurora_stat_file                  | SETOF record     | OUT filename text, OUT
allocated_bytes bigint, OUT used_.| func | stable     | safe       | invoker  | Lists
all files present in Aurora storage
                                  |                  |.bytes bigint
                         |      |            |            |          |
aurora_stat_get_db_commit_latency | bigint           | oid
                          | func | stable     | restricted | invoker  | Per DB commit
latency in microsecs
```

aurora_replica_status

すべての Aurora PostgreSQL リーダーノードのステータスを表示します。

構文

```
aurora_replica_status()
```

引数

なし

戻り型

次の列を含む SETOF レコード。

- `server_id` - DB インスタンス ID (識別子)。
- `session_id` — 現在のセッションの一意の識別子。プライマリインスタンスおよびリーダーインスタンスについて次のように返されます。

  - プライマリインスタンスについては、session_id は常に `MASTER_SESSION_ID' です。
  - リーダーインスタンスについては、session_id は常にリーダーインスタンスの UUID (ユニバーサル一意識別子) です。
- durable_lsn — ストレージに保存されているログシーケンス番号 (LSN)。
  - プライマリボリュームの場合、現在有効なプライマリボリューム耐久性 LSN (VDL)。
  - セカンダリボリュームの場合、セカンダリが正常に適用されたプライマリのVDL。

> **Note**
>
> ログシーケンス番号 (LSN) は、データベーストランザクションログ内のレコードを識別する一意の連続番号です。LSN は、より大きな LSN がシーケンスの後の方で発生したトランザクションを表すように順序付けられます。

- highest_lsn_rcvd — ライター DB インスタンスから DB インスタンスが受信した最も高い (最新の) LSN。
- current_read_lsn — すべてのリーダーに適用された最新のスナップショットの LSN。
- cur_replay_latency_in_usec — セカンダリでのログの再生に要すると推測されるマイクロ秒数。
- active_txns - 現在アクティブなトランザクションの数。
- is_current - 使用されません。
- last_transport_error — 前回のレプリケーションエラーコード。
- last_error_timestamp — 最後のレプリケーションエラーのタイムスタンプ。
- last_update_timestamp — レプリカステータスの最終更新のタイムスタンプ。Aurora PostgreSQL 13.9 以降では、接続先の DB インスタンスの last_update_timestamp 値は NULL に設定されます。
- feedback_xmin — レプリカのホットスタンバイ feedback_xmin。DB インスタンスで使用される最小の (最も古い) アクティブトランザクション ID。
- feedback_epoch - DB インスタンスがホットスタンバイ情報を生成するときに使用するエポック。
- replica_lag_in_msec — リーダーインスタンスがライターインスタンスより遅れている時間 (ミリ秒単位)。
- log_stream_speed_in_kib_per_second — キロバイト/秒単位のログストリームスループット。

- log_buffer_sequence_number — ログバッファのシーケンス番号。
- oldest_read_view_trx_id - 使用されません。
- oldest_read_view_lsn - ストレージから読み取るために DB インスタンスが使用した最も古い LSN。
- pending_read_ios — レプリカで保留中の未処理のページ読み取り。
- read_ios — レプリカでのページ読み取りの総数。
- iops - 使用されません。
- cpu — レプリカプロセスによる CPU 使用率。これは、インスタンスではなく、プロセスによるによる CPU 使用率であることに注意してください。インスタンスによる CPU 使用率の詳細については、「Amazon Aurora のインスタンスレベルのメトリクス」を参照してください。

使用に関する注意事項

aurora_replica_status 関数は、Aurora PostgreSQL DB クラスターのレプリカステータスマネージャーから値を返します。この関数を使用して、Aurora DB クラスター内のすべての DB インスタンスのメトリックを含め、Aurora PostgreSQL DB クラスターのレプリケーションのステータスに関する情報を取得できます。例えば、次の操作を実行できます。

- Aurora PostgreSQL DB クラスター内のインスタンスのタイプ (ライター、リーダー) に関する情報を取得する - この情報を取得するには、次の列の値を確認します。
  - server_id - インスタンスの作成時に指定したインスタンスの名前が含まれます。プライマリ (ライター) インスタンスの場合など、名前は、通常、Aurora PostgreSQL DB クラスター用に作成した名前に -instance-1 を付加することによって自動的に作成されます。
  - session_id — session_id フィールドは、インスタンスがリーダーかライターかを示します。ライターインスタンスの場合、session_id は常に "MASTER_SESSION_ID" に設定されます。リーダーインスタンスの場合、session_id は特定のリーダーの UUID に設定されます。
- レプリカのラグなど、一般的なレプリケーションの問題を診断する — レプリカのラグは、リーダーインスタンスのページキャッシュがライターインスタンスのページキャッシュより遅れている時間 (ミリ秒) です。このラグは、Amazon Aurora でのレプリケーション で説明されているように、Aurora クラスターが非同期レプリケーションを使用しているために発生します。これは、この関数によって返される結果の replica_lag_in_msec 列に表示されます。ラグは、スタンバイサーバーでのリカバリとの競合によってクエリがキャンセルされた場合にも発生することがあります。pg_stat_database_conflicts() をチェックして、このような競合がレプリカラグを引き起こしている (または引き起こしていない) ことを確認できます。詳細については、

『PostgreSQL ドキュメント』の「統計コレクター」を参照してください。高可用性とレプリケーションの詳細については、「Amazon Aurora よくある質問」を参照してください。

Amazon CloudWatch は 時間の経過とともに `replica_lag_in_msec` の結果を `AuroraReplicaLag` メトリクスとして保存します。Aurora 向け CloudWatch メトリクスの使用については、「Amazon CloudWatch を使用した Amazon Aurora メトリクスのモニタリング」を参照してください。

Aurora リードレプリカおよび再起動のトラブルシューティングの詳細については、AWS Support センター の「Amazon Aurora リードレプリカが遅れて再起動したのはなぜですか。」を参照してください。

例

次の例は、Aurora PostgreSQL DB クラスター内のすべてのインスタンスのレプリケーションステータスを取得する方法を示しています。

```
=> SELECT *
FROM aurora_replica_status();
```

次の例は、`docs-lab-apg-main` Aurora PostgreSQL DB クラスター内のライターインスタンスを示しています。

```
=> SELECT server_id,
    CASE
        WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
        ELSE 'reader'
    END AS instance_role
FROM aurora_replica_status()
WHERE session_id = 'MASTER_SESSION_ID';
        server_id       | instance_role
------------------------+---------------
 db-119-001-instance-01 | writer
```

次の例では、クラスター内のすべてのリーダーインスタンスをリストします。

```
=> SELECT server_id,
    CASE
        WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
        ELSE 'reader'
    END AS instance_role
```

```
FROM aurora_replica_status()
WHERE session_id <> 'MASTER_SESSION_ID';
        server_id       | instance_role
------------------------+---------------
db-119-001-instance-02  | reader
db-119-001-instance-03  | reader
db-119-001-instance-04  | reader
db-119-001-instance-05  | reader
(4 rows)
```

次の例では、すべてのインスタンス、各インスタンスがライターより遅れている時間、および最後の更新からの経過時間をリストします。

```
=> SELECT server_id,
    CASE
        WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
        ELSE 'reader'
    END AS instance_role,
    replica_lag_in_msec AS replica_lag_ms,
    round(extract (epoch FROM (SELECT age(clock_timestamp(), last_update_timestamp))) *
 1000) AS last_update_age_ms
FROM aurora_replica_status()
ORDER BY replica_lag_in_msec NULLS FIRST;
       server_id        | instance_role | replica_lag_ms | last_update_age_ms
------------------------+---------------+----------------+--------------------
 db-124-001-instance-03 | writer        |         [NULL] |               1756
 db-124-001-instance-01 | reader        |             13 |               1756
 db-124-001-instance-02 | reader        |             13 |               1756
(3 rows)
```

aurora_stat_activity

サーバープロセスごとに 1 行を返し、そのプロセスの現在のアクティビティに関連する情報を表示します。

構文

```
aurora_stat_activity();
```

引数

なし

戻り型

サーバープロセスごとに 1 行を返します。pg_stat_activity 列に加えて、次のフィールドが追加されます。

- planid - 計画識別子

使用に関する注意事項

現在のクエリ実行計画を示す追加の plan_id 列を含む同じ列を返す pg_stat_activity の補足ビュー。

ビューが plan_id を返すには、aurora_compute_plan_id を有効にする必要があります。

この関数は、Aurora PostgreSQL バージョン 14.10、15.5、およびそれ以降のすべてのバージョンで使用できます。

例

以下のクエリ例では、上位負荷を query_id と plan_id 別に集計しています。

```
db1=# select count(*), query_id, plan_id
db1-# from aurora_stat_activity() where state = 'active'
db1-# and pid <> pg_backend_pid()
db1-# group by query_id, plan_id
db1-# order by 1 desc;

count |  query_id             |  plan_id
-------+-----------------------+-------------
 11    | -5471422286312252535 | -2054628807
 3     | -6907107586630739258 | -815866029
 1     | 5213711845501580017  |  300482084
(3 rows)
```

query_id に使用される計画が変更されると、新しい plan_id が aurora_stat_activity によって報告されます。

```
count  |  query_id            |  plan_id
-------+----------------------+-------------
 10    | -5471422286312252535 | 1602979607
 1     | -6907107586630739258 | -1809935983
 1     | -2446282393000597155 | -207532066
(3 rows)
```

aurora_stat_backend_waits

特定のバックエンドプロセスの待機アクティビティの統計を表示します。

構文

```
aurora_stat_backend_waits(pid)
```

引数

pid – バックエンドプロセスの ID。プロセス ID は、pg_stat_activity ビューを使用して取得できます。

戻り型

次の列を含む SETOF レコード。

- type_id – 待機イベントのタイプを示す数値。いくつか例を挙げると、軽量ロック (LWLock) の場合は 1、ロックの場合は 3、またはクライアントセッションの場合は 6 などです。これらの値は、例 に示すように、この関数の結果を aurora_stat_wait_type 関数の列と結合することで意味を持ちます。
- event_id – 待機イベントの識別番号。この値を aurora_stat_wait_event の列と結合して、意味のあるイベント名を取得します。
- waits – 指定したプロセス ID に累積された待機数のカウント。
- wait_time – ミリ秒単位の待機時間。

使用に関する注意事項

この関数を使用して、接続を開いた後に発生した特定のバックエンド (セッション) 待機イベントを分析できます。待機イベントの名前とタイプに関するより意味のある情報を得るには、例に示すよう

に JOIN を使用して、この関数 aurora_stat_wait_type と aurora_stat_wait_event を組み合わせることができます。

例

この例では、バックエンドプロセス ID 3027 のすべての待機、タイプ、イベント名を示しています。

```
=> SELECT type_name, event_name, waits, wait_time
        FROM aurora_stat_backend_waits(3027)
NATURAL JOIN aurora_stat_wait_type()
NATURAL JOIN aurora_stat_wait_event();
type_name |       event_name       | waits | wait_time
----------+------------------------+-------+------------
 LWLock    | ProcArrayLock          |     3 |         27
 LWLock    | wal_insert             |   423 |      16336
 LWLock    | buffer_content         | 11840 |    1033634
 LWLock    | lock_manager           | 23821 |    5664506
 Lock      | tuple                  | 10258 |  152280165
 Lock      | transactionid          | 78340 | 1239808783
 Client    | ClientRead             | 34072 |   17616684
 IO        | ControlFileSyncUpdate  |     2 |          0
 IO        | ControlFileWriteUpdate |     4 |         32
 IO        | RelationMapRead        |     2 |        795
 IO        | WALWrite               | 36666 |      98623
 IO        | XactSync               |  4867 |    7331963
```

この例では、すべてのアクティブなセッションの現在および累積の待機タイプと待機イベントを示しています (pg_stat_activity state <> 'idle')。ただし、関数を呼び出している現在のセッションは表示されません (pid <> pg_backend_pid())。

```
=> SELECT a.pid,
            a.usename,
            a.app_name,
            a.current_wait_type,
            a.current_wait_event,
            a.current_state,
            wt.type_name AS wait_type,
            we.event_name AS wait_event,
            a.waits,
            a.wait_time
        FROM (SELECT pid,
                     usename,
```

```
                    left(application_name,16) AS app_name,
                    coalesce(wait_event_type,'CPU') AS current_wait_type,
                    coalesce(wait_event,'CPU') AS current_wait_event,
                    state AS current_state,
                    (aurora_stat_backend_waits(pid)).*
                FROM pg_stat_activity
               WHERE pid <> pg_backend_pid()
                 AND state <> 'idle') a
NATURAL JOIN aurora_stat_wait_type() wt
NATURAL JOIN aurora_stat_wait_event() we;
  pid  | usename  | app_name | current_wait_type | current_wait_event | current_state |
 wait_type |       wait_event       | waits | wait_time
-------+----------+----------+-------------------+--------------------+---------------
+-----------+------------------------+-------+-----------
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | wal_insert             |  1937 |     29975
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | buffer_content         | 22903 |    760498
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | lock_manager           | 10012 |    223207
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 Lock      | tuple                  | 20315 |  63081529
 .
 .
 .
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 IO        | WALWrite               | 93293 |    237440
 30099 | postgres | pgbench  | Lock              | transactionid      | active        |
 IO        | XactSync               | 13010 |  19525143
 30100 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | ProcArrayLock          |     6 |        53
 30100 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | wal_insert             |  1913 |     25450
 30100 | postgres | pgbench  | Lock              | transactionid      | active        |
 LWLock    | buffer_content         | 22874 |    778005
 .
 .
 .
 30109 | postgres | pgbench  | IO                | XactSync           | active        |
 LWLock    | ProcArrayLock          |     3 |        71
 30109 | postgres | pgbench  | IO                | XactSync           | active        |
 LWLock    | wal_insert             |  1940 |     27741
 30109 | postgres | pgbench  | IO                | XactSync           | active        |
 LWLock    | buffer_content         | 22962 |    776352
```

```
 30109 | postgres | pgbench  | IO               | XactSync          | active        |
 LWLock    | lock_manager           |  9879 |    218826
 30109 | postgres | pgbench  | IO               | XactSync          | active        |
 Lock      | tuple                  | 20401 |  63581306
 30109 | postgres | pgbench  | IO               | XactSync          | active        |
 Lock      | transactionid          | 50769 | 211645008
 30109 | postgres | pgbench  | IO               | XactSync          | active        |
 Client    | ClientRead             | 89901 |  44192439
```

この例では、すべてのアクティブなセッションについて、現在および上位 3 つの累積の待機タイプと待機イベントを示しています (pg_stat_activity state <> 'idle')。現在のセッションは除きます (pid <>pg_backend_pid( ))。

```
=> SELECT top3.*
       FROM (SELECT a.pid,
                    a.usename,
                    a.app_name,
                    a.current_wait_type,
                    a.current_wait_event,
                    a.current_state,
                    wt.type_name AS wait_type,
                    we.event_name AS wait_event,
                    a.waits,
                    a.wait_time,
                    RANK() OVER (PARTITION BY pid ORDER BY a.wait_time DESC)
               FROM (SELECT pid,
                            usename,
                            left(application_name,16) AS app_name,
                            coalesce(wait_event_type,'CPU') AS current_wait_type,
                            coalesce(wait_event,'CPU') AS current_wait_event,
                            state AS current_state,
                            (aurora_stat_backend_waits(pid)).*
                       FROM pg_stat_activity
                      WHERE pid <> pg_backend_pid()
                        AND state <> 'idle') a
       NATURAL JOIN aurora_stat_wait_type() wt
       NATURAL JOIN aurora_stat_wait_event() we) top3
 WHERE RANK <=3;
  pid  | usename  | app_name | current_wait_type | current_wait_event | current_state |
 wait_type |   wait_event    |  waits  | wait_time  | rank
-------+----------+----------+-------------------+--------------------+---------------
+-----------+-----------------+---------+------------+------
```

```
 20567 | postgres | psql     | CPU               | CPU                | active        |
 LWLock    | wal_insert      |   25000 |   67512003 |    1
 20567 | postgres | psql     | CPU               | CPU                | active        |
 IO        | WALWrite        | 3071758 |    1016961 |    2
 20567 | postgres | psql     | CPU               | CPU                | active        |
 IO        | BufFileWrite    |   20750 |     184559 |    3
 27743 | postgres | pgbench  | Lock              | transactionid      | active        |
 Lock      | transactionid   |  237350 | 1265580011 |    1
 27743 | postgres | pgbench  | Lock              | transactionid      | active        |
 Lock      | tuple           |   93641 |  341472318 |    2
 27743 | postgres | pgbench  | Lock              | transactionid      | active        |
 Client    | ClientRead      |  417556 |  204796837 |    3
 .
 .
 .
 27745 | postgres | pgbench  | IO                | XactSync           | active        |
 Lock      | transactionid   |  238068 | 1265816822 |    1
 27745 | postgres | pgbench  | IO                | XactSync           | active        |
 Lock      | tuple           |   93210 |  338312247 |    2
 27745 | postgres | pgbench  | IO                | XactSync           | active        |
 Client    | ClientRead      |  419157 |  207836533 |    3
 27746 | postgres | pgbench  | Lock              | transactionid      | active        |
 Lock      | transactionid   |  237621 | 1264528811 |    1
 27746 | postgres | pgbench  | Lock              | transactionid      | active        |
 Lock      | tuple           |   93563 |  339799310 |    2
 27746 | postgres | pgbench  | Lock              | transactionid      | active        |
 Client    | ClientRead      |  417304 |  208372727 |    3
```

aurora_stat_bgwriter

aurora_stat_bgwriterは、Optimized Reads キャッシュ書き込みに関する情報を表示する統計ビューです。

構文

```
aurora_stat_bgwriter()
```

引数

なし

戻り型

すべての pg_stat_bgwriter 列と以下の追加列を含む SETOF レコード。pg_stat_bgwriter 列の詳細については、「pg_stat_bgwriter」を参照してください。

この関数の統計情報は pg_stat_reset_shared("bgwriter") を使用してリセットできます。

- orcache_blks_written – 書き込まれた Optimized Reads キャッシュデータブロックの総数。
- orcache_blk_write_time – track_io_timing を有効にすると、Optimized Reads キャッシュデータブロックの書き込みにかかった合計時間をミリ秒単位で追跡します。詳細については、track_io_timing を参照してください。

使用に関する注意事項

この関数は、次の Aurora PostgreSQL バージョンで使用できます。

- 15.4 以降の 15 バージョン
- 14.9 以降の 14 バージョン

例

```
=> select * from aurora_stat_bgwriter();
-[ RECORD 1 ]-----------------+-----------
orcache_blks_written          | 246522
orcache_blk_write_time        | 339276.404
```

aurora_stat_database

pg_stat_database のすべての列を保持し、最後に新しい列を追加します。

構文

```
aurora_stat_database()
```

引数

なし

戻り型

すべての pg_stat_database 列と以下の追加列を含む SETOF レコード。pg_stat_database 列の詳細については、「pg_stat_database」を参照してください。

- storage_blks_read - このデータベースの Aurora ストレージから読み取られた共有ブロックの総数。
- orcache_blks_hit - このデータベース内の Optimized Reads キャッシュヒットの総数。
- local_blks_read - このデータベースで読み取られたローカルブロックの総数。
- storage_blk_read_time - track_io_timing を有効にすると、Aurora ストレージからのデータファイルブロックの読み取りにかかった合計時間をミリ秒単位で追跡します。それ以外の場合、値はゼロです。詳細については、track_io_timing を参照してください。
- local_blk_read_time - track_io_timing を有効にすると、ローカルデータファイルブロックの読み取りにかかった合計時間をミリ秒単位で追跡します。それ以外の場合、値は 0 です。詳細については、track_io_timing を参照してください。
- orcache_blk_read_time - track_io_timing を有効にすると、Optimized Reads キャッシュからデータファイルブロックを読み取るのにかかった合計時間をミリ秒単位で追跡します。それ以外の場合は 0 です。詳細については、track_io_timing を参照してください。

> **Note**
>
> blks_read の値は、storage_blks_read、orcache_blks_hit、local_blks_read の合計です。
> blk_read_time の値は、storage_blk_read_time、orcache_blk_read_time、local_blk_read_time の合計です。

使用に関する注意事項

この関数は、次の Aurora PostgreSQL バージョンで使用できます。

- 15.4 以降の 15 バージョン
- 14.9 以降の 14 バージョン

例

次の例は、すべての pg_stat_database 列を保持し、末尾に 6 つの新しい列を追加する方法を示しています。

```
=> select * from aurora_stat_database() where datid=14717;
-[ RECORD 1 ]------------+------------------------------
datid                    | 14717
datname                  | postgres
numbackends              | 1
xact_commit              | 223
xact_rollback            | 4
blks_read                | 1059
blks_hit                 | 11456
tup_returned             | 27746
tup_fetched              | 5220
tup_inserted             | 165
tup_updated              | 42
tup_deleted              | 91
conflicts                | 0
temp_files               | 0
temp_bytes               | 0
deadlocks                | 0
checksum_failures        |
checksum_last_failure    |
blk_read_time            | 3358.689
blk_write_time           | 0
session_time             | 1076007.997
active_time              | 3684.371
idle_in_transaction_time | 0
sessions                 | 10
sessions_abandoned       | 0
sessions_fatal           | 0
sessions_killed          | 0
stats_reset              | 2023-01-12 20:15:17.370601+00
orcache_blks_hit         | 425
orcache_blk_read_time    | 89.934
storage_blks_read        | 623
storage_blk_read_time    | 3254.914
local_blks_read          | 0
local_blk_read_time      | 0
```

aurora_stat_dml_activity

Aurora PostgreSQL クラスター内のデータベースに対する各タイプのデータ操作言語 (DML) オペレーションの累積アクティビティをレポートします。

構文

```
aurora_stat_dml_activity(database_oid)
```

引数

database_oid

Aurora PostgreSQL クラスター内のデータベースのオブジェクト ID (OID)。

戻り型

SETOF レコード

使用に関する注意事項

`aurora_stat_dml_activity` 関数は、PostgreSQL エンジン 11.6 以降と互換性がある Aurora PostgreSQL リリース 3.1 でのみ利用可能です。

この関数を、多数のデータベースを持つ Aurora PostgreSQL クラスターで使用して、どちらのデータベースの DML アクティビティが遅いか、または両方とも遅いかを特定します。

`aurora_stat_dml_activity` 関数は、SELECT、INSERT、UPDATE、DELETE オペレーションの実行回数と累積レイテンシーをマイクロ秒単位で返します。レポートには、成功した DML オペレーションのみが含まれます。

この統計をリセットするには、PostgreSQL 統計アクセス関数 `pg_stat_reset` を使用します。この統計が最後にリセットされた時刻をチェックするには、`pg_stat_get_db_stat_reset_time` 関数を使用します。PostgreSQL 統計アクセス関数の詳細については、PostgreSQL ドキュメントの統計コレクターを参照してください。

例

次の例は、接続データベース用に DML アクティビティ統計をレポートする方法を示しています。

```
--Define the oid variable from connected database by using \gset
```

```
=> SELECT oid,
          datname
     FROM pg_database
    WHERE datname=(select current_database()) \gset
=> SELECT *
     FROM aurora_stat_dml_activity(:oid);
select_count | select_latency_microsecs | insert_count | insert_latency_microsecs |
 update_count | update_latency_microsecs | delete_count | delete_latency_microsecs
--------------+--------------------------+--------------+--------------------------
+--------------+--------------------------+--------------+--------------------------
       178957 |                 66684115 |       171065 |                 28876649 |
    519538 |            1454579206167 |            1 |                    53027


-- Showing the same results with expanded display on
=> SELECT *
     FROM aurora_stat_dml_activity(:oid);
-[ RECORD 1 ]------------+-------------
select_count             | 178957
select_latency_microsecs | 66684115
insert_count             | 171065
insert_latency_microsecs | 28876649
update_count             | 519538
update_latency_microsecs | 1454579206167
delete_count             | 1
delete_latency_microsecs | 53027
```

次の例は、Aurora PostgreSQL クラスター内のすべてのデータベースの DML アクティビティ統計を示しています。このクラスターには、postgres および mydb という 2 つのデータベースがあります。コンマ区切りのリストは、select_count、select_latency_microsecs、insert_count、insert_latency_microsecs フィールドと一致しています。

Aurora PostgreSQL は、rdsadmin という名前のシステムデータベースを作成し、バックアップ、修復、ヘルスチェック、レプリケーションなどの管理オペレーションをサポートするために使用します。これらの DML オペレーションは、Aurora PostgreSQL クラスターには影響しません。

```
=> SELECT oid,
    datname,
    aurora_stat_dml_activity(oid)
    FROM pg_database;
oid  |    datname     |                   aurora_stat_dml_activity
```

```
-------+---------------
+-------------------------------------------------------------------
 14006 | template0      | (,,,,,,,)
 16384 | rdsadmin       | (2346623,1211703821,4297518,817184554,0,0,0,0)
     1 | template1      | (,,,,,,,)
 14007 | postgres       |
 (178961,66716329,171065,28876649,519538,1454579206167,1,53027)
 16401 | mydb           | (200246,64302436,200036,107101855,600000,83659417514,0,0)
```

次の例は、読みやすさを高めるために列で編成された、すべてのデータベースの DML アクティビティ統計を示しています。

```
SELECT db.datname,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 1), '()') AS select_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 2), '()') AS select_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 3), '()') AS insert_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 4), '()') AS insert_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 5), '()') AS update_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 6), '()') AS update_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 7), '()') AS delete_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 8), '()') AS delete_latency_microsecs
FROM  (SELECT datname,
              aurora_stat_dml_activity(oid) AS asdmla
         FROM pg_database
      ) AS db;

    datname     | select_count | select_latency_microsecs | insert_count |
 insert_latency_microsecs | update_count | update_latency_microsecs | delete_count |
 delete_latency_microsecs
----------------+--------------+--------------------------+--------------
+--------------------------+--------------+--------------------------+--------------
+--------------------------
 template0      |              |                          |              |
             |              |                          |              |
 rdsadmin       | 4206523      | 2478812333               | 7009414      | 1338482258
             | 0            | 0                        | 0            | 0
 template1      |              |                          |              |
             |              |                          |              |
 fault_test     | 66           | 452099                   | 0            | 0
             | 0            | 0                        | 0            | 0
 db_access_test | 1            | 5982                     | 0            | 0
             | 0            | 0                        | 0            | 0
```

```
 postgres       | 42035        | 95179203                  | 5752         | 2678832898
            | 21157        | 441883182488              | 2            | 1520
 mydb           | 71           | 453514                    | 0            | 0
            | 1            | 190                       | 1            | 152
```

次の例は、OID 16401 を持つデータベースの各 DML オペレーションの平均累積レイテンシー (累積レイテンシーをカウントで割った値) を示しています。

```
=> SELECT select_count,
          select_latency_microsecs,
          select_latency_microsecs/NULLIF(select_count,0) select_latency_per_exec,
          insert_count,
          insert_latency_microsecs,
          insert_latency_microsecs/NULLIF(insert_count,0) insert_latency_per_exec,
          update_count,
          update_latency_microsecs,
          update_latency_microsecs/NULLIF(update_count,0) update_latency_per_exec,
          delete_count,
          delete_latency_microsecs,
          delete_latency_microsecs/NULLIF(delete_count,0) delete_latency_per_exec
     FROM aurora_stat_dml_activity(16401);
-[ RECORD 1 ]------------+-------------
select_count             | 451312
select_latency_microsecs | 80205857
select_latency_per_exec  | 177
insert_count             | 451001
insert_latency_microsecs | 123667646
insert_latency_per_exec  | 274
update_count             | 1353067
update_latency_microsecs | 200900695615
update_latency_per_exec  | 148478
delete_count             | 12
delete_latency_microsecs | 448
delete_latency_per_exec  | 37
```

aurora_stat_get_db_commit_latency

Aurora PostgreSQL データベースの累積コミットレイテンシーをマイクロ秒単位で取得します。コミットレイテンシーは、クライアントがコミットリクエストを送信してからコミット確認を受信するまでの時間として測定されます。

構文

```
aurora_stat_get_db_commit_latency(database_oid)
```

引数

database_oid

Aurora PostgreSQL データベースのオブジェクト ID (OID)。

戻り型

SETOF レコード

使用に関する注意事項

Amazon CloudWatch はこの関数を使用して、平均コミットレイテンシーを計算します。Amazon CloudWatch メトリクスの詳細、およびハイコミットレイテンシーのトラブルシューティング方法については、Amazon RDS コンソールでのメトリクスの表示 および Making better decisions about Amazon RDS with Amazon CloudWatch metrics を参照してください。

この統計をリセットするには、PostgreSQL 統計アクセス関数 pg_stat_reset を使用します。この統計が最後にリセットされた時刻をチェックするには、pg_stat_get_db_stat_reset_time 関数を使用します。PostgreSQL 統計アクセス関数の詳細については、PostgreSQL ドキュメントの統計コレクターを参照してください。

例

次の例では、pg_database クラスターのデータベースごとに累積コミットレイテンシーを取得します。

```
=> SELECT oid,
    datname,
    aurora_stat_get_db_commit_latency(oid)
    FROM pg_database;

  oid  |    datname     | aurora_stat_get_db_commit_latency
-------+----------------+-----------------------------------
 14006 | template0      |                                 0
 16384 | rdsadmin       |                         654387789
     1 | template1      |                                 0
 16401 | mydb           |                            229556
 69768 | postgres       |                             22011
```

次の例では、現在接続されているデータベースの累積コミットレイテンシーを取得します。この例では、aurora_stat_get_db_commit_latency 関数を呼び出す前に、\gset を使用して oid 引数の可変を定義し、接続しているデータベースからその値を設定します。

```
--Get the oid value from the connected database before calling
 aurora_stat_get_db_commit_latency
=> SELECT oid
     FROM pg_database
    WHERE datname=(SELECT current_database()) \gset
=> SELECT *
     FROM aurora_stat_get_db_commit_latency(:oid);

 aurora_stat_get_db_commit_latency
-----------------------------------
                        1424279160
```

次の例では、pg_database クラスターの mydb データベース用の累積コミットレイテンシーを取得します。次に、この統計をリセットするには pg_stat_reset 関数を呼び出し、結果を表示します。最後に pg_stat_get_db_stat_reset_time 関数を使用して、この統計が最後にリセットされた時刻をチェックします。

```
=> SELECT oid,
    datname,
    aurora_stat_get_db_commit_latency(oid)
    FROM pg_database
    WHERE datname = 'mydb';

  oid  |  datname  | aurora_stat_get_db_commit_latency
-------+-----------+-----------------------------------
 16427 | mydb      |                           3320370


=> SELECT pg_stat_reset();
 pg_stat_reset
---------------


=> SELECT oid,
          datname,
          aurora_stat_get_db_commit_latency(oid)
     FROM pg_database
    WHERE datname = 'mydb';
```

```
  oid  |  datname  | aurora_stat_get_db_commit_latency
-------+-----------+-----------------------------------
 16427 | mydb      |                                 6


=> SELECT *
     FROM pg_stat_get_db_stat_reset_time(16427);

 pg_stat_get_db_stat_reset_time
--------------------------------
 2021-04-29 21:36:15.707399+00
```

aurora_stat_logical_wal_cache

スロットごとのログ先行書き込み (WAL) のキャッシュ使用状況を表示します。

構文

```
SELECT * FROM aurora_stat_logical_wal_cache()
```

引数

なし

戻り型

次の列を含む SETOF レコード。

- name – レプリケーションスロットの名前。
- active_pid— Walsender プロセスの ID。
- cache_hit— 前回のリセット以降の WAL キャッシュヒットの総数。
- cache_miss— 前回のリセット以降の WAL キャッシュミスの総数。
- blks_read— WAL キャッシュの読み取りリクエストの総数。
- hit_rate— WAL キャッシュヒットレート (cache_hit / blks_read)。
- last_reset_timestamp— カウンターが最後にリセットされた時刻。

使用に関する注意事項

この関数は、次のバージョンで使用できます。

- Aurora PostgreSQL 14.7
- Aurora PostgreSQL バージョン 13.8 以降
- Aurora PostgreSQL バージョン 12.12 以降
- Aurora PostgreSQL バージョン 11.17 以降

例

次の例は、一方のみがアクティブな 2 つのレプリケーションスロットを示しています。aurora_stat_logical_wal_cache 関数。

```
=> SELECT *
     FROM aurora_stat_logical_wal_cache();
    name    | active_pid | cache_hit | cache_miss | blks_read | hit_rate |
 last_reset_timestamp
------------+------------+-----------+------------+-----------+----------
+-------------------------------
 test_slot1 |      79183 |        24 |          0 |        24 | 100.00%  | 2022-08-05
 17:39:56.830635+00
 test_slot2 |            |         1 |          0 |         1 | 100.00%  | 2022-08-05
 17:34:04.036795+00
(2 rows)
```

aurora_stat_memctx_usage

各 PostgreSQL プロセスのメモリコンテキスト使用状況を報告します。

構文

```
aurora_stat_memctx_usage()
```

引数

なし

戻り型

次の列を含む SETOF レコード。

- pid - プロセスの ID。
- name – メモリコンテキストの名前。

- allocated — 基になるメモリサブシステムからメモリコンテキストによって取得されたバイト数。
- used — メモリコンテキストのクライアントにコミットされたバイト数。
- instances — このタイプの現在存在するコンテキストの数。

使用に関する注意事項

この関数は、各 PostgreSQL プロセスのメモリコンテキスト使用状況を表示します。一部のプロセスには anonymous というラベルが付いています。プロセスには制限付きのキーワードが含まれているため、公開されません。

この関数は、次の Aurora PostgreSQL バージョン以降で使用できます。

- 15.3 以降の 15 バージョン
- 14.8 以降の 14 バージョン
- 13.11 以降の 13 バージョン
- 12.15 以降の 12 バージョン
- 11.20 以降の 11 バージョン

例

次の例は、aurora_stat_memctx_usage 関数呼び出しの結果を示しています。

```
=> SELECT *
     FROM aurora_stat_memctx_usage();

    pid| name                             | allocated |   used  | instances
-------+----------------------------------+-----------+---------+-----------
123864 | Miscellaneous                    |     19520 |   15064 |         3
123864 | Aurora File Context              |      8192 |     616 |         1
123864 | Aurora WAL Context               |      8192 |     296 |         1
123864 | CacheMemoryContext               |    524288 |  422600 |         1
123864 | Catalog tuple context            |     16384 |   13736 |         1
123864 | ExecutorState                    |     32832 |   28304 |         1
123864 | ExprContext                      |      8192 |    1720 |         1
123864 | GWAL record construction         |      1024 |     832 |         1
123864 | MdSmgr                           |      8192 |     296 |         1
123864 | MessageContext                   |    532480 |  353832 |         1
123864 | PortalHeapMemory                 |      1024 |     488 |         1
```

```
123864 | PortalMemory                    |      8192 |     576 |         1
123864 | printtup                        |      8192 |     296 |         1
123864 | RelCache hash table entries     |      8192 |    8152 |         1
123864 | RowDescriptionContext           |      8192 |    1344 |         1
123864 | smgr relation context           |      8192 |     296 |         1
123864 | Table function arguments        |      8192 |     352 |         1
123864 | TopTransactionContext           |      8192 |     632 |         1
123864 | TransactionAbortContext         |     32768 |     296 |         1
123864 | WAL record construction         |     50216 |   43904 |         1
123864 | hash table                      |     65536 |   52744 |         6
123864 | Relation metadata               |    191488 |  124240 |        87
104992 | Miscellaneous                   |      9280 |    7728 |         3
104992 | Aurora File Context             |      8192 |     376 |         1
104992 | Aurora WAL Context              |      8192 |     296 |         1
104992 ||Autovacuum Launcher             |      8192 |     296 |         1
104992 | Autovacuum database list        |     16384 |     744 |         2
104992 | CacheMemoryContext              |    262144 |  140288 |         1
104992 | Catalog tuple context           |      8192 |     296 |         1
104992 | GWAL record construction        |      1024 |     832 |         1
104992 | MdSmgr                          |      8192 |     296 |         1
104992 | PortalMemory                    |      8192 |     296 |         1
104992 | RelCache hash table entries     |      8192 |     296 |         1
104992 | smgr relation context           |      8192 |     296 |         1
104992 | Autovacuum start worker (tmp)   |      8192 |     296 |         1
104992 | TopTransactionContext           |     16384 |     592 |         2
104992 | TransactionAbortContext         |     32768 |     296 |         1
104992 | WAL record construction         |     50216 |   43904 |         1
104992 | hash table                      |     49152 |   34024 |         4
(39 rows)
```

一部の制限付きキーワードは非表示になり、出力は次のようになります。

```
postgres=>SELECT *
     FROM aurora_stat_memctx_usage();

    pid| name                            | allocated |   used  | instances
-------+---------------------------------+-----------+---------+-----------
  5482 | anonymous                       |      8192 |     456 |         1
  5482 | anonymous                       |      8192 |     296 |         1
```

aurora_stat_optimized_reads_cache

この関数は階層型キャッシュの統計情報を表示します。

構文

```
aurora_stat_optimized_reads_cache()
```

引数

なし

戻り型

次の列を含む SETOF レコード。

- `total_size` – Optimized Reads キャッシュの合計サイズ。
- `used_size` – Optimized Reads キャッシュの使用済みページサイズ。

使用に関する注意事項

この関数は、次の Aurora PostgreSQL バージョンで使用できます。

- 15.4 以降の 15 バージョン
- 14.9 以降の 14 バージョン

例

次の例は、r6gd.8xlarge インスタンスでの出力を示しています。

```
=> select pg_size_pretty(total_size) as total_size, pg_size_pretty(used_size)
                as used_size from aurora_stat_optimized_reads_cache();
total_size | used_size
-----------+-----------
1054 GB    | 975 GB
```

aurora_stat_plans

追跡されたすべての実行計画の行を返します。

構文

```
aurora_stat_plans(
    showtext
```

```
)
```

引数

- showtext – クエリと計画のテキストを表示します。有効な値は、NULL、true または false です。True はクエリと計画のテキストを表示します。

戻り型

aurora_stat_statements からのすべての列とそれに続く追加の列を含む追跡計画ごとに行を返します。

- planid - 計画識別子
- explain_plan – 計画の説明テキスト
- plan_type:
  - no plan - 計画はキャプチャされませんでした
  - estimate - 推定コストでキャプチャされた計画
  - actual - EXPLAIN ANALYZE でキャプチャされた計画
- plan_captured_time – 計画が最後にキャプチャされた時刻

使用に関する注意事項

追跡する計画については、aurora_compute_plan_id が有効で、pg_stat_statements が shared_preload_libraries 内にある必要があります。

使用可能な計画の数は、pg_stat_statements.max パラメータで設定された値によって制御されます。compute_plan_id を有効にすると、aurora_stat_plans でこの指定された値までの計画を追跡できます。

この関数は、Aurora PostgreSQL バージョン 14.10、15.5、およびそれ以降のすべてのバージョンで使用できます。

例

以下の例では、クエリ識別子 -5471422286312252535 の 2 つの計画がキャプチャされ、ステートメントの統計が planid によって追跡されます。

```
db1=# select calls, total_exec_time, planid, plan_captured_time, explain_plan
```

```
db1-# from aurora_stat_plans(true)
db1-# where queryid = '-5471422286312252535'

calls    |  total_exec_time   |   planid    |      plan_captured_time       |
                explain_plan
---------+--------------------+-------------+-------------------------------
+-------------------------------------------------------------------
 1532632 |  3209846.097107853 |  1602979607 | 2023-10-31 03:27:16.925497+00 | Update on
 pgbench_branches                                        +
         |                    |             |                               |   ->
 Bitmap Heap Scan on pgbench_branches                       +
         |                    |             |                               |
 Recheck Cond: (bid = 76)                                   +
         |                    |             |                               |         -
>  Bitmap Index Scan on pgbench_branches_pkey          +
         |                    |             |                               |
     Index Cond: (bid = 76)
   61365 | 124078.18012200127 | -2054628807 | 2023-10-31 03:20:09.85429+00  | Update on
 pgbench_branches                                        +
         |                    |             |                               |   ->
 Index Scan using pgbench_branches_pkey on pgbench_branches+
         |                    |             |                               |
 Index Cond: (bid = 17)
```

aurora_stat_reset_wal_cache

論理 WAL キャッシュのカウンターをリセットします。

構文

特定のスロットをリセットするには

```
SELECT * FROM aurora_stat_reset_wal_cache('slot_name')
```

すべてのスロットをリセットするには

```
SELECT * FROM aurora_stat_reset_wal_cache(NULL)
```

引数

NULL、または slot_name

戻り型

ステータスメッセージ、テキスト文字列

- 論理 WAL キャッシュカウンターのリセット — 成功メッセージ このテキストは、関数が成功すると返されます。
- レプリケーションスロットが見つかりません。もう一度試してください。– 障害メッセージ このテキストは、関数が成功しなかった場合に返されます。

使用に関する注意事項

この関数は、次のバージョンで使用できます。

- Aurora PostgreSQL 14.5 以降
- Aurora PostgreSQL バージョン 13.8 以降
- Aurora PostgreSQL バージョン 12.12 以降
- Aurora PostgreSQL バージョン 11.17 以降

例

次の例では、aurora_stat_reset_wal_cache 関数を使用して test_results という名前のスロットをリセットし、存在しないスロットをリセットします。

```
=> SELECT *
     FROM aurora_stat_reset_wal_cache('test_slot');
aurora_stat_reset_wal_cache
-------------------------------------
 Reset the logical wal cache counter.
(1 row)
=> SELECT *
     FROM aurora_stat_reset_wal_cache('slot-not-exist');
aurora_stat_reset_wal_cache
-----------------------------------------------
 Replication slot not found. Please try again.
(1 row)
```

aurora_stat_statements

すべての pg_stat_statements 列を表示し、最後にさらに列を追加します。

構文

```
aurora_stat_statements(showtext boolean)
```

引数

showtext boolean

戻り型

すべての pg_stat_statements 列と以下の追加列を含む SETOF レコード。pg_stat_statements 列の詳細については、「pg_stat_statements」を参照してください。

この関数の統計情報は pg_stat_statements_reset( ) を使用してリセットできます。

- storage_blks_read - このステートメントによって Aurora ストレージから読み取られた共有ブロックの総数。
- orcache_blks_hit - このステートメントによる Optimized Reads キャッシュヒットの総数。
- storage_blk_read_time - track_io_timing を有効にすると、ステートメントが Aurora ストレージからのデータファイルブロックの読み取りにかかった合計時間をミリ秒単位で追跡します。それ以外の場合、値はゼロです。詳細については、track_io_timing を参照してください。
- local_blk_read_time - track_io_timing を有効にすると、ステートメントがローカルデータファイルブロックの読み取りにかかった合計時間をミリ秒単位で追跡します。それ以外の場合、値は 0 です。詳細については、track_io_timing を参照してください。
- orcache_blk_read_time - track_io_timing を有効にすると、ステートメントが Optimized Reads キャッシュからデータファイルブロックを読み取るのにかかった合計時間をミリ秒単位で追跡します。それ以外の場合は 0 です。詳細については、track_io_timing を参照してください。

使用に関する注意事項

aurora_stat_statements() 関数を使用するには、shared_preload_libraries パラメータに pg_stat_statements 拡張子を含める必要があります。

この関数は、次の Aurora PostgreSQL バージョンで使用できます。

- 15.4 以降の 15 バージョン

- 14.9 以降の 14 バージョン

例

次の例は、すべての pg_stat_statements 列を保持し、末尾に新しい列を 5 つ追加する方法を示しています。

```
=> select * from aurora_stat_statements(true) where queryid=-7342090857217643794;
-[ RECORD 1 ]---------+--------------------------------------------------
userid                | 10
dbid                  | 16419
toplevel              | t
queryid               | -7342090857217643794
query                 | CREATE TABLE quad_point_tbl AS                   +
                      |     SELECT point(unique1,unique2) AS p FROM tenk1
plans                 | 0
total_plan_time       | 0
min_plan_time         | 0
max_plan_time         | 0
mean_plan_time        | 0
stddev_plan_time      | 0
calls                 | 1
total_exec_time       | 571.844376
min_exec_time         | 571.844376
max_exec_time         | 571.844376
mean_exec_time        | 571.844376
stddev_exec_time      | 0
rows                  | 10000
shared_blks_hit       | 462
shared_blks_read      | 422
shared_blks_dirtied   | 0
shared_blks_written   | 55
local_blks_hit        | 0
local_blks_read       | 0
local_blks_dirtied    | 0
local_blks_written    | 0
temp_blks_read        | 0
temp_blks_written     | 0
blk_read_time         | 170.634621
blk_write_time        | 0
wal_records           | 0
wal_fpi               | 0
wal_bytes             | 0
```

```
storage_blks_read     | 47
orcache_blks_hit      | 375
storage_blk_read_time | 124.505772
local_blk_read_time   | 0
orcache_blk_read_time | 44.684038
```

aurora_stat_system_waits

Aurora PostgreSQL DB インスタンス用に待機イベント情報をレポートします。

構文

```
aurora_stat_system_waits()
```

引数

なし

戻り型

SETOF レコード

使用に関する注意事項

この関数は、現在接続している DB インスタンスによって生成された各待機イベントの累積待機数と累積待機時間を返します。

返されるレコードセットには、次のフィールドが含まれます。

- `type_id` - 待機イベントタイプの ID。
- `event_id` - 待機イベントの ID。
- `waits` - 待機イベントが発生した回数。
- `wait_time` - このイベントの待機に費やされた合計時間 (マイクロ秒)。

この関数によって返される統計は、DB インスタンスの再起動時にリセットされます。

例

次の例は、`aurora_stat_system_waits` 関数呼び出しの結果を示しています。

```
=> SELECT *
    FROM aurora_stat_system_waits();
 type_id | event_id  |   waits   |  wait_time
---------+-----------+-----------+--------------
       1 |  16777219 |        11 |        12864
       1 |  16777220 |       501 |       174473
       1 |  16777270 |     53171 |     23641847
       1 |  16777271 |        23 |       319668
       1 |  16777274 |        60 |        12759
.
.
.
      10 | 167772231 |    204596 |    790945212
      10 | 167772232 |         2 |        47729
      10 | 167772234 |         1 |          888
      10 | 167772235 |         2 |           64
```

次の例は、aurora_stat_wait_event および aurora_stat_wait_type とともにこの関数を使用して、より読みやすい結果を生成する方法を示しています。

```
=> SELECT type_name,
             event_name,
             waits,
             wait_time
        FROM aurora_stat_system_waits()
NATURAL JOIN aurora_stat_wait_event()
NATURAL JOIN aurora_stat_wait_type();

 type_name |       event_name       |  waits   |  wait_time
-----------+------------------------+----------+--------------
 LWLock    | XidGenLock             |       11 |        12864
 LWLock    | ProcArrayLock          |      501 |       174473
 LWLock    | buffer_content         |    53171 |     23641847
 LWLock    | rdsutils               |        2 |        12764
 Lock      | tuple                  |    75686 |   2033956052
 Lock      | transactionid          |  1765147 |  47267583409
 Activity  | AutoVacuumMain         |   136868 |  56305604538
 Activity  | BgWriterHibernate      |     7486 |  55266949471
 Activity  | BgWriterMain           |     7487 |   1508909964
.
.
.
 IO        | SLRURead               |        3 |        11756
```

```
IO        | WALWrite                | 52544463 |    388850428
IO        | XactSync                |   187073 |    597041642
IO        | ClogRead                |        2 |        47729
IO        | OutboundCtrlRead        |        1 |          888
IO        | OutboundCtrlWrite       |        2 |           64
```

aurora_stat_wait_event

Aurora PostgreSQL でサポートされているすべての待機イベントを一覧表示します。Aurora PostgreSQL のバージョンの詳細については、「Amazon Aurora PostgreSQL のイベント」を参照してください。

構文

```
aurora_stat_wait_event()
```

引数

なし

戻り型

次の列を含む SETOF レコード。

- type_id - 待機イベントタイプの ID。
- event_id - 待機イベントの ID。
- type_name — 待機タイプ名
- event_name — 待機イベント名

使用に関する注意事項

ID の代わりに、イベントタイプを持つイベント名を表示するには、aurora_stat_wait_type や aurora_stat_system_waits などの他の関数とともにこの関数を使用します。この関数によって返される待機イベント名は、aurora_wait_report 関数によって返されるものと同じです。

例

次の例は、aurora_stat_wait_event 関数呼び出しの結果を示しています。

```
=>  SELECT *
    FROM aurora_stat_wait_event();

 type_id | event_id  |               event_name
---------+-----------+-----------------------------------------
       1 |  16777216 | <unassigned:0>
       1 |  16777217 | ShmemIndexLock
       1 |  16777218 | OidGenLock
       1 |  16777219 | XidGenLock
.
.
.
       9 | 150994945 | PgSleep
       9 | 150994946 | RecoveryApplyDelay
      10 | 167772160 | BufFileRead
      10 | 167772161 | BufFileWrite
      10 | 167772162 | ControlFileRead
.
.
.
      10 | 167772226 | WALInitWrite
      10 | 167772227 | WALRead
      10 | 167772228 | WALSync
      10 | 167772229 | WALSyncMethodAssign
      10 | 167772230 | WALWrite
      10 | 167772231 | XactSync
.
.
.
      11 | 184549377 | LsnAllocate
```

次の例では、aurora_stat_wait_type および aurora_stat_wait_event を結合してタイプ名とイベント名を返し、読みやすさを向上させます。

```
=> SELECT *
    FROM aurora_stat_wait_type() t
    JOIN aurora_stat_wait_event() e
      ON t.type_id = e.type_id;

 type_id | type_name | type_id | event_id  |               event_name
---------+-----------+---------+-----------+-----------------------------------------
       1 | LWLock    |       1 |  16777216 | <unassigned:0>
       1 | LWLock    |       1 |  16777217 | ShmemIndexLock
```

```
       1 | LWLock     |       1 |  16777218 | OidGenLock
       1 | LWLock     |       1 |  16777219 | XidGenLock
       1 | LWLock     |       1 |  16777220 | ProcArrayLock
.
.
.
       3 | Lock       |       3 |  50331648 | relation
       3 | Lock       |       3 |  50331649 | extend
       3 | Lock       |       3 |  50331650 | page
       3 | Lock       |       3 |  50331651 | tuple
.
.
.
      10 | IO         |      10 | 167772214 | TimelineHistorySync
      10 | IO         |      10 | 167772215 | TimelineHistoryWrite
      10 | IO         |      10 | 167772216 | TwophaseFileRead
      10 | IO         |      10 | 167772217 | TwophaseFileSync
.
.
.
      11 | LSN        |      11 | 184549376 | LsnDurable
```

aurora_stat_wait_type

Aurora PostgreSQL でサポートされているすべての待機タイプを一覧表示します。

構文

```
aurora_stat_wait_type()
```

引数

なし

戻り型

次の列を含む SETOF レコード。

- type_id - 待機イベントタイプの ID。
- type_name — 待機タイプ名。

使用に関する注意事項

ID の代わりにイベントタイプを持つイベント名を表示するには、aurora_stat_wait_event や aurora_stat_system_waits などの他の関数とともにこの関数を使用します。この関数によって返される待機タイプ名は、aurora_wait_report 関数によって返されるものと同じです。

例

次の例は、aurora_stat_wait_type 関数呼び出しの結果を示しています。

```
=> SELECT *
     FROM aurora_stat_wait_type();
 type_id | type_name
---------+-----------
       1 | LWLock
       3 | Lock
       4 | BufferPin
       5 | Activity
       6 | Client
       7 | Extension
       8 | IPC
       9 | Timeout
      10 | IO
      11 | LSN
```

aurora_version

Amazon Aurora PostgreSQL 互換エディションバージョン番号の文字列値を返します。

構文

```
aurora_version()
```

引数

なし

戻り型

CHAR 型または VARCHAR 型の文字列

使用に関する注意事項

この関数は、Amazon Aurora PostgreSQL 互換エディションデータベースエンジンのバージョンを表示します。バージョン番号は、*####.####.###* という形式の文字列として返されます。Aurora PostgreSQL のバージョン番号の詳細については、「Aurora バージョン番号」を参照してください。

Aurora PostgreSQL DB クラスターのメンテナンスウィンドウを設定することで、マイナーバージョンアップグレードを適用するタイミングを選択できます。この方法の詳細は、「Amazon Aurora DB クラスターのメンテナンス」を参照してください。

PostgreSQL バージョン 13.3、12.8、11.13、10.18、およびそれ以降のすべてのリリースからは、Aurora バージョン番号は PostgreSQL バージョン番号に従います。すべての Aurora PostgreSQL リリースの詳細については、『Aurora PostgreSQL のリリースノート』の「Amazon Aurora PostgreSQL の更新」を参照してください。

例

次の例は、PostgreSQL 12.7、Aurora PostgreSQL リリース 4.2 を実行している Aurora PostgreSQL DB クラスターに対して `aurora_version` 関数を呼び出し、その後、Aurora PostgreSQL バージョン 13.3 を実行しているクラスターに対して同じ関数を実行した結果を示しています。

```
=> SELECT * FROM aurora_version();
aurora_version
----------------
 4.2.2
SELECT * FROM aurora_version();
aurora_version
----------------
 13.3.0
```

この例は、Aurora PostgreSQL のバージョンに関する詳細を取得するために、さまざまなオプションを指定して関数を使用する方法を示しています。この例では、PostgreSQL のバージョン番号とは異なる Aurora バージョン番号があります。

```
=> SHOW SERVER_VERSION;
 server_version
----------------
 12.7
(1 row)
```

```
=> SELECT * FROM aurora_version();
aurora_version
---------------
 4.2.2
(1 row)

=> SELECT current_setting('server_version') AS "PostgreSQL Compatiblility";
PostgreSQL Compatiblility
--------------------------
 12.7
(1 row)

=> SELECT version() AS "PostgreSQL Compatiblility Full String";
PostgreSQL Compatiblility Full String
------------------------------------------------------------------------------------------------
 PostgreSQL 12.7 on aarch64-unknown-linux-gnu, compiled by aarch64-unknown-linux-gnu-
gcc (GCC) 7.4.0, 64-bit
(1 row)

=> SELECT 'Aurora: '
    || aurora_version()
    || ' Compatible with PostgreSQL: '
    || current_setting('server_version') AS "Instance Version";
Instance Version
-----------------------------------------------
 Aurora: 4.2.2 Compatible with PostgreSQL: 12.7
(1 row)
```

次の例では、前の例と同じオプションで関数を使用します。この例では、PostgreSQL のバージョン番号とは異なる Aurora バージョン番号はありません。

```
=> SHOW SERVER_VERSION;
server_version
---------------
 13.3

=> SELECT * FROM aurora_version();
aurora_version
---------------
 13.3.0
=> SELECT current_setting('server_version') AS "PostgreSQL Compatiblility";
PostgreSQL Compatiblility
```

```
---------------------------
 13.3

=> SELECT version() AS "PostgreSQL Compatiblility Full String";
PostgreSQL Compatiblility Full String
------------------------------------------------------------------------------------------------------------
 PostgreSQL 13.3 on x86_64-pc-linux-gnu, compiled by x86_64-pc-linux-gnu-gcc (GCC)
 7.4.0, 64-bit
=> SELECT 'Aurora: '
    || aurora_version()
    || ' Compatible with PostgreSQL: '
    || current_setting('server_version') AS "Instance Version";
Instance Version
-----------------------------------------------------
 Aurora: 13.3.0 Compatible with PostgreSQL: 13.3
```

aurora_volume_logical_start_lsn

Aurora クラスターボリュームの論理先書きログ (WAL) ストリーム内のレコードの先頭を識別するために使用されるログシーケンス番号 (LSN) を返します。

構文

```
aurora_volume_logical_start_lsn()
```

引数

なし

戻り型

pg_lsn

使用に関する注意事項

この関数は、特定の Aurora クラスターボリュームの論理 WAL ストリーム内のレコードの先頭を識別します。この関数は、論理レプリケーションと Aurora 高速クローニングを使用してメジャーバージョンアップグレードを実行し、スナップショットまたはデータベースクローンを作成する LSN を決定するときに使用できます。その後、論理レプリケーションを使用して、LSN の後に記録された新しいデータを継続的にストリーミングし、パブリッシャーからサブスクライバーへの変更を同期できます。

論理レプリケーションをメジャーバージョンアップグレードに使用する方法の詳細については、「論理レプリケーションを使用して Aurora PostgreSQL のメジャーバージョンアップグレードを実行する」を参照してください。

この関数は、次の Aurora PostgreSQL バージョンで使用できます。

- バージョン 15 の 15.2 以降
- バージョン 14 の中の 14.3 以降
- バージョン 13 の 13.6 以降
- バージョン 12 の 12.10 以降
- 11.15 以上の 11 バージョン
- 10.20 以上の 10 バージョン

例

次のクエリを使用してログシーケンス番号 (LSN) を取得できます。

```
postgres=> SELECT aurora_volume_logical_start_lsn();

aurora_volume_logical_start_lsn
---------------
0/402E2F0
(1 row)
```

aurora_wait_report

この関数は、期間中の待機イベントアクティビティを表示します。

構文

```
aurora_wait_report([time])
```

引数

時間 (オプション)

秒単位の時間。デフォルトは 10 秒です。

戻り型

次の列を含む SETOF レコード。

- type_name — 待機タイプ名
- event_name — 待機イベント名
- wait — 待機数
- wait_time — ミリ秒単位の待機時間
- ms_per_wait — 待機数による平均ミリ秒
- waits_per_xact — 1 トランザクションの数による平均待機数
- ms_per_wait — 待機数による平均ミリ秒

使用に関する注意事項

この関数は、PostgreSQL 9.6.6 以上のバージョンと互換性がある Aurora PostgreSQL リリース 1.1 で使用できます。

この関数を使用するには、まず、次のように Aurora PostgreSQL `aurora_stat_utils` 拡張機能を作成する必要があります。

```
=> CREATE extension aurora_stat_utils;
CREATE EXTENSION
```

使用可能な Aurora PostgreSQL 拡張機能バージョンの詳細については、『Aurora PostgreSQL のリリースノート』の「Amazon Aurora PostgreSQL の拡張機能バージョン」を参照してください。

この関数は、aurora_stat_system_waits() 関数と pg_stat_database PostgreSQL 統計ビューからの統計データの 2 つのスナップショットを比較して、インスタンスレベルの待機イベントを計算します。

`aurora_stat_system_waits()` と `pg_stat_database` の詳細については、『PostgreSQL ドキュメント』の「統計コレクター」を参照してください。

この関数を実行すると、最初のスナップショットを取得し、指定された秒数だけ待ってから、2 番目のスナップショットを取得します。この関数は 2 つのスナップショットを比較し、差を返します。この差は、その時間間隔におけるインスタンスのアクティビティを表します。

ライターインスタンスでは、この関数はコミットされたトランザクションの数と TPS (1 秒あたりのトランザクション) も表示します。この関数は、インスタンスレベルで情報を返し、インスタンス上のすべてのデータベースを含みます。

例

この例は、aurora_stat_utils 拡張機能を作成して、aurora_log_report 関数を使用できるようにする方法を示しています。

```
=> CREATE extension aurora_stat_utils;
CREATE EXTENSION
```

この例は、10 秒間の待機レポートをチェックする方法を示しています。

```
=> SELECT *
     FROM aurora_wait_report();
NOTICE:  committed 34 transactions in 10 seconds (tps 3)
 type_name |    event_name     | waits | wait_time | ms_per_wait | waits_per_xact |
 ms_per_xact
-----------+-------------------+-------+-----------+-------------+----------------
+-------------
 Client    | ClientRead        |    26 |  30003.00 |    1153.961 |           0.76 |
 882.441
 Activity  | WalWriterMain     |    50 |  10051.32 |     201.026 |           1.47 |
 295.627
 Timeout   | PgSleep           |     1 |  10049.52 |   10049.516 |           0.03 |
 295.574
 Activity  | BgWriterHibernate |     1 |  10048.15 |   10048.153 |           0.03 |
 295.534
 Activity  | AutoVacuumMain    |    18 |   9941.66 |     552.314 |           0.53 |
 292.402
 Activity  | BgWriterMain      |     1 |    201.09 |     201.085 |           0.03 |
   5.914
 IO        | XactSync          |    15 |     25.34 |       1.690 |           0.44 |
   0.745
 IO        | RelationMapRead   |    12 |      0.54 |       0.045 |           0.35 |
   0.016
 IO        | WALWrite          |    84 |      0.21 |       0.002 |           2.47 |
   0.006
 IO        | DataFileExtend    |     1 |      0.02 |       0.018 |           0.03 |
   0.001
```

この例は、60 秒間の待機レポートをチェックする方法を示しています。

```
=> SELECT *
     FROM aurora_wait_report(60);
NOTICE:  committed 1544 transactions in 60 seconds (tps 25)
 type_name |       event_name       |  waits  | wait_time | ms_per_wait |
 waits_per_xact | ms_per_xact
-----------+------------------------+---------+-----------+-------------
+----------------+-------------
 Lock      | transactionid          |    6422 | 477000.53 |      74.276 |
 4.16 |     308.938
 Client    | ClientRead             |    8265 | 270752.99 |      32.759 |
 5.35 |     175.358
 Activity  | CheckpointerMain       |       1 |  60100.25 |   60100.246 |
 0.00 |      38.925
 Timeout   | PgSleep                |       1 |  60098.49 |   60098.493 |
 0.00 |      38.924
 Activity  | WalWriterMain          |     296 |  60010.99 |     202.740 |
 0.19 |      38.867
 Activity  | AutoVacuumMain         |     107 |  59827.84 |     559.139 |
 0.07 |      38.749
 Activity  | BgWriterMain           |     290 |  58821.83 |     202.834 |
 0.19 |      38.097
 IO        | XactSync               |    1295 |  55220.13 |      42.641 |
 0.84 |      35.764
 IO        | WALWrite               | 6602259 |  47810.94 |       0.007 |
 4276.07 |      30.966
 Lock      | tuple                  |     473 |  29880.67 |      63.173 |
 0.31 |      19.353
 LWLock    | buffer_mapping         |     142 |   3540.13 |      24.930 |
 0.09 |       2.293
 Activity  | BgWriterHibernate      |     290 |   1124.15 |       3.876 |
 0.19 |       0.728
 IO        | BufFileRead            |    7615 |    618.45 |       0.081 |
 4.93 |       0.401
 LWLock    | buffer_content         |      73 |    345.93 |       4.739 |
 0.05 |       0.224
 LWLock    | lock_manager           |      62 |    191.44 |       3.088 |
 0.04 |       0.124
 IO        | RelationMapRead        |      72 |      5.16 |       0.072 |
 0.05 |       0.003
 LWLock    | ProcArrayLock          |       1 |      2.01 |       2.008 |
 0.00 |       0.001
```

```
IO        | ControlFileWriteUpdate |       2 |      0.03 |       0.013 |
0.00 |       0.000
IO        | DataFileExtend         |       1 |      0.02 |       0.018 |
0.00 |       0.000
IO        | ControlFileSyncUpdate  |       1 |      0.00 |       0.000 |
0.00 |       0.000
```

Amazon Aurora PostgreSQL のパラメータ

Amazon Aurora DB クラスターの管理は、Amazon RDS DB インスタンスの場合と同じように、DB パラメータグループのパラメータを使用して行います。ただし、Amazon Aurora は、Aurora DB クラスターに複数の DB インスタンスがあるという点で Amazon RDS とは異なります。次のように、Amazon Aurora DB クラスターの管理に使用するパラメータの中には、クラスター全体に適用されるパラメータと、DB クラスター内の一部の DB インスタンスのみに適用されるパラメータがあります。

- DB クラスターのパラメータグループ – DB クラスターパラメータグループには、Aurora DB クラスター全体に適用される一連のエンジン設定パラメータが含まれます。例えば、クラスターキャッシュ管理は、DB クラスターパラメータグループに含まれる `apg_ccm_enabled` パラメータで制御される Aurora DB クラスターの機能です。DB クラスターパラメータグループには、クラスターを構成する DB インスタンスの DB パラメータグループのデフォルト設定も含まれています。
- DB パラメータグループ – DB パラメータグループは、そのエンジンタイプの特定の DB インスタンスに適用されるエンジン設定値のセットです。PostgreSQL DB エンジンの DB パラメータグループは、RDS for PostgreSQL DB インスタンスと Aurora PostgreSQL DB クラスターによって使用されます。これらの構成設定が適用されるプロパティ (メモリバッファのサイズなど) は、Aurora クラスター内の DB インスタンス間で異なる場合があります。

DB クラスター パラメータグループでクラスターレベルのパラメータを管理しています。インスタンスレベルのパラメータは、DB パラメータグループで管理されます。パラメータは、Amazon RDS コンソール、AWS CLI、または Amazon RDS API を使用して管理できます。クラスターレベルのパラメータとインスタンスレベルのパラメータでは、管理のためのコマンドが別です。

- DB クラスターパラメータグループのクラスターレベルのパラメータを管理するには、AWS CLI の modify-db-cluster-parameter-group コマンドを使用します。
- DB クラスター内の DB インスタンスに対する DB パラメータグループのインスタンスレベルのパラメータを管理するには、AWS CLI の modify-db-parameter-group コマンドを使用します。

AWS CLI の詳細については、AWS Command Line Interface ユーザーガイドの「AWS CLI の使用」を参照してください。

パラメータグループの詳細については、「「パラメータグループを使用する」」を参照してください。

Aurora PostgreSQL DB クラスターパラメータと DB パラメータの表示

RDS for PostgreSQL DB インスタンスおよび Aurora PostgreSQL DB クラスターで使用可能なすべてのデフォルトパラメータグループを AWS Management Console で表示できます。すべての DB エンジンおよび DB クラスターのタイプとバージョンのデフォルトのパラメータグループが、各 AWS リージョンについて一覧表示されます。カスタムパラメータグループも一覧表示されます。

AWS Management Console で表示するのではなく、DB クラスターパラメータグループと DB パラメータグループに含まれるパラメータを、AWS CLI または Amazon RDS API で一覧表示することもできます。例えば、DB クラスターパラメータグループのパラメータを表示するには、次のように、AWS CLI の describe-db-cluster-parameters コマンドを使用します。

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name
 default.aurora-postgresql12
```

このコマンドは、各パラメータの詳細な JSON 記述を返します。返される情報の量を減らすには、`--query` オプションを使用して必要な情報を指定します。例えば、デフォルトの Aurora PostgreSQL 12 DB クラスターパラメータグループのパラメータ名、その説明、および取り得る値を次のように取得できます。

Linux、macOS、Unix の場合:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name
 default.aurora-postgresql12 \
    --query 'Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

Windows の場合:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name
 default.aurora-postgresql12 ^
    --query "Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

Aurora DB クラスターパラメータグループには、特定の Aurora DB エンジンの DB インスタンスパラメータグループとデフォルト値が含まれます。DB パラメータのリストは、同じデフォルトの Aurora PostgreSQL のデフォルトパラメータグループから、AWS CLI コマンドの describe-db-parameters を使用して次のように取得できます。

Linux、macOS、Unix の場合:

```
aws rds describe-db-parameters --db-parameter-group-name default.aurora-postgresql12 \
    --query 'Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

Windows の場合:

```
aws rds describe-db-parameters --db-parameter-group-name default.aurora-postgresql12 ^
    --query "Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

前述のコマンドは、DB クラスターまたは DB パラメータグループからのパラメータのリストと、クエリで指定された説明やその他の詳細を返します。以下に、応答の例を示します。

```
[
    [
        {
            "ParameterName": "apg_enable_batch_mode_function_execution",
            "ApplyType": "dynamic",
            "Description": "Enables batch-mode functions to process sets of rows at a
 time.",
            "AllowedValues": "0,1"
        }
    ],
    [
        {
            "ParameterName": "apg_enable_correlated_any_transform",
            "ApplyType": "dynamic",
            "Description": "Enables the planner to transform correlated ANY Sublink
 (IN/NOT IN subquery) to JOIN when possible.",
            "AllowedValues": "0,1"
        }
    ],...
```

Aurora PostgreSQL バージョン 14 のデフォルトの DB クラスターパラメータと DB パラメータの値を含む表を次に示します。

Aurora PostgreSQL クラスターレベルのパラメータ

AWS マネジメントコンソール、AWS CLI、または Amazon RDS API を使用して、特定の Aurora PostgresSQL バージョンのクラスターレベルパラメータを表示できます。RDS コンソールでの Aurora PostgreSQL DB クラスターパラメータグループ内のパラメータの表示方法については、「DB クラスターパラメータグループのパラメータ値を表示する」を参照してください。

クラスターレベルのパラメータの中には、すべてのバージョンで使用できないものもあり、一部は廃止されています。特定の Aurora PostgreSQL バージョンのパラメータを表示する方法については、「Aurora PostgreSQL DB クラスターパラメータと DB パラメータの表示」を参照してください。

例えば、次の表は Aurora PostgreSQL バージョン 14 のデフォルトの DB クラスターパラメータグループで使用可能なパラメータを示しています。独自のカスタム DB パラメータグループを指定せずに Aurora PostgreSQL DB クラスターを作成すると、DB クラスターは、選択したバージョンのデフォルトの Aurora DB クラスターパラメータグループ (`default.aurora-postgresql14`、`default.aurora-postgresql13` など) を使用して作成されます。

同じデフォルトの DB クラスターパラメータグループの DB インスタンスパラメータの一覧については、「Aurora PostgreSQL インスタンスレベルのパラメータ」を参照してください。

パラメータ名	説明	デフォルト値
ansi_constraint_trigger_ordering	制約トリガーの起動順序を ANSI SQL スタンダードと互換性があるように変更します。	–
ansi_force_foreign_key_checks	カスケード削除やカスケード更新などの参照アクションは、アクションに存在するさまざまなトリガーコンテキストに関係なく、常に実行されるようにします。	–
ansi_qualified_update_set_target	UPDATE ..。SET ステートメントでテーブルとスキーマ修飾子をサポートします。	–
apg_ccm_enabled	クラスターのクラスターキャッシュ管理を有効または無効にします。	–
apg_enable_batch_mode_function_execution	バッチモード関数が一度に一連の行を処理できるようにします。	–

パラメータ名	説明	デフォルト値
apg_enable_correlated_any_transform	プランナーが、可能な場合、相関関係のある ANY Sublink (IN/NOT IN サブクエリ) を JOIN に変換できるようにします。	–
apg_enable_function_migration	プランナーが対象となるスカラー関数を FROM 句に移行できるようにします。	–
apg_enable_not_in_transform	可能であれば、プランナーが NOT IN サブクエリを ANTI JOIN に変換できるようにします。	–
apg_enable_remove_redundant_inner_joins	プランナーが冗長な内部結合を削除できるようにします。	–
apg_enable_semijoin_push_down	ハッシュ結合のセミジョインフィルタの使用を有効にします。	–
apg_plan_mgmt.capture_plan_baselines	計画取得のベースラインモード。manual - すべての SQL ステートメントに対して計画取得を有効にします。off - 計画取得を無効にします。automatic - 対象となる基準を満たす pg_stat_statements 内のステートメントに対して計画取得を有効にします。	オフ
apg_plan_mgmt.max_databases	apg_plan_mgmt を使用してクエリを管理できるデータベースの最大数を設定します。	10
apg_plan_mgmt.max_plans	apg_plan_mgmt によってキャッシュされることができる計画の最大数を設定します。	10000
apg_plan_mgmt.plan_retention_period	計画が最後に使用された last_used の日付から、計画が自動的に削除されるまでの最大日数。	32
apg_plan_mgmt.unapproved_plan_execution_threshold	未承認計画は、見積総コストがこの額未満の場合実行されます。	0

パラメータ名	説明	デフォルト値
apg_plan_mgmt.use_plan_baselines	管理ステートメントに、承認済み計画または決定済み計画のみを使用します。	false
application_name	統計情報とログで報告されるアプリケーション名を設定します。	–
array_nulls	配列への NULL 要素の入力を有効にします。	–
aurora_compute_plan_id	クエリ実行計画をモニタリングして、現在のデータベース負荷の原因となる実行計画を検出し、実行計画のパフォーマンス統計を経時的に追跡できます。詳細については、「Aurora PostgreSQL のクエリ実行計画のモニタリング」を参照してください。	オン
authentication_timeout	(s) クライアント認証が完了するまでに許容される最大時間を設定します。	–
auto_explain.log_analyze	計画のログ記録に EXPLAIN ANALYZE を使用します。	–
auto_explain.log_buffers	バッファの使用状況をログに記録します。	–
auto_explain.log_format	計画のログ記録に使用する EXPLAIN 形式。	–
auto_explain.log_min_duration	最小実行時間を設定します。この値を超えると計画がログに記録されます。	–
auto_explain.log_nested_statement	ネストされたステートメントをログに記録します。	–
auto_explain.log_timing	行数だけでなく、タイミングデータを収集します。	–

パラメータ名	説明	デフォルト値
auto_explain.log_triggers	トリガー統計を計画に含めます。	–
auto_explain.log_verbose	計画のログ記録に EXPLAIN VERBOSE を使用します。	–
auto_explain.sample_rate	処理するクエリの割合。	–
autovacuum	autovacuum サブプロセスを起動します。	–
autovacuum_analyze_scale_factor	分析する前のタプルの挿入、更新、削除の数 (reltuples の割合として指定)。	0.05
autovacuum_analyze_threshold	分析する前のタプルの挿入、更新、削除の最小数。	–
autovacuum_freeze_max_age	トランザクション ID の循環を防ぐためにテーブルに対して autovacuum を実行する期間。	–
autovacuum_max_workers	同時に実行される autovacuum ワーカープロセスの最大数を設定します。	GREATEST(DBInstanceClassMemory/64371566592,3)
autovacuum_multixact_freeze_max_age	multixact の循環を防ぐためにテーブルに対して autovacuum を実行する multixact 期間。	–
autovacuum_naptime	(s) autovacuum の実行の間でスリープ状態になっている時間。	5
autovacuum_vacuum_cost_delay	(ミリ秒) autovacuum でのバキューム処理のコスト遅延。	5
autovacuum_vacuum_cost_limit	autovacuum でバキューム処理を停止する制限値となるバキューム処理のコスト	GREATEST(log(DBInstanceClassMemory/21474836480)*600,200)

パラメータ名	説明	デフォルト値
autovacuum_vacuum_insert_scale_factor	バキューム処理をする前のタプルの挿入の数 (reltuples の割合として指定)。	–
autovacuum_vacuum_insert_threshold	バキューム処理をする前のタプル挿入の最小数。-1 を指定するとバキュームの挿入が無効になります。	–
autovacuum_vacuum_scale_factor	バキューム処理をする前のタプルの更新または削除の数 (reltuples の割合として指定)。	0.1
autovacuum_vacuum_threshold	バキューム処理をする前のタプルの更新また削除の最小数。	–
autovacuum_work_mem	(kB) 各 autovacuum ワーカープロセスで使用するメモリの最大量を設定します。	GREATEST(DBInstanceClassMemory/32768,131072)
babelfishpg_tds.default_server_name	デフォルトの Babelfish サーバーの名前。	Microsoft SQL Server
babelfishpg_tds.listen_addresses	TDS をリッスンするためのホスト名または IP アドレスを設定します。	*
babelfishpg_tds.port	サーバーがリッスンする TDS TCP ポートを設定します。	1433
babelfishpg_tds.tds_debug_log_level	TDS でのログ記録レベルを設定します。0 にするとログ記録は無効になります。	1
babelfishpg_tds.tds_default_numeric_precision	エンジンが精度を指定していない場合に TDS 列のメタデータで送信される数値型のデフォルト精度を設定します。	38
babelfishpg_tds.tds_default_numeric_scale	エンジンがスケールを指定していない場合に TDS 列のメタデータで送信される数値型のデフォルトスケールを設定します。	8

パラメータ名	説明	デフォルト値
babelfishpg_tds.tds_default_packet_size	すべての SQL Server クライアントを接続するためのデフォルトのパケットサイズを設定します。	4096
babelfishpg_tds.tds_default_protocol_version	すべてのクライアントを接続するためのデフォルトの TDS プロトコルバージョンを設定します。	DEFAULT
babelfishpg_tds.tds_ssl_encrypt	SSL 暗号化オプションを設定します。	0
babelfishpg_tds.tds_ssl_max_protocol_version	tds セッションに使用する最大の SSL/TLS プロトコルバージョンを設定します。	TLSv1.2
babelfishpg_tds.tds_ssl_min_protocol_version	tds セッションに使用する最小の SSL/TLS プロトコルバージョンを設定します。	Aurora PostgreSQL バージョン 16 の TLSv1.2、Aurora PostgreSQL バージョン 16 より古いバージョンの TLSv1
babelfishpg_tsql.default_locale	CREATE COLLATION によって作成された照合に使用されるデフォルトのロケール。	en-US
babelfishpg_tsql.migration_mode	複数のユーザーデータベースがサポートされるかどうかを定義します。	Aurora PostgreSQL バージョン 16 の multi-db、Aurora PostgreSQL バージョン 16 より古いバージョンの single-db
babelfishpg_tsql.server_collation_name	デフォルトのサーバー照合順序の名前	sql_latin1_general_cp1_ci_as
babelfishpg_tsql.version	@@VERSION 変数の出力を設定します。	デフォルト

パラメータ名	説明	デフォルト値
backend_flush_after	(8Kb) 以前に実行された書き込みがディスクにフラッシュされるまでのページ数。	–
backslash_quote	文字列リテラルで \\ を許可するかどうかを指定します。	–
backtrace_functions	これらの関数のエラーのバックトレースをログに記録します。	–
bytea_output	バイトの出力形式を設定します。	–
check_function_bodies	CREATE FUNCTION の実行中に関数の本体をチェックします。	–
client_connection_check_interval	クエリ実行中の切断を確認する時間間隔を設定します。	–
client_encoding	クライアントの文字セットエンコードを設定します。	UTF8
client_min_messages	クライアントへ送信されるメッセージレベルを設定します。	–
compute_query_id	クエリ ID を計算します。	auto
config_file	サーバーのメイン設定ファイルを設定します。	/rdsdbdata/config/postgresql.conf
constraint_exclusion	クエリを最適化するために、プランナーが制約を使用できるようにします。	–
cpu_index_tuple_cost	インデックススキャンの実行中に各インデックスエントリを処理する際にかかるコストに対するプランナーの見積もりを設定します。	–
cpu_operator_cost	演算子や関数呼び出しのそれぞれを処理する際にかかるコストに対するプランナーの見積もりを設定します。	–

パラメータ名	説明	デフォルト値
cpu_tuple_cost	各タプル (行) の処理にかかるコストに対するプランナーの見積もりを設定します。	–
cron.database_name	pg_cron メタデータテーブルを保存するようにデータベースを設定します	postgres
cron.log_run	実行されたすべてのジョブを job_run_details テーブルにログとして記録します。	オン
cron.log_statement	実行前のすべての cron ステートメントをログ記録します。	オフ
cron.max_running_jobs	同時に実行できるジョブの最大数。	5
cron.use_background_workers	pg_cron のバックグラウンドワーカーを有効にします。	オン
cursor_tuple_fraction	取得されるカーソル行の割合に対するプランナーの見積もりを設定します。	–
data_directory	サーバーのデータディレクトリを設定します。	/rdsdbdata/db
datestyle	日付と時刻の値の表示形式を設定します。	–
db_user_namespace	データベースごとのユーザー名を有効にします。	–
deadlock_timeout	(ms) デッドロックをチェックするまでロックを待機する時間を設定します。	–
debug_pretty_print	分析ツリーや計画ツリーの表示をインデントして見やすくします。	–
debug_print_parse	各クエリの分析ツリーをログに記録します。	–
debug_print_plan	各クエリの実行計画をログに記録します。	–

パラメータ名	説明	デフォルト値
debug_print_rewritten	各クエリの書き直された分析ツリーをログに記録します。	–
default_statistics_target	デフォルトの統計情報の対象を設定します。	–
default_tablespace	テーブルとインデックスを作成するためのデフォルトのテーブルスペースを設定します。	–
default_toast_compression	圧縮可能な値のデフォルトの圧縮方法を設定します。	–
default_transaction_deferrable	新しいトランザクションのデフォルトの遅延ステータスを設定します。	–
default_transaction_isolation	新しい各トランザクションのトランザクション分離レベルを設定します。	–
default_transaction_read_only	新しいトランザクションのデフォルトの読み取り専用ステータスを設定します。	–
effective_cache_size	(8kB) ディスクキャッシュのサイズに関するプランナーの予測を設定します。	SUM(DBInstanceClassMemory/12038,-50003)
effective_io_concurrency	ディスクサブシステムで効率的に処理できる同時リクエストの数。	–
enable_async_append	プランナーが非同期追加計画を使用できるようにします。	–
enable_bitmapscan	プランナーがビットマップスキャン計画を使用できるようにします。	–
enable_gathermerge	プランナーがマージ収集計画を使用できるようにします。	–

パラメータ名	説明	デフォルト値
enable_hashagg	プランナーがハッシュされた集計計画を使用できるようにします。	–
enable_hashjoin	プランナーがハッシュ結合計画を使用できるようにします。	–
enable_incremental_sort	プランナーが増分ソートステップを使用できるようにします。	–
enable_indexonlyscan	プランナーがインデックスのみのスキャン計画を使用できるようにします。	–
enable_indexscan	プランナーがインデックススキャン計画を使用できるようにします。	–
enable_material	プランナーがマテリアル化を使用できるようにします。	–
enable_memoize	プランナーがメモを使用できるようにします	–
enable_mergejoin	プランナーがマージ結合計画を使用できるようにします。	–
enable_nestloop	プランナーがネステッドループ結合計画を使用できるようにします。	–
enable_parallel_append	プランナーが並列追加計画を使用できるようにします。	–
enable_parallel_hash	プランナーが並列ハッシュ計画を使用できるようにします。	–
enable_partition_pruning	計画時および実行時のパーティションプルーニングを有効にします。	–
enable_partitionwise_aggregate	パーティション単位の集計とグループ化を有効にします。	–

パラメータ名	説明	デフォルト値
enable_partitionwise_join	パーティション単位の結合を有効にします。	–
enable_seqscan	プランナーがシーケンシャルスキャン計画を使用できるようにします。	–
enable_sort	プランナーが明示的なソートステップを使用できるようにします。	–
enable_tidscan	プランナーが TID スキャン計画を使用できるようにします。	–
escape_string_warning	通常の文字列リテラルにバックスラッシュのエスケープ文字が含まれている場合に警告を出します。	–
exit_on_error	エラーがあればセッションを終了します。	–
extra_float_digits	浮動小数点値の表示桁数を設定します。	–
force_parallel_mode	並列クエリ機能の使用を強制します。	–
from_collapse_limit	FROM リストのサイズを設定します。この値を超えるとサブクエリが折りたたまれなくなります。	–
geqo	遺伝的クエリ最適化を有効にします。	–
geqo_effort	GEQO: 他の GEQO パラメータのデフォルト値を設定するために使用されます。	–
geqo_generations	GEQO: アルゴリズムの反復の数。	–
geqo_pool_size	GEQO: 母集団内の個体の数。	–
geqo_seed	GEQO: 無作為のパスを選択するための初期値。	–

パラメータ名	説明	デフォルト値
geqo_selection_bias	GEQO: 母集団内の選択圧。	–
geqo_threshold	FROM 項目のしきい値を設定します。この値を超えると GEQO が使用されます。	–
gin_fuzzy_search_limit	GIN による完全一致検索で許可される結果の最大数を設定します。	–
gin_pending_list_limit	(kB) GIN インデックスの保留リストの最大サイズを設定します。	–
hash_mem_multiplier	ハッシュテーブルに使用する work_mem の乗数。	–
hba_file	サーバーの hba 設定ファイルを設定します。	/rdsdbdata/config/pg_hba.conf
hot_standby_feedback	ホットスタンバイからプライマリへのフィードバックを許可し、クエリの競合を回避します。	オン
huge_pages	DB インスタンスが、共有バッファで使用されるような、大きく連続したメモリチャンクで動作しているときのオーバーヘッドを軽減します。t3.medium,db.t3.large,db.t4g.medium,db.t4g.large インスタンスクラス以外のすべての DB インスタンスクラスについて、デフォルトでオンになっています。	オン
ident_file	サーバー ID 設定ファイルを設定します。	/rdsdbdata/config/pg_ident.conf
idle_in_transaction_session_timeout	(ms) アイドリングトランザクションに許容される最大実行時間を設定します。	86400000

パラメータ名	説明	デフォルト値
idle_session_timeout	指定された時間を超えてアイドル状態 (クライアントからのクエリを待機している状態) であるが、オープントランザクション内ではないセッションを終了させます	–
intervalstyle	間隔値の表示形式を設定します。	–
join_collapse_limit	FROM リストのサイズを設定します。この値を超えると JOIN 構造が平坦化されなくなります。	–
krb_caseins_users	GSSAPI (汎用セキュリティサービス API) ユーザー名を大文字と小文字を区別しない (true) かどうかを設定します。デフォルトでは、このパラメータは false に設定されているため、Kerberos はユーザー名の大文字と小文字が区別されることを想定しています。詳細については、PostgreSQL のドキュメントの「GSSAPI Authentication」を参照してください。	false
lc_messages	メッセージを表示する言語を設定します。	–
lc_monetary	金額の書式のロケールを設定します。	–
lc_numeric	数値の書式のロケールを設定します。	–
lc_time	日付と時刻の書式のロケールを設定します。	–
listen_addresses	リッスンするホスト名または IP アドレスを設定します。	*
lo_compat_privileges	ラージオブジェクトの権限チェックの下位互換モードを有効にします。	0

パラメータ名	説明	デフォルト値
log_autovacuum_min_duration	(ms) autovacuum に関する最小実行時間を設定します。この値を超えると autovacuum アクションがログに記録されます。	10000
log_connections	成功した各接続をログに記録します。	–
log_destination	サーバーログの出力先を設定します。	stderr
log_directory	ログファイルの保存先ディレクトリを設定します。	/rdsdbdata/log/error
log_disconnections	セッションの終了をログに記録します (セッションの有効期間も含まれます)。	–
log_duration	完了した各 SQL ステートメントの期間をログに記録します。	–
log_error_verbosity	ログに記録されるメッセージの詳細を設定します。	–
log_executor_stats	実行プログラムのパフォーマンスの統計情報をサーバーログに書き込みます。	–
log_file_mode	ログファイルのファイルアクセス許可を設定します。	0644
log_filename	ログファイルのファイル名のパターンを設定します。	postgresql.log.%Y-%m-%d-%H%M
logging_collector	サブプロセスを開始して、stderr 出力や csvlogs をログファイルにキャプチャします。	1
log_hostname	接続ログにホスト名を記録します。	0
logical_decoding_work_mem	(kB) この量のメモリは、ディスクに書き込むことなく、各内部リオーダバッファで使用できます。	–

パラメータ名	説明	デフォルト値
log_line_prefix	各ログ行の先頭に付ける情報を制御します。	%t:%r:%u@%d:%p]:
log_lock_waits	長期間にわたるロックの待機をログに記録します。	–
log_min_duration_sample	(ms) ステートメントのサンプリングに関する最小実行時間を設定します。この値を超えるとステートメントがサンプリングされてログに記録されます。サンプリングは log_statement_sample_rate によって決定されます。	–
log_min_duration_statement	(ms) ステートメントに関する最小実行時間を設定します。この値を超えるとステートメントがログに記録されます。	–
log_min_error_statement	設定したレベル以上のエラーが発生したすべてのステートメントをログに記録します。	–
log_min_messages	ログに記録するメッセージレベルを設定します。	–
log_parameter_max_length	(B) ステートメントをログに記録するときに、ログに記録されるパラメータ値を最初の N バイトに制限します。	–
log_parameter_max_length_on_error	(B) エラーを報告するときに、ログに記録されるパラメータ値を最初の N バイトに制限します。	–
log_parser_stats	分析のパフォーマンスの統計情報をサーバーログに書き込みます。	–
log_planner_stats	プランナーのパフォーマンスの統計情報をサーバーログに書き込みます。	–
log_replication_commands	各レプリケーションコマンドをログに記録します。	–

パラメータ名	説明	デフォルト値
log_rotation_age	(min) N 分が経過するとログファイルのローテーションが自動的に発生します。	60
log_rotation_size	(kB) N キロバイトを超えるとログファイルのローテーションが自動的に発生します。	100000
log_statement	ログに記録するステートメントのタイプを設定します。	–
log_statement_sample_rate	ログに記録される log_min_duration_sample を超えるステートメントの割合。	–
log_statement_stats	累積処理のパフォーマンスの統計情報をサーバーログに書き込みます。	–
log_temp_files	(kB) 指定したサイズ (キロバイト) を超える一時ファイルの使用をログに記録します。	–
log_timezone	ログメッセージで使用するタイムゾーンを設定します。	UTC
log_transaction_sample_rate	新しいトランザクションに対してログに記録するトランザクションの割合を設定します。	–
log_truncate_on_rotation	ログローテーション中に同じ名前の既存のログファイルを切り捨てます。	0
maintenance_io_concurrency	メンテナンス作業に使用される effective_io_concurrency のバリアント。	1
maintenance_work_mem	(kB) メンテナンスオペレーションに使用するメモリの最大量を設定します。	GREATEST(DBInstanceClassMemory/63963136*1024,65536)

パラメータ名	説明	デフォルト値
max_connections	同時接続の最大数を設定します。	LEAST(DBInstanceClassMemory/9531392,5000)
max_files_per_process	各サーバープロセスで同時に開くことができるファイルの最大数を設定します。	–
max_locks_per_transaction	トランザクションあたりのロックの最大数を設定します。	64
max_logical_replication_workers	論理レプリケーションワーカープロセスの最大数。	–
max_parallel_maintenance_workers	メンテナンスオペレーションあたりの並列プロセスの最大数を設定します。	–
max_parallel_workers	一度にアクティブにできる並列ワーカーの最大数を設定します。	GREATEST($DBInstanceVCPU/2,8)
max_parallel_workers_per_gather	エグゼキューターノードあたりの並列プロセスの最大数を設定します。	–
max_pred_locks_per_page	ページあたりの述語ロックタプルの最大数を設定します。	–
max_pred_locks_per_relation	リレーションあたりの述語ロックページとタプルの最大数を設定します。	–
max_pred_locks_per_transaction	トランザクションあたりの述語ロックの最大数を設定します。	–
max_prepared_transactions	同時に準備できるトランザクションの最大数を設定します。	0
max_replication_slots	サーバーがサポートできるレプリケーションスロットの最大数を設定します。	20

パラメータ名	説明	デフォルト値
max_slot_wal_keep_size	(MB) ディスク上の WAL がこの量のスペースを占有している場合、レプリケーションスロットは障害があるとマークされ、セグメントは削除またはリサイクルのために解放されます。	–
max_stack_depth	(kB) スタックの深度の最大値をキロバイト単位で設定します。	6144
max_standby_streaming_delay	(ms) ホットスタンバイサーバーがストリーミングされた WAL データを処理しているときにクエリをキャンセルするまでの最大遅延を設定します。	14000
max_sync_workers_per_subscription	サブスクリプションあたりの同期ワーカーの最大数	2
max_wal_senders	同時に実行される WAL 送信者プロセスの最大数を設定します。	10
max_worker_processes	同時ワーカープロセスの最大数を設定します。	GREATEST($DBInstanceVCPU*2, 8)
min_dynamic_shared_memory	(MB) 起動時に予約された動的共有メモリ量。	–
min_parallel_index_scan_size	(8kB) パラレルスキャンのインデックスデータの最小量を設定します。	–
min_parallel_table_scan_size	(8kB) パラレルスキャンのテーブルデータの最小量を設定します。	–
old_snapshot_threshold	(分) スナップショットが古すぎるため、スナップショット取得後に変更されたページ読み取ることができないと判定されるまでの時間。	–

パラメータ名	説明	デフォルト値
orafce.nls_date_format	Oracle Date の出力動作をエミュレートします。	–
orafce.timezone	sysdate 関数に使用するタイムゾーンを指定します。	–
parallel_leader_participation	Gather と Gather Merge もサブプランを実行するかどうかをコントロールします。	–
parallel_setup_cost	並列クエリのワーカープロセスを起動するコストのプランナーの見積もりを設定します。	–
parallel_tuple_cost	ワーカーからマスターバックエンドへの各タプル (行) を渡すのにかかるコストのプランナーの見積もりを設定します。	–
password_encryption	パスワードを暗号化します。	–
pgaudit.log	セッション監査ログ記録によってログに記録されるステートメントのクラスを指定します。	–
pgaudit.log_catalog	ステートメント内のすべてのリレーションが pg_catalog 内にある場合に、セッションログ記録を有効にするように指定します。	–
pgaudit.log_level	ログエントリに使用されるログレベルを指定します。	–
pgaudit.log_parameter	ステートメントとともに渡されたパラメータを監査ログ記録に含めるように指定します。	–
pgaudit.log_relation	セッション監査ログ記録で、SELECT ステートメントまたは DML ステートメントで参照されるリレーション (TABLE、VIEW など) ごとに個別のログエントリを作成するかどうかを指定します。	–

パラメータ名	説明	デフォルト値
pgaudit.log_statement_once	ログ記録に、ステートメントテキストとパラメータを、ステートメントとサブステートメントの組み合わせの最初のログエントリとともに含めるか、すべてのエントリとともに含めるかを指定します。	–
pgaudit.role	オブジェクト監査ログ記録に使用するマスターロールを指定します。	–
pg_bigm.enable_recheck	これは、フルテキスト検索の内部プロセスである Recheck を実行するかどうかを指定します。	オン
pg_bigm.gin_key_limit	これは、フルテキスト検索に使用される検索キーワードの 2-gram の最大数を指定します。	0
pg_bigm.last_update	pg_bigm モジュールの最終更新日が報告されます。	2013.11.22
pg_bigm.similarity_limit	類似性検索で使用される最小しきい値を指定します。	0.3
pg_hint_plan.debug_print	ヒント解析の結果をログに記録します。	–
pg_hint_plan.enable_hint	プランナーが、クエリの前のヒントコメントで指定された計画を使用するようにします。	–
pg_hint_plan.enable_hint_table	プランナーが、テーブルルックアップを使用してヒントを取得しないようにします。	–
pg_hint_plan.message_level	デバッグメッセージのメッセージレベル。	–
pg_hint_plan.parse_messages	解析エラーのメッセージレベル。	–

パラメータ名	説明	デフォルト値
pglogical.batch_inserts	可能であれば、バッチ挿入	–
pglogical.conflict_log_level	解決された競合の口グ記録に使用する口グレベルを設定します。	–
pglogical.conflict_resolution	解決可能な競合の競合解決に使用されるメソッドを設定します。	–
pglogical.extra_connection_options	すべてのピアノード接続に追加する接続オプション	–
pglogical.synchronous_commit	pglogical 固有の同期コミット値	–
pglogical.use_spi	低レベル API の代わりに SPI を使用して変更を適用します	–
pgtle.clientauth_databases_to_skip	clientauth 機能でスキップするデータベースのリスト。	–
pgtle.clientauth_db_name	clientauth 機能でどのデータベースが使用されるかを制御します。	–
pgtle.clientauth_num_parallel_workers	clientauth 機能に使用されるバックグラウンドワーカーの数。	–
pgtle.clientauth_users_to_skip	clientauth 機能でスキップするユーザーのリスト。	–
pgtle.enable_clientauth	clientauth 機能を有効にします。	–
pgtle.passcheck_db_name	クラスター全体の passcheck 機能に使用されるデータベースを設定します。	–
pg_prewarm.autoprewarm	自動プレウォームワーカーを開始します。	–

パラメータ名	説明	デフォルト値
pg_prewarm.autoprewarm_interval	共有バッファのダンプ間隔を設定します。	–
pg_similarity.block_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.block_threshold	ブロック類似度関数で使用されるしきい値を設定します。	–
pg_similarity.block_tokenizer	ブロック類似性関数のトークナイザを設定します。	–
pg_similarity.cosine_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.cosine_threshold	コサイン類似度関数で使用されるしきい値を設定します。	–
pg_similarity.cosine_tokenizer	コサイン類似度関数のトークナイザを設定します。	–
pg_similarity.dice_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.dice_threshold	Dice 類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.dice_tokenizer	Dice 類似度メジャーのトークナイザを設定します。	–
pg_similarity.euclidean_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.euclidean_threshold	ユークリッド類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.euclidean_tokenizer	ユークリッド類似度メジャーのトークナイザを設定します。	–

パラメータ名	説明	デフォルト値
pg_similarity.hamming_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.hamming_threshold	ブロック類似度メトリクスで使用されるしきい値を設定します。	–
pg_similarity.jaccard_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.jaccard_threshold	Jaccard 類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.jaccard_tokenizer	Jaccard 類似度メジャーのトークナイザを設定します。	–
pg_similarity.jaro_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.jaro_threshold	Jaro 類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.jarowinkler_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.jarowinkler_threshold	Jarowinkler 類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.levenshtein_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.levenshtein_threshold	Levenshtein 類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.matching_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.matching_threshold	マッチング係数の類似度メジャーで使用されるしきい値を設定します。	–

パラメータ名	説明	デフォルト値
pg_similarity.matching_tokenizer	マッチング係数の類似度メジャーのトークナイザを設定します。	–
pg_similarity.mongeelkan_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.mongeelkan_threshold	Monge-Elkan 類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.mongeelkan_tokenizer	Monge-Elkan 類似度メジャーのトークナイザを設定します。	–
pg_similarity.nw_gap_penalty	Needleman-Wunsch 類似度メジャーで使用されるギャップペナルティを設定します。	–
pg_similarity.nw_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.nw_threshold	Needleman-Wunsch 類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.overlap_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.overlap_threshold	オーバーラップ係数の類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.overlap_tokenizer	オーバーラップ係数の類似度メジャーのトークナイザを設定します。	–
pg_similarity.qgram_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.qgram_threshold	q-gram 類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.qgram_tokenizer	q-gram メジャーのトークナイザを設定します。	–

パラメータ名	説明	デフォルト値
pg_similarity.swg_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.swg_threshold	Smith-Waterman-Gotoh 類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.sw_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.sw_threshold	Smith-Waterman 類似度メジャーで使用されるしきい値を設定します。	–
pg_stat_statements.max	pg_stat_statements によって追跡されるステートメントの最大数を設定します。	–
pg_stat_statements.save	pg_stat_statements 統計情報をサーバーのシャットダウン全体にわたって保存します。	–
pg_stat_statements.track	pg_stat_statements によって追跡されるステートメントを選択します。	–
pg_stat_statements.track_planning	pg_stat_statements で計画期間を追跡するかどうかを選択します。	–
pg_stat_statements.track_utility	pg_stat_statements によってユーティリティコマンドを追跡するかどうかを選択します。	–
plan_cache_mode	カスタムプランまたは汎用プランのプランナーによる選択を制御します。	–
port	サーバーがリッスンする TCP ポートを設定します。	EndPointPort
postgis.gdal_enabled_drivers	Postgres 9.3.5 以降で PostGIS で使用される GDAL ドライバーを有効または無効にします。	ENABLE_ALL
quote_all_identifiers	SQL フラグメントを生成するときに、すべての識別子を引用符で囲みます。	–

パラメータ名	説明	デフォルト値
random_page_cost	非連続的に取得されたディスクページのコストに対するプランナーの見積もりを設定します。	–
rdkit.dice_threshold	Dice 類似度のしきい値が低い。類似度がしきい値より低い分子は、# オペレーションでは類似しません。	–
rdkit.do_chiral_sss	部分構造のマッチングで立体化学を考慮するかどうか。false の場合、部分構造のマッチングで立体化学情報は使用されません。	–
rdkit.tanimoto_threshold	Tanimoto 類似度の下限しきい値。類似度がしきい値より低い分子は、% オペレーションでは類似しません。	–
rds.accepted_password_auth_method	ローカルに保存されているパスワードを使用して接続を強制的に認証します。	md5+scram
rds.adaptive_autovacuum	アダプティブ autovacuum を有効/無効にする RDS パラメータ。	1
rds.babelfish_status	Aurora PostgreSQL の Babelfish を有効/無効にする RDS パラメータ。	オフ
rds.enable_plan_management	apg_plan_mgmt 拡張機能を有効または無効にします。	0

パラメータ名	説明	デフォルト値
rds.extensions	RDS によって提供される拡張機能のリスト	address_standardizer、address_standardizer_data_us、apg_plan_mgmt、aurora_stat_utils、amcheck、autoinc、aws_commons、aws_ml、aws_s3、aws_lambda、bool_plperl、bloom、btree_gin、btree_gist、citext、cube、dblink、dict_int、dict_xsyn、earthdistance、fuzzystrmatch、hll、hstore、hstore_plperl、insert_username、intagg、intarray、ip4r、isn、jsonb_plperl、lo、log_fdw、ltree、moddatetime、old_snapshot、oracle_fdw、orafce、pgaudit、pgcrypto、pglogical、pgrouting、pgrowlocks、pgstattuple、pgtap、pg_bigm、pg_buffercache、pg_cron、pg_freespacemap、pg_hint_plan、pg_partm

パラメータ名	説明	デフォルト値
		an、pg_prewarm、pg_proctab、pg_repack、pg_similarity、pg_stat_statements、pg_trgm、pg_visibility、plcoffee、plls、plperl、plpgsql、plprofiler、pltcl、plv8、postgis、postgis_tiger_geocoder、postgis_raster、postgis_topology、postgres_fdw、prefix、rdkit、rds_tools、refint、sslinfo、tablefunc、tds_fdw、test_parser、tsm_system_rows、tsm_system_time、unaccent、uuid-ossp
rds.force_admin_logging_level	カスタマーデータベースの RDS 管理者ユーザーアクションの口グメッセージを参照してください。	–
rds.force_autovacuum_logging_level	autovacuum オペレーションに関連する口グメッセージを参照してください。	WARNING
rds.force_ssl	強制的に SSL 接続するようにします。	0

パラメータ名	説明	デフォルト値
rds.global_db_rpo	(秒) 違反時にユーザーコミットをブロックするリカバリポイントの客観的なしきい値。 **Important** このパラメータは Aurora PostgreSQL ベースのグローバルデータベース用です。グローバルデータベース以外の場合は、デフォルト値のままにします。このパラメータの使用に関する詳細については、「the section called "Aurora PostgreSQL- ベースのグローバルデータベースの RPO (目標復旧時点) 管理"」を参照してください。	–
rds.logical_replication	論理デコードを有効にします。	0
rds.logically_replicate_unlogged_tables	ログに記録されていないテーブルは論理的に複製されます。	1
rds.log_retention_period	N 分より古い PostgreSQL ログは Amazon RDS で削除されます。	4320
rds.pg_stat_ramdisk_size	統計ラムディスクのサイズ (MB 単位)。ゼロ以外の値を指定すると、ラムディスクがセットアップされます。このパラメータは、Aurora PostgreSQL 14 以下のすべてのバージョンで利用できます。	0
rds.rds_superuser_reserved_connections	rds_superusers 用に予約されている接続スロットの数を設定します。このパラメータは、バージョン 15 以前でのみ使用できます。詳細については、PostgreSQL ドキュメントの「reserved connections」を参照してください。	2

パラメータ名	説明	デフォルト値
rds.restrict_password_commands	パスワード関連のコマンドを rds_password のメンバーに制限します	–
rds.superuser_variables	rds_superuser 変更ステートメントを昇格させるスーパーユーザー専用変数のリスト。	session_replication_role
recovery_init_sync_method	クラッシュリカバリの前にデータディレクトリを同期する方法を設定します。	syncfs
remove_temp_files_after_crash	バックエンドクラッシュ後に一時ファイルを削除します。	0
restart_after_crash	バックエンドクラッシュ後にサーバーを再初期化します。	–
row_security	行セキュリティを有効にします。	–
search_path	スキーマによって修飾されていない名前でスキーマを検索する順序を設定します。	–
seq_page_cost	連続的に取得されたディスクページのコストに対するプランナーの見積もりを設定します。	–
session_replication_role	トリガーと再書き込みルールに対するセッション動作を設定します。	–
shared_buffers	(8kB) サーバーで使用される共有メモリバッファの数を設定します。	SUM(DBInstanceClassMemory/12038,-50003)
shared_preload_libraries	サーバーにプリロードする共有ライブラリを一覧表示します。	pg_stat_statements
ssl	SSL 接続を有効にします。	1
ssl_ca_file	SSL サーバー権限ファイルの場所。	/rdsdbdata/rds-metadata/ca-cert.pem

パラメータ名	説明	デフォルト値
ssl_cert_file	SSL サーバー証明書ファイルの場所。	/rdsdbdata/rds-metadata/server-cert.pem
ssl_ciphers	安全な接続で使用できる TLS 暗号のリストを設定します。	–
ssl_crl_dir	SSL 証明書の失効リストディレクトリの場所。	/rdsdbdata/rds-metadata/ssl_crl_dir/
ssl_key_file	SSL サーバーのプライベートキーファイルの場所	/rdsdbdata/rds-metadata/server-key.pem
ssl_max_protocol_version	許容される 最大の SSL/TLS プロトコルのバージョンを設定します	–
ssl_min_protocol_version	許容される 最小の SSL/TLS プロトコルのバージョンを設定します	TLSv1.2
standard_conforming_strings	... 文字列をリテラルのバックスラッシュとして扱います。	–
statement_timeout	(ms) すべてのステートメントに許可される最大実行時間を設定します。	–
stats_temp_directory	統計情報の一時ファイルを指定したディレクトリに書き込みます。	/rdsdbdata/db/pg_stat_tmp
superuser_reserved_connections	スーパーユーザー用に予約されている接続スロットの数を設定します。	3
synchronize_seqscans	シーケンシャルスキャンの同期を有効にします。	–
synchronous_commit	現在のトランザクションの同期レベルを設定します。	オン
tcp_keepalives_count	TCP キープアライブを再送信する最大回数。	–

パラメータ名	説明	デフォルト値
tcp_keepalives_idle	(s) TCP キープアライブを発行する時間間隔。	–
tcp_keepalives_interval	(s) TCP キープアライブを再送信する時間間隔。	–
temp_buffers	(8kB) 各セッションで使用される一時バッファの最大数を設定します。	–
temp_file_limit	特定の PostgreSQL プロセスが一時ファイルに使用できるディスク容量の合計をキロバイト単位で制限します。明示的な一時テーブルに使用される領域は除きます。	-1
temp_tablespaces	一時テーブルとソートファイルで使用するテーブルスペースを設定します。	–
timezone	表示やタイムスタンプの解釈で必要となるタイムゾーンを設定します。	UTC
track_activities	実行中のコマンドに関する情報を収集します。	–
track_activity_query_size	pg_stat_activity.current_query 用に予約するサイズをバイト単位で設定します。	4096
track_commit_timestamp	トランザクションのコミット時間を収集します。	–
track_counts	データベースアクティビティの統計情報を収集します。	–
track_functions	データベースアクティビティの関数レベルの統計情報を収集します。	pl
track_io_timing	データベース IO アクティビティのタイミングに関する統計情報を収集します。	1
track_wal_io_timing	WAL I/O アクティビティのタイミングに関する統計情報を収集します。	–

パラメータ名	説明	デフォルト値
transform_null_equals	expr=NULL を expr IS NULL として扱います。	–
update_process_title	アクティブな SQL コマンドを表示するようにプロセスのタイトルを更新します。	–
vacuum_cost_delay	(ms) バキューム処理のコスト遅延 (ミリ秒単位)。	–
vacuum_cost_limit	バキューム処理を停止する制限値となるバキューム処理のコスト。	–
vacuum_cost_page_dirty	バキューム処理によってダーティになったページに対するバキューム処理のコスト。	–
vacuum_cost_page_hit	バッファキャッシュ内で検出されたページに対するバキューム処理のコスト。	–
vacuum_cost_page_miss	バッファキャッシュ内で検出されなかったページに対するバキューム処理のコスト。	0
vacuum_defer_cleanup_age	VACUUM と HOT クリーンアップが延期されるトランザクションの数 (存在する場合)。	–
vacuum_failsafe_age	VACUUM が循環による停止を回避するためにフェイルセーフをトリガーするべき期間。	1200000000
vacuum_freeze_min_age	VACUUM でテーブルの行をフリーズするまでの最小期間。	–
vacuum_freeze_table_age	VACUUM でテーブル全体をスキャンしタプルをフリーズするまでの期間。	–
vacuum_multixact_failsafe_age	VACUUM が循環による停止を回避するためにフェイルセーフをトリガーするべき Multixact 期間。	1200000000
vacuum_multixact_freeze_min_age	VACUUM でテーブルの行の MultiXactId をフリーズするまでの最小期間。	–

パラメータ名	説明	デフォルト値
vacuum_multixact_freeze_table_age	VACUUM でテーブル全体をスキャンしタプルをフリーズするまでの Multixact 期間。	–
wal_buffers	(8kB) WAL 用の共有メモリ内のディスクページバッファの数を設定します。	–
wal_receiver_create_temp_slot	永続スロットが設定されていない場合に、WAL レシーバーが一時的なレプリケーションスロットを作成するかどうかを設定します。	0
wal_receiver_status_interval	(s) WAL レシーバーステータスレポートをプライマリにするための最大時間間隔を設定します。	–
wal_receiver_timeout	(ms) プライマリからデータを受信するまでの最大待機時間を設定します。	30000
wal_sender_timeout	(ms) WAL レプリケーションを待機する最大時間を設定します。	–
work_mem	(kB) クエリワークスペースに使用するメモリの最大量を設定します。	–
xmlbinary	バイナリ値を XML にエンコードする方法を設定します。	–
xmloption	黙示的な分析とシリアル化オペレーションでの XML データをドキュメントとして見なすか、コンテンツのフラグメントとして見なすかを設定します。	–

Aurora PostgreSQL インスタンスレベルのパラメータ

AWS マネジメントコンソール、AWS CLI、または Amazon RDS API を使用して、特定の Aurora PostgresSQL バージョンのインスタンスレベルパラメータを表示できます。RDS コンソールでの Aurora PostgreSQL DB パラメータグループ内のパラメータの表示方法については、「DB パラメータグループのパラメータ値を表示する」を参照してください。

インスタンスレベルのパラメータの中には、すべてのバージョンで使用できないものもあり、一部は廃止されています。特定の Aurora PostgreSQL バージョンのパラメータを表示する方法については、「Aurora PostgreSQL DB クラスターパラメータと DB パラメータの表示」を参照してください。

例えば、次の表は Aurora PostgreSQL DB クラスターの特定の DB インスタンスに適用されるパラメータリストを示しています。このリストは、`--db-parameter-group-name` の値に `default.aurora-postgresql14` を指定して AWS CLI コマンドの describe-db-parameters を実行することで生成されます。

同じデフォルトの DB パラメータグループの DB クラスターパラメータの一覧については、「Aurora PostgreSQL クラスターレベルのパラメータ」を参照してください。

パラメータ名	説明	デフォルト値
apg_enable_batch_mode_function_execution	バッチモード関数が一度に一連の行を処理できるようにします。	–
apg_enable_correlated_any_transform	プランナーが、可能な場合、相関関係のある ANY Sublink (IN/NOT IN サブクエリ) を JOIN に変換できるようにします。	–
apg_enable_function_migration	プランナーが対象となるスカラー関数を FROM 句に移行できるようにします。	–
apg_enable_not_in_transform	可能であれば、プランナーが NOT IN サブクエリを ANTI JOIN に変換できるようにします。	–
apg_enable_remove_redundant_inner_joins	プランナーが冗長な内部結合を削除できるようにします。	–

パラメータ名	説明	デフォルト値
apg_enable_semijoin_push_down	ハッシュ結合のセミジョインフィルタの使用を有効にします。	–
apg_plan_mgmt.capture_plan_baselines	計画取得のベースラインモード。manual - すべての SQL ステートメントに対して計画取得を有効にします。off - 計画取得を無効にします。automatic - 対象となる基準を満たす pg_stat_statements 内のステートメントに対して計画取得を有効にします。	オフ
apg_plan_mgmt.max_databases	apg_plan_mgmt を使用してクエリを管理できるデータベースの最大数を設定します。	10
apg_plan_mgmt.max_plans	apg_plan_mgmt によってキャッシュされることができる計画の最大数を設定します。	10000
apg_plan_mgmt.plan_retention_period	計画が最後に使用された last_used の日付から、計画が自動的に削除されるまでの最大日数。	32
apg_plan_mgmt.unapproved_plan_execution_threshold	未承認計画は、見積総コストがこの額未満の場合実行されます。	0
apg_plan_mgmt.use_plan_baselines	管理ステートメントに、承認済み計画または決定済み計画のみを使用します。	false
application_name	統計情報とログで報告されるアプリケーション名を設定します。	–
aurora_compute_plan_id	クエリ実行計画をモニタリングして、現在のデータベース負荷の原因となる実行計画を検出し、実行計画のパフォーマンス統計を経時的に追跡できます。詳細については、「Aurora PostgreSQL のクエリ実行計画のモニタリング」を参照してください。	オン

パラメータ名	説明	デフォルト値
authentication_timeout	(秒) クライアント認証が完了するまでに許容される最大時間を設定します。	–
auto_explain.log_analyze	計画のログ記録に EXPLAIN ANALYZE を使用します。	–
auto_explain.log_buffers	バッファの使用状況をログに記録します。	–
auto_explain.log_format	計画のログ記録に使用する EXPLAIN 形式。	–
auto_explain.log_min_duration	最小実行時間を設定します。この値を超えると計画がログに記録されます。	–
auto_explain.log_nested_statement	ネストされたステートメントをログに記録します。	–
auto_explain.log_timing	行数だけでなく、タイミングデータを収集します。	–
auto_explain.log_triggers	トリガー統計を計画に含めます。	–
auto_explain.log_verbose	計画のログ記録に EXPLAIN VERBOSE を使用します。	–
auto_explain.sample_rate	処理するクエリの割合。	–
babelfishpg_tds.listen_addresses	TDS をリッスンするためのホスト名または IP アドレスを設定します。	*
babelfishpg_tds.tds_debug_log_level	TDS でのログ記録レベルを設定します。0 にするとログ記録は無効になります。	1
backend_flush_after	(8kB) 以前に実行された書き込みがディスクにフラッシュされるまでのページ数。	–

パラメータ名	説明	デフォルト値
bytea_output	バイトの出力形式を設定します。	–
check_function_bodies	CREATE FUNCTION の実行中に関数の本体をチェックします。	–
client_connection_check_interval	クエリ実行中の切断を確認する時間間隔を設定します。	–
client_min_messages	クライアントへ送信されるメッセージレベルを設定します。	–
config_file	サーバーのメイン設定ファイルを設定します。	/rdsdbdata/config/postgresql.conf
constraint_exclusion	クエリを最適化するために、プランナーが制約を使用できるようにします。	–
cpu_index_tuple_cost	インデックススキャンの実行中に各インデックスエントリを処理する際にかかるコストに対するプランナーの見積もりを設定します。	–
cpu_operator_cost	演算子や関数呼び出しのそれぞれを処理する際にかかるコストに対するプランナーの見積もりを設定します。	–
cpu_tuple_cost	各タプル (行) の処理にかかるプランナーのコスト見積もりを設定します。	–
cron.database_name	pg_cron メタデータテーブルを保存するようにデータベースを設定します	postgres
cron.log_run	実行されたすべてのジョブを job_run_details テーブルにログとして記録します。	オン
cron.log_statement	実行前のすべての cron ステートメントをログ記録します。	オフ

パラメータ名	説明	デフォルト値
cron.max_running_jobs	同時に実行できるジョブの最大数。	5
cron.use_background_workers	pg_cron のバックグラウンドワーカーを有効にします。	オン
cursor_tuple_fraction	取得されるカーソル行の割合に対するプランナーの見積もりを設定します。	–
db_user_namespace	データベースごとのユーザー名を有効にします。	–
deadlock_timeout	(ミリ秒) デッドロックをチェックするまでロックを待機する時間を設定します。	–
debug_pretty_print	分析ツリーや計画ツリーの表示をインデントして見やすくします。	–
debug_print_parse	各クエリの分析ツリーをログに記録します。	–
debug_print_plan	各クエリの実行計画をログに記録します。	–
debug_print_rewritten	各クエリの書き直された分析ツリーをログに記録します。	–
default_statistics_target	デフォルトの統計情報の対象を設定します。	–
default_transaction_deferrable	新しいトランザクションのデフォルトの遅延ステータスを設定します。	–
default_transaction_isolation	新しい各トランザクションのトランザクション分離レベルを設定します。	–
default_transaction_read_only	新しいトランザクションのデフォルトの読み取り専用ステータスを設定します。	–

パラメータ名	説明	デフォルト値
effective_cache_size	(8kB) ディスクキャッシュのサイズに関するプランナーの予測を設定します。	SUM(DBInstanceClassMemory/12038,-50003)
effective_io_concurrency	ディスクサブシステムで効率的に処理できる同時リクエストの数。	–
enable_async_append	プランナーが非同期追加計画を使用できるようにします。	–
enable_bitmapscan	プランナーがビットマップスキャン計画を使用できるようにします。	–
enable_gathermerge	プランナーがマージ収集計画を使用できるようにします。	–
enable_hashagg	プランナーがハッシュされた集計計画を使用できるようにします。	–
enable_hashjoin	プランナーがハッシュ結合計画を使用できるようにします。	–
enable_incremental_sort	プランナーが増分ソートステップを使用できるようにします。	–
enable_indexonlyscan	プランナーがインデックスのみのスキャン計画を使用できるようにします。	–
enable_indexscan	プランナーがインデックススキャン計画を使用できるようにします。	–
enable_material	プランナーがマテリアル化を使用できるようにします。	–
enable_memoize	プランナーがメモを使用できるようにします	–
enable_mergejoin	プランナーがマージ結合計画を使用できるようにします。	–

パラメータ名	説明	デフォルト値
enable_nestloop	プランナーがネステッドループ結合計画を使用できるようにします。	–
enable_parallel_append	プランナーが並列追加計画を使用できるようにします。	–
enable_parallel_hash	プランナーが並列ハッシュ計画を使用できるようにします。	–
enable_partition_pruning	計画時および実行時のパーティションプルーニングを有効にします。	–
enable_partitionwise_aggregate	パーティション単位の集計とグループ化を有効にします。	–
enable_partitionwise_join	パーティション単位の結合を有効にします。	–
enable_seqscan	プランナーがシーケンシャルスキャン計画を使用できるようにします。	–
enable_sort	プランナーが明示的なソートステップを使用できるようにします。	–
enable_tidscan	プランナーが TID スキャン計画を使用できるようにします。	–
escape_string_warning	通常の文字列リテラルにバックスラッシュのエスケープ文字が含まれている場合に警告を出します。	–
exit_on_error	エラーがあればセッションを終了します。	–
force_parallel_mode	並列クエリ機能の使用を強制します。	–
from_collapse_limit	FROM リストのサイズを設定します。この値を超えるとサブクエリが折りたたまれなくなります。	–

パラメータ名	説明	デフォルト値
geqo	遺伝的クエリ最適化を有効にします。	–
geqo_effort	GEQO: 他の GEQO パラメータのデフォルト値を設定するために使用されます。	–
geqo_generations	GEQO: アルゴリズムの反復の数。	–
geqo_pool_size	GEQO: 母集団内の個体の数。	–
geqo_seed	GEQO: 無作為のパスを選択するための初期値。	–
geqo_selection_bias	GEQO: 母集団内の選択圧。	–
geqo_threshold	FROM 項目のしきい値を設定します。この値を超えると GEQO が使用されます。	–
gin_fuzzy_search_limit	GIN による完全一致検索で許可される結果の最大数を設定します。	–
gin_pending_list_limit	(kB) GIN インデックスの保留リストの最大サイズを設定します。	–
hash_mem_multiplier	ハッシュテーブルに使用する work_mem の乗数。	–
hba_file	サーバーの hba 設定ファイルを設定します。	/rdsdbdata/config/pg_hba.conf
hot_standby_feedback	ホットスタンバイからプライマリへのフィードバックを許可し、クエリの競合を回避します。	オン
ident_file	サーバー ID 設定ファイルを設定します。	/rdsdbdata/config/pg_ident.conf
idle_in_transaction_session_timeout	(ミリ秒) アイドリングトランザクションに許容される最大実行時間を設定します。	86400000

パラメータ名	説明	デフォルト値
idle_session_timeout	指定された時間を超えてアイドル状態 (クライアントからのクエリを待機している状態) であるが、オープントランザクション内ではないセッションを終了させます	–
join_collapse_limit	FROM リストのサイズを設定します。この値を超えると JOIN 構造が平坦化されなくなります。	–
lc_messages	メッセージを表示する言語を設定します。	–
listen_addresses	リッスンするホスト名または IP アドレスを設定します。	*
lo_compat_privileges	ラージオブジェクトの権限チェックの下位互換モードを有効にします。	0
log_connections	成功した各接続をログに記録します。	–
log_destination	サーバーログの出力先を設定します。	stderr
log_directory	ログファイルの保存先ディレクトリを設定します。	/rdsdbdata/log/error
log_disconnections	セッションの終了をログに記録します (セッションの有効期間も含まれます)。	–
log_duration	完了した各 SQL ステートメントの期間をログに記録します。	–
log_error_verbosity	ログに記録されるメッセージの詳細を設定します。	–
log_executor_stats	実行プログラムのパフォーマンスの統計情報をサーバーログに書き込みます。	–
log_file_mode	ログファイルのファイルアクセス許可を設定します。	0644

パラメータ名	説明	デフォルト値
log_filename	ログファイルのファイル名のパターンを設定します。	postgresql.log.%Y-%m-%d-%H%M
logging_collector	サブプロセスを開始して、stderr 出力や csvlogs をログファイルにキャプチャします。	1
log_hostname	接続ログにホスト名を記録します。	0
logical_decoding_work_mem	(kB) この量のメモリは、ディスクに書き込むことなく、各内部リオーダバッファで使用できます。	–
log_line_prefix	各ログ行の先頭に付ける情報を制御します。	%t:%r:%u@%d:%p]:
log_lock_waits	長期間にわたるロックの待機をログに記録します。	–
log_min_duration_sample	(ミリ秒) ステートメントのサンプリングに関する最小実行時間を設定します。この値を超えるとステートメントがサンプリングされてログに記録されます。サンプリングは log_statement_sample_rate によって決定されます。	–
log_min_duration_statement	(ミリ秒) ステートメントに関する最小実行時間を設定します。この値を超えるとステートメントがログに記録されます。	–
log_min_error_statement	設定したレベル以上のエラーが発生したすべてのステートメントをログに記録します。	–
log_min_messages	ログに記録するメッセージレベルを設定します。	–
log_parameter_max_length	(B) ステートメントをログに記録するときに、ログに記録されるパラメータ値を最初の N バイトに制限します。	–

パラメータ名	説明	デフォルト値
log_parameter_max_length_on_error	(B) エラーを報告するときに、ログに記録されるパラメータ値を最初の N バイトに制限します。	–
log_parser_stats	分析のパフォーマンスの統計情報をサーバーログに書き込みます。	–
log_planner_stats	プランナーのパフォーマンスの統計情報をサーバーログに書き込みます。	–
log_replication_commands	各レプリケーションコマンドをログに記録します。	–
log_rotation_age	(分) N 分が経過するとログファイルのローテーションが自動的に発生します。	60
log_rotation_size	(kB) N キロバイトを超えるとログファイルのローテーションが自動的に発生します。	100000
log_statement	ログに記録するステートメントのタイプを設定します。	–
log_statement_sample_rate	ログに記録される log_min_duration_sample を超えるステートメントの割合。	–
log_statement_stats	累積処理のパフォーマンスの統計情報をサーバーログに書き込みます。	–
log_temp_files	(kB) 指定したサイズ (キロバイト) を超える一時ファイルの使用をログに記録します。	–
log_timezone	ログメッセージで使用するタイムゾーンを設定します。	UTC
log_truncate_on_rotation	ログローテーション中に同じ名前の既存のログファイルを切り捨てます。	0

パラメータ名	説明	デフォルト値
maintenance_io_concurrency	メンテナンス作業に使用される effective_io_concurrency のバリアント。	1
maintenance_work_mem	(kB) メンテナンスオペレーションに使用するメモリの最大量を設定します。	GREATEST(DBInstanceClassMemory/63963136*1024, 65536)
max_connections	同時接続の最大数を設定します。	LEAST(DBInstanceClassMemory/9531392, 5000)
max_files_per_process	各サーバープロセスで同時に開くことができるファイルの最大数を設定します。	–
max_locks_per_transaction	トランザクションあたりのロックの最大数を設定します。	64
max_parallel_maintenance_workers	メンテナンスオペレーションあたりの並列プロセスの最大数を設定します。	–
max_parallel_workers	一度にアクティブにできる並列ワーカーの最大数を設定します。	GREATEST($DBInstanceVCPU/2, 8)
max_parallel_workers_per_gather	エグゼキューターノードあたりの並列プロセスの最大数を設定します。	–
max_pred_locks_per_page	ページあたりの述語ロックタプルの最大数を設定します。	–
max_pred_locks_per_relation	リレーションあたりの述語ロックページとタプルの最大数を設定します。	–
max_pred_locks_per_transaction	トランザクションあたりの述語ロックの最大数を設定します。	–

パラメータ名	説明	デフォルト値
max_slot_wal_keep_size	(MB) ディスク上の WAL がこの量のスペースを占有している場合、レプリケーションスロットは障害があるとマークされ、セグメントは削除またはリサイクルのために解放されます。	–
max_stack_depth	(kB) スタックの深度の最大値をキロバイト単位で設定します。	6144
max_standby_streaming_delay	(ミリ秒) ホットスタンバイサーバーがストリーミングされた WAL データを処理しているときにクエリをキャンセルするまでの最大遅延を設定します。	14000
max_worker_processes	同時ワーカープロセスの最大数を設定します。	GREATEST($DBInstanceVCPU*2, 8)
min_dynamic_shared_memory	(MB) 起動時に予約された動的共有メモリ量。	–
min_parallel_index_scan_size	(8kB) パラレルスキャンのインデックスデータの最小量を設定します。	–
min_parallel_table_scan_size	(8kB) パラレルスキャンのテーブルデータの最小量を設定します。	–
old_snapshot_threshold	(分) スナップショットが古すぎるためにスナップショットが取得された後に変更されたページ読み取ることができないと判定されるまでの時間。	–
parallel_leader_participation	Gather と Gather Merge もサブプランを実行するかどうかをコントロールします。	–
parallel_setup_cost	並列クエリのワーカープロセスを起動するコストのプランナーの見積もりを設定します。	–

パラメータ名	説明	デフォルト値
parallel_tuple_cost	各タプル (行) をワーカーからマスターバックエンドに渡す際のプランナーのコスト見積もりを設定します。	–
pgaudit.log	セッション監査ログ記録によってログに記録されるステートメントのクラスを指定します。	–
pgaudit.log_catalog	ステートメント内のすべてのリレーションが pg_catalog 内にある場合に、セッションログ記録を有効にするように指定します。	–
pgaudit.log_level	ログエントリに使用されるログレベルを指定します。	–
pgaudit.log_parameter	ステートメントとともに渡されたパラメータを監査ログ記録に含めるように指定します。	–
pgaudit.log_relation	セッション監査ログ記録で、SELECT ステートメントまたは DML ステートメントで参照されるリレーション (TABLE、VIEW など) ごとに個別のログエントリを作成するかどうかを指定します。	–
pgaudit.log_statement_once	ログ記録に、ステートメントテキストとパラメータを、ステートメントとサブステートメントの組み合わせの最初のログエントリとともに含めるか、すべてのエントリとともに含めるかを指定します。	–
pgaudit.role	オブジェクト監査ログ記録に使用するマスターロールを指定します。	–
pg_bigm.enable_recheck	これは、フルテキスト検索の内部プロセスである Recheck を実行するかどうかを指定します。	オン

パラメータ名	説明	デフォルト値
pg_bigm.gin_key_limit	これは、フルテキスト検索に使用される検索キーワードの 2-gram の最大数を指定します。	0
pg_bigm.last_update	pg_bigm モジュールの最終更新日が報告されます。	2013.11.22
pg_bigm.similarity_limit	類似性検索で使用される最小しきい値を指定します。	0.3
pg_hint_plan.debug_print	ヒント解析の結果を口グに記録します。	–
pg_hint_plan.enable_hint	プランナーが、クエリの前のヒントコメントで指定された計画を使用するようにします。	–
pg_hint_plan.enable_hint_table	プランナーが、テーブルルックアップを使用してヒントを取得しないようにします。	–
pg_hint_plan.message_level	デバッグメッセージのメッセージレベル。	–
pg_hint_plan.parse_messages	解析エラーのメッセージレベル。	–
pglogical.batch_inserts	可能であれば、バッチ挿入	–
pglogical.conflict_log_level	解決された競合の口グ記録に使用する口グレベルを設定します。	–
pglogical.conflict_resolution	解決可能な競合の競合解決に使用されるメソッドを設定します。	–
pglogical.extra_connection_options	すべてのピアノード接続に追加する接続オプション	–
pglogical.synchronous_commit	pglogical 固有の同期コミット値	–

パラメータ名	説明	デフォルト値
pglogical.use_spi	低レベル API の代わりに SPI を使用して変更を適用します	–
pg_similarity.block_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.block_threshold	ブロック類似度関数で使用されるしきい値を設定します。	–
pg_similarity.block_tokenizer	ブロック類似性関数のトークナイザを設定します。	–
pg_similarity.cosine_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.cosine_threshold	コサイン類似度関数で使用されるしきい値を設定します。	–
pg_similarity.cosine_tokenizer	コサイン類似度関数のトークナイザを設定します。	–
pg_similarity.dice_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.dice_threshold	Dice 類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.dice_tokenizer	Dice 類似度メジャーのトークナイザを設定します。	–
pg_similarity.euclidean_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.euclidean_threshold	ユークリッド類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.euclidean_tokenizer	ユークリッド類似度メジャーのトークナイザを設定します。	–

パラメータ名	説明	デフォルト値
pg_similarity.hamming_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.hamming_threshold	ブロック類似度メトリクスで使用されるしきい値を設定します。	–
pg_similarity.jaccard_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.jaccard_threshold	Jaccard 類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.jaccard_tokenizer	Jaccard 類似度メジャーのトークナイザを設定します。	–
pg_similarity.jaro_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.jaro_threshold	Jaro 類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.jarowinkler_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.jarowinkler_threshold	Jarowinkler 類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.levenshtein_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.levenshtein_threshold	Levenshtein 類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.matching_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.matching_threshold	マッチング係数の類似度メジャーで使用されるしきい値を設定します。	–

パラメータ名	説明	デフォルト値
pg_similarity.matching_tokenizer	マッチング係数の類似度メジャーのトークナイザを設定します。	–
pg_similarity.mongeelkan_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.mongeelkan_threshold	Monge-Elkan 類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.mongeelkan_tokenizer	Monge-Elkan 類似度メジャーのトークナイザを設定します。	–
pg_similarity.nw_gap_penalty	Needleman-Wunsch 類似度メジャーで使用されるギャップペナルティを設定します。	–
pg_similarity.nw_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.nw_threshold	Needleman-Wunsch 類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.overlap_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.overlap_threshold	オーバーラップ係数の類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.overlap_tokenizer	オーバーラップ係数の類似度メジャーのトークナイザを設定します。	–
pg_similarity.qgram_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.qgram_threshold	q-gram 類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.qgram_tokenizer	q-gram メジャーのトークナイザを設定します。	–

パラメータ名	説明	デフォルト値
pg_similarity.swg_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.swg_threshold	Smith-Waterman-Gotoh 類似度メジャーで使用されるしきい値を設定します。	–
pg_similarity.sw_is_normalized	結果の値が正規化されるかどうかを設定します。	–
pg_similarity.sw_threshold	Smith-Waterman 類似度メジャーで使用されるしきい値を設定します。	–
pg_stat_statements.max	pg_stat_statements によって追跡されるステートメントの最大数を設定します。	–
pg_stat_statements.save	pg_stat_statements 統計情報をサーバーのシャットダウン全体にわたって保存します。	–
pg_stat_statements.track	pg_stat_statements によって追跡されるステートメントを選択します。	–
pg_stat_statements.track_planning	pg_stat_statements で計画期間を追跡するかどうかを選択します。	–
pg_stat_statements.track_utility	pg_stat_statements によってユーティリティコマンドを追跡するかどうかを選択します。	–
postgis.gdal_enabled_drivers	Postgres 9.3.5 以降で PostGIS で使用される GDAL ドライバーを有効または無効にします。	ENABLE_ALL
quote_all_identifiers	SQL フラグメントを生成するときに、すべての識別子を引用符で囲みます。	–
random_page_cost	非連続的に取得されたディスクページのコストに対するプランナーの見積もりを設定します。	–

パラメータ名	説明	デフォルト値
rds.enable_memory_management	Aurora PostgreSQL 12.17、13.13、14.10、15.5 以降のバージョンでは、メモリ管理機能が向上し、空きメモリ不足による安定性の問題やデータベースの再起動を未然に防ぐことができます。詳細については、「Aurora PostgreSQL のメモリ管理が改善されました」を参照してください。	True
rds.force_admin_logging_level	カスタマーデータベースの RDS 管理者ユーザーアクションのログメッセージを参照してください。	–
rds.log_retention_period	N 分より古い PostgreSQL ログは Amazon RDS で削除されます。	4320
rds.memory_allocation_guard	Aurora PostgreSQL 11.21、12.16、13.12、14.9、 15.4 以前のバージョンのメモリ管理機能が向上し、空きメモリ不足による安定性の問題やデータベースの再起動が防止されました。詳細については、「Aurora PostgreSQL のメモリ管理が改善されました」を参照してください。	False
rds.pg_stat_ramdisk_size	統計ラムディスクのサイズ (MB 単位)。ゼロ以外の値を指定すると、ラムディスクがセットアップされます。	0
rds.rds_superuser_reserved_connections	rds_superusers 用に予約されている接続スロットの数を設定します。このパラメータは、バージョン 15 以前でのみ使用できます。詳細については、PostgreSQL ドキュメントの「reserved connections」を参照してください。	2
rds.superuser_variables	rds_superuser 変更ステートメントを昇格させるスーパーユーザー専用変数のリスト。	session_replication_role

パラメータ名	説明	デフォルト値
remove_temp_files_after_crash	バックエンドクラッシュ後に一時ファイルを削除します。	0
restart_after_crash	バックエンドクラッシュ後にサーバーを再初期化します。	–
row_security	行セキュリティを有効にします。	–
search_path	スキーマによって修飾されていない名前でスキーマを検索する順序を設定します。	–
seq_page_cost	連続的に取得されたディスクページのコストに対するプランナーの見積もりを設定します。	–
session_replication_role	トリガーと再書き込みルールに対するセッション動作を設定します。	–
shared_buffers	(8kB) サーバーで使用される共有メモリバッファの数を設定します。	SUM(DBInstanceClassMemory/12038,-50003)
shared_preload_libraries	サーバーにプリロードする共有ライブラリを一覧表示します。	pg_stat_statements
ssl_ca_file	SSL サーバー権限ファイルの場所。	/rdsdbdata/rds-metadata/ca-cert.pem
ssl_cert_file	SSL サーバー証明書ファイルの場所。	/rdsdbdata/rds-metadata/server-cert.pem
ssl_crl_dir	SSL 証明書の失効リストディレクトリの場所。	/rdsdbdata/rds-metadata/ssl_crl_dir/
ssl_key_file	SSL サーバーのプライベートキーファイルの場所	/rdsdbdata/rds-metadata/server-key.pem
standard_conforming_strings	... 文字列をリテラルのバックスラッシュとして扱います。	–

パラメータ名	説明	デフォルト値
statement_timeout	(ミリ秒) すべてのステートメントに許可される最大実行時間を設定します。	–
stats_temp_directory	統計情報の一時ファイルを指定したディレクトリに書き込みます。	/rdsdbdata/db/pg_stat_tmp
superuser_reserved_connections	スーパーユーザー用に予約されている接続スロットの数を設定します。	3
synchronize_seqscans	シーケンシャルスキャンの同期を有効にします。	–
tcp_keepalives_count	TCP キープアライブを再送信する最大回数。	–
tcp_keepalives_idle	(秒) TCP キープアライブを発行する時間間隔。	–
tcp_keepalives_interval	(秒) TCP キープアライブを再送信する時間間隔。	–
temp_buffers	(8kB) 各セッションで使用される一時バッファの最大数を設定します。	–
temp_file_limit	特定の PostgreSQL プロセスが一時ファイルに使用できるディスク容量の合計をキロバイト単位で制限します。明示的な一時テーブルに使用される領域は除きます。	-1
temp_tablespaces	一時テーブルとソートファイルで使用するテーブルスペースを設定します。	–
track_activities	実行中のコマンドに関する情報を収集します。	–
track_activity_query_size	pg_stat_activity.current_query 用に予約するサイズをバイト単位で設定します。	4096
track_counts	データベースアクティビティの統計情報を収集します。	–

パラメータ名	説明	デフォルト値
track_functions	データベースアクティビティの関数レベルの統計情報を収集します。	pl
track_io_timing	データベース IO アクティビティのタイミングに関する統計情報を収集します。	1
transform_—_equals	expr=– as expr IS – として扱います。	–
update_process_title	アクティブな SQL コマンドを表示するようにプロセスのタイトルを更新します。	–
wal_receiver_status_interval	(秒) WAL レシーバーステータスレポートをプライマリにするための最大時間間隔を設定します。	–
work_mem	(kB) クエリワークスペースに使用するメモリの最大量を設定します。	–
xmlbinary	バイナリ値を XML にエンコードする方法を設定します。	–
xmloption	黙示的な分析とシリアル化オペレーションでの XML データをドキュメントとして見なすか、コンテンツのフラグメントとして見なすかを設定します。	–

Amazon Aurora PostgreSQL のイベント

以下は、Aurora PostgreSQL の代表的な待機イベントです。待機イベントと Aurora PostgreSQL DB クラスターのチューニングの詳細については、「Aurora PostgreSQL の待機イベントでのチューニング」を参照してください。

Activity:ArchiverMain

アーカイバープロセスはアクティビティを待っています。

Activity:AutoVacuumMain

自動バキュームランチャープロセスはアクティビティを待っています。

Activity:BgWriterHibernate

バックグラウンドライタープロセスは、アクティビティを待っている間は休止状態になっています。

Activity:BgWriterMain

バックグラウンドライタープロセスはアクティビティを待っています。

Activity:CheckpointerMain

チェックポインタープロセスはアクティビティを待っています。

Activity:LogicalApplyMain

論理的なレプリケーション適用プロセスがアクティビティを待っています。

Activity:LogicalLauncherMain

論理的なレプリケーションランチャープロセスはアクティビティを待っています。

Activity:PgStatMain

統計コレクタプロセスはアクティビティを待っています。

Activity:RecoveryWalAll

プロセスは、リカバリ時にストリーミングからのログ先行書き込み (WAL) を待っています。

Activity:RecoveryWalStream

起動プロセスは、ストリーミングリカバリ中にログ先行書き込み (WAL) が到着するのを待っています。

Activity:SysLoggerMain

syslogger プロセスはアクティビティを待っています。

Activity:WalReceiverMain

ログ先行書き込み (WAL) レシーバプロセスがアクティビティを待っています。

Activity:WalSenderMain

ログ先行書き込み (WAL) 送信者プロセスがアクティビティを待っています。

Activity:WalWriterMain

ログ先行書き込み (WAL) ライタープロセスがアクティビティを待っています。

BufferPin:BufferPin

プロセスがバッファ上の排他 PIN を取得するのを待っています。

Client:GSSOpenServer

汎用セキュリティサービスアプリケーションプログラムインターフェイス (GSSAPI) セッションの確立中に、プロセスがクライアントからのデータの読み取りを待っています。

Client:ClientRead

バックエンドプロセスは PostgreSQL クライアントからのデータ受信を待っています。詳細については、「クライアント: ClientRead」を参照してください。

Client:ClientWrite

バックエンドプロセスが PostgreSQL クライアントにより多くのデータを送信するのを待っています。詳細については、「クライアント: ClientWrite」を参照してください。

Client:LibPQWalReceiverConnect

プロセスはログ先行書き込み (WAL) レシーバで、リモートサーバーへの接続が確立するのを待っています。

Client:LibPQWalReceiverReceive

プロセスはログ先行書き込み (WAL) レシーバで、リモートサーバーからのデータ受信を待っています。

Client:SSLOpenServer

プロセスは、接続の試行中に Secure Sockets Layer (SSL) を待っています。

Client:WalReceiverWaitStart

プロセスは、スタートアッププロセスがストリーミングレプリケーションの初期データを送信するのを待っています。

Client:WalSenderWaitForWAL

プロセスは、WAL 送信者プロセスでログ先行書き込み (WAL) がフラッシュされるのを待っています。

Client:WalSenderWriteData

WAL 送信者プロセスでログ先行書き込み (WAL) レシーバからの応答を処理するときに、プロセスはアクティビティを待っています。

CPU

バックエンドプロセスは CPU でアクティブであるか、CPU を待っています。詳細については、「CPU」を参照してください。

Extension:extension

バックエンドプロセスは、拡張機能またはモジュールによって定義された条件を待っています。

IO:AuroraOptimizedReadsCacheRead

プロセスは、ページが共有メモリで使用できないため、Optimized Reads 階層型キャッシュからの読み取りを待っています。

IO:AuroraOptimizedReadsCacheSegmentTruncate

プロセスは Optimized Reads 階層型キャッシュセグメントファイルが切り捨てられるのを待っています。

IO:AuroraOptimizedReadsCacheWrite

バックグラウンドライタープロセスは、Optimized Reads 階層型キャッシュへの書き込みを待っています。

IO:AuroraStorageLogAllocate

セッションはメタデータを割り当てて、トランザクションログの書き込みを準備しています。

IO:BufFileRead

作業メモリパラメータで定義された容量よりも多くのメモリが必要な操作の場合、エンジンはディスク上に一時ファイルを作成します。この待機イベントは、操作が一時ファイルから読み込まれたときに発生します。詳細については、「IO:BufFileRead および IO:BufFileWrite」を参照してください。

IO:BufFileWrite

作業メモリパラメータで定義された容量よりも多くのメモリが必要な操作の場合、エンジンはディスク上に一時ファイルを作成します。この待機イベントは、操作が一時ファイルに書き込むときに発生します。詳細については、「IO:BufFileRead および IO:BufFileWrite」を参照してください。

IO:ControlFileRead

プロセスが、pg_control ファイルからの読み取りを待っています。

IO:ControlFileSync

プロセスは、pg_control ファイルが耐久性の高いストレージに到達するのを待っています。

IO:ControlFileSyncUpdate

プロセスは、pg_control ファイルへの更新が耐久性の高いストレージに到達するのを待っています。

IO:ControlFileWrite

プロセスが、pg_control ファイルへの書き込みを待っています。

IO:ControlFileWriteUpdate

プロセスは、pg_control ファイルを更新するための書き込みを待っています。

IO:CopyFileRead

プロセスは、ファイルコピー操作中の読み取りを待っています。

IO:CopyFileWrite

プロセスは、ファイルコピー操作中の書き込みを待っています。

IO:DataFileExtend

プロセスは、リレーションデータファイルが拡張されるのを待っています。す。

IO:DataFileFlush

プロセスは、リレーションデータファイルが耐久性の高いストレージに到達するのを待っています。

IO:DataFileImmediateSync

プロセスは、リレーションデータファイルと耐久性の高いストレージとの即時同期を待っています。

IO:DataFilePrefetch

プロセスは、リレーションデータファイルからの非同期プリフェッチを待っています。

IO:DataFileSync

プロセスは、リレーションデータファイルの変更が耐久性の高いストレージに到達するのを待っています。

IO:DataFileRead

バックエンドプロセスが共有バッファ内のページを見つけようとしましたが、見つからなかったので、ストレージから読み取りました。詳細については、「IO:DataFileRead」を参照してください。

IO:DataFileTruncate

プロセスは、リレーションデータファイルが切り捨てられるのを待っています。

IO:DataFileWrite

プロセスは、リレーションデータファイルへの書き込みを待っています。

IO:DSMFillZeroWrite

プロセスは、動的共有メモリバッキングファイルにゼロバイトを書き込むのを待っています。

IO:LockFileAddToDataDirRead

プロセスが、データディレクトリロックファイルに行を追加している間、読み取りを待っています。

IO:LockFileAddToDataDirSync

プロセスは、データディレクトリロックファイルに行を追加している間、データが耐久性の高いストレージに到達するのを待っています。

IO:LockFileAddToDataDirWrite

プロセスが、データディレクトリロックファイルに行を追加している間、書き込みを待っています。

IO:LockFileCreateRead

プロセスが、データディレクトリロックファイルの作成中に読み取りを待っています。

IO:LockFileCreateSync

プロセスが、データディレクトリロックファイルを作成している間に、データが耐久性の高いストレージに到達するのを待っています。

IO:LockFileCreateWrite

プロセスが、データディレクトリロックファイルの作成中に書き込みを待っています。

IO:LockFileReCheckDataDirRead

プロセスが、データディレクトリロックファイルの再チェック中に読み取りを待っています。

IO:LogicalRewriteCheckpointSync

プロセスは、チェックポイント中に論理的な書き換えマッピングが耐久性の高いストレージに到達するのを待っています。

IO:LogicalRewriteMappingSync

プロセスは、論理的な書き換え中にマッピングデータが耐久性の高いストレージに到達するのを待っています。

IO:LogicalRewriteMappingWrite

プロセスは、論理的な書き換え中にマッピングデータの書き込みを待っています。

IO:LogicalRewriteSync

プロセスは、論理的な書き換えマッピングが耐久性の高いストレージに到達するのを待っています。

IO:LogicalRewriteTruncate

プロセスは、論理的な書き換え中にマッピングデータの切り捨てを待っています。

IO:LogicalRewriteWrite

プロセスは、論理的な書き換えマッピングの書き込みを待っています。

IO:RelationMapRead

プロセスは、リレーションマップファイルの読み取りを待っています。

IO:RelationMapSync

プロセスは、リレーションマップファイルが耐久性の高いストレージに到達するのを待っています。

IO:RelationMapWrite

プロセスは、リレーションマップファイルへの書き込みを待っています。

IO:ReorderBufferRead

プロセスは、リオーダバッファの管理中に読み取りを待っています。

IO:ReorderBufferWrite

プロセスは、リオーダバッファの管理中に書き込みを待っています。

IO:ReorderLogicalMappingRead

プロセスは、リオーダバッファの管理中に論理的なマッピングの読み取りを待っています。

IO:ReplicationSlotRead

プロセスは、レプリケーションスロット制御ファイルからの読み取りを待っています。

IO:ReplicationSlotRestoreSync

プロセスは、レプリケーションスロット制御ファイルがメモリに復元される間、耐久性の高いストレージに到達するのを待っています。

IO:ReplicationSlotSync

プロセスは、レプリケーションスロット制御ファイルが耐久性の高いストレージに到達するのを待っています。

IO:ReplicationSlotWrite

プロセスは、レプリケーションスロット制御ファイルへの書き込みを待っています。

IO:SLRUFlushSync

プロセスは、チェックポイントまたはデータベースのシャットダウン中に、シンプルな最も長い時間使われていない (SLRU) データが耐久ストレージに到達するのを待っています。

IO:SLRURead

プロセスは、シンプルな最も長い時間使われていない (SLRU) ページの読み取りを待っています。

IO:SLRUSync

プロセスは、シンプルな最も長い時間使われていない (SLRU) データが、ページの書き込み後に耐久性の高いストレージへ到達するのを待っています。

IO:SLRUWrite

プロセスは、シンプルな最も長い時間使われていない (SLRU) ページの書き込みを待っています。

IO:SnapbuildRead

プロセスは、シリアル化された履歴カタログのスナップショットの読み取りを待っています。

IO:SnapbuildSync

プロセスは、シリアル化された履歴カタログのスナップショットが耐久性の高いストレージに到達するのを待っています。

IO:SnapbuildWrite

プロセスは、シリアル化された履歴カタログのスナップショットの書き込みを待っています。

IO:TimelineHistoryFileSync

プロセスは、ストリーミングレプリケーションで受信したタイムラインの履歴ファイルが耐久性の高いストレージに到達するのを待っています。

IO:TimelineHistoryFileWrite

プロセスは、ストリーミングレプリケーションで受信したタイムラインの履歴ファイルの書き込みを待っています。

IO:TimelineHistoryFileWrite

プロセスは、タイムラインの履歴ファイルの読み取りを待っています。

IO:TimelineHistorySync

プロセスは、新しく作成されたタイムラインの履歴ファイルが耐久性の高いストレージに到達するのを待っています。

IO:TimelineHistoryWrite

プロセスは、新しく作成されたタイムラインの履歴ファイルの書き込みを待っています。

IO:TimelineHistoryWrite

プロセスは 2 フェーズ状態のファイルの読み取りを待っています。

IO:TwophaseFileSync

プロセスは、2 フェーズ状態のファイルが耐久性の高いストレージに到達するのを待っています。

IO:TwophaseFileWrite

プロセスは 2 フェーズ状態のファイルの書き込みを待っています。

IO:WALBootstrapSync

プロセスは、ブートストラップ中にログ先行書き込み (WAL) が耐久性の高いストレージに到達するのを待っています。

IO:WALBootstrapWrite

プロセスは、ブートストラップ中にログ先行書き込み (WAL) ページの書き込みを待っています。

IO:WALCopyRead

既存のものをコピーして、新しいログ先行書き込み (WAL) セグメントを作成するときに、プロセスは読み取りを待っています。

IO:WALCopySync

プロセスは、既存のものをコピーして作成した、新しいログ先行書き込み (WAL) セグメントが耐久性の高いストレージに到達するのを待っています。

IO:WALCopyWrite

既存のものをコピーして新しいログ先行書き込み (WAL) セグメントを作成するときに、プロセスは書き込みを待っています。

IO:WALInitSync

プロセスは、新しく初期化されたログ先行書き込み (WAL) ファイルが耐久性の高いストレージに到達するのを待っています。

IO:WALInitWrite

新しいログ先行書き込み (WAL) ファイルを初期化中に、プロセスが書き込みを待っています。

IO:WALRead

プロセスはログ先行書き込み (WAL) ファイルからの読み取りを待っています。

IO:WALSenderTimelineHistoryRead

プロセスは、WAL 送信者のタイムラインコマンド中に、タイムラインの履歴ファイルからの読み取りを待っています。

IO:WALSync

プロセスは、ログ先行書き込み (WAL) ファイルが耐久性の高いストレージに到達するのを待っています。

IO:WALSyncMethodAssign

プロセスは、新しいログ先行書き込み (WAL) 同期メソッドを割り当てる間、データが耐久性の高いストレージに到達するのを待っています。

IO:WALWrite

プロセスはログ先行書き込み (WAL) ファイルへの書き込みを待っています。

IO:XactSync

バックエンドプロセスは、Aurora ストレージサブシステムが通常のトランザクションのコミット、または準備済みトランザクションのコミットあるいはロールバックを承認するのを待っています。詳細については、「IO:XactSync」を参照してください。

IPC:BackupWaitWalArchive

プロセスは、バックアップを正常にアーカイブするために必要なログ先行書き込み (WAL) ファイルを待っています。

IPC:AuroraOptimizedReadsCacheWriteStop

プロセスは、バックグラウンドライターが Optimized Reads 階層型キャッシュへの書き込みを停止するのを待っています。

IPC:BgWorkerShutdown

プロセスはバックグラウンドワーカーがシャットダウンするのを待っています。

IPC:BgWorkerStartup

プロセスはバックグラウンドワーカーのスタートを待っています。

IPC:BtreePage

プロセスは、パラレル B ツリースキャンの継続に必要なページ番号が使用可能になるのを待っています。

IPC:CheckpointDone

プロセスは、チェックポイントが完了するのを待っています。

IPC:CheckpointStart

プロセスがチェックポイントのスタートを待っています。

IPC:ClogGroupUpdate

プロセスは、グループリーダーがトランザクションの終了時にトランザクションステータスを更新するのを待っています。

IPC:DamRecordTxAck

バックエンドプロセスがデータベースアクティビティストリーミングイベントを生成し、イベントが永続化するのを待っています。詳細については、「IPC:DamRecordTxAck」を参照してください。

IPC:ExecuteGather

プロセスは、Gather Plan ノードの実行中に、子プロセスからのアクティビティを待っています。

IPC:Hash/Batch/Allocating

プロセスは、選択されたパラレルハッシュ参加者がハッシュテーブルを割り当てるのを待っています。

IPC:Hash/Batch/Electing

プロセスは、ハッシュテーブルを割り当てるためにパラレルハッシュ参加者を選択しています。

IPC:Hash/Batch/Loading

プロセスは、他のパラレルハッシュ参加者がハッシュテーブルのロードを完了するのを待っています。

IPC:Hash/Build/Allocating

プロセスは、選択されたパラレルハッシュ参加者が初期のハッシュテーブルを割り当てるのを待っています。

IPC:Hash/Build/Electing

プロセスは、初期のハッシュテーブルを割り当てるためにパラレルハッシュ参加者を選択しています。

IPC:Hash/Build/HashingInner

プロセスは、他のパラレルハッシュ参加者が内部リレーションのハッシュを完了するのを待っています。

IPC:Hash/Build/HashingOuter

プロセスは、他のパラレルハッシュ参加者が外部リレーションのパーティショニングを完了するのを待っています。

IPC:Hash/GrowBatches/Allocating

プロセスは、選択されたパラレルハッシュ参加者がより多くのバッチを割り当てるのを待っています。

IPC:Hash/GrowBatches/Deciding

プロセスは、将来のバッチの増加を決定するために、パラレルハッシュ参加者を選択しています。

IPC:Hash/GrowBatches/Electing

プロセスは、より多くのバッチを割り当てるために、パラレルハッシュ参加者を選択しています。

IPC:Hash/GrowBatches/Finishing

プロセスは、選択されたパラレルハッシュ参加者が将来のバッチの増加を決定するのを待っています。

IPC:Hash/GrowBatches/Repartitioning

プロセスは、他のパラレルハッシュ参加者がリパーティショニングを完了させるのを待っています。

IPC:Hash/GrowBuckets/Allocating

プロセスは、選択されたパラレルハッシュ参加者がより多くのバケットの割り当てを完了するのを待っています。

IPC:Hash/GrowBuckets/Electing

プロセスは、より多くのバケットを割り当てるためにパラレルハッシュ参加者を選択しています。

IPC:Hash/GrowBuckets/Reinserting

プロセスは、他のパラレルハッシュ参加者が新しいバケットへタプル挿入を完了するのを待っています。

IPC:HashBatchAllocate

プロセスは、選択されたパラレルハッシュ参加者がハッシュテーブルを割り当てるのを待っています。

IPC:HashBatchElect

プロセスは、ハッシュテーブルを割り当てるパラレルハッシュ参加者を選択するのを待っています。

IPC:HashBatchLoad

プロセスは、他のパラレルハッシュ参加者がハッシュテーブルのロードを完了するのを待っています。

IPC:HashBuildAllocate

プロセスは、選択されたパラレルハッシュ参加者が初期ハッシュテーブルを割り当てるのを待っています。

IPC:HashBuildElect

プロセスは、初期のハッシュテーブルを割り当てるパラレルハッシュ参加者を選択するのを待っています。

IPC:HashBuildHashInner

プロセスは、他のパラレルハッシュ参加者が内部リレーションのハッシュを完了するのを待っています。

IPC:'HashBuildHashOuter

プロセスは、他のパラレルハッシュ参加者が外部リレーションのパーティショニングを完了するのを待っています。

IPC:HashGrowBatchesAllocate

プロセスは、選択されたパラレルハッシュ参加者がより多くのバッチを割り当てるのを待っています。

IPC:'HashGrowBatchesDecide

プロセスは、将来のバッチの増加を決定するために、パラレルハッシュ参加者を選択するのを待っています。

IPC:HashGrowBatchesElect

プロセスは、より多くのバッチを割り当てるために、パラレルハッシュ参加者を選択するのを待っています。

IPC:HashGrowBatchesFinish

プロセスは、選択されたパラレルハッシュ参加者が将来のバッチの増加を決定するのを待っています。

IPC:HashGrowBatchesRepartition

プロセスは、他のパラレルハッシュ参加者が再パーティショニングを完了するのを待っています。

IPC:HashGrowBucketsAllocate

プロセスは、選択されたパラレルハッシュ参加者がより多くのバケットの割り当てを完了するのを待っています。

IPC:HashGrowBucketsElect

プロセスは、より多くのバケットを割り当てるためにパラレルハッシュ参加者を選択するのを待っています。

IPC:HashGrowBucketsReinsert

プロセスは、他のパラレルハッシュ参加者が新しいバケットへのタプルの挿入を完了するのを待っています。

IPC:LogicalSyncData

プロセスは、論理的なレプリケーションリモートサーバーが初期のテーブル同期のためにデータを送信するのを待っています。

IPC:LogicalSyncStateChange

プロセスは、論理的なレプリケーションリモートサーバーが状態を変更するのを待っています。

IPC:MessageQueueInternal

プロセスは、別のプロセスが共有メッセージキューに接続されるのを待っています。

IPC:MessageQueuePutMessage

プロセスは、共有メッセージキューへのプロトコルメッセージの書き込みを待っています。

IPC:MessageQueueReceive

プロセスは、共有メッセージキューからバイトを受信するのを待っています。

IPC:MessageQueueSend

プロセスは、共有メッセージキューにバイトを送信するのを待っています。

IPC:ParallelBitmapScan

プロセスは、パラレルビットマップスキャンが初期化されるのを待っています。

IPC:ParallelCreateIndexScan

プロセスは、パラレルの CREATE INDEX ワーカーがヒープスキャンを完了するのを待っています。

IPC:ParallelFinish

プロセスは、パラレルワーカーがコンピューティングを完了するのを待っています。

IPC:ProcArrayGroupUpdate

プロセスは、パラレルオペレーションの最後にグループリーダーがトランザクション ID をクリアするのを待っています。

IPC:ProcSignalBarrier

プロセスは、バリアイベントがすべてのバックエンドで処理されるのを待っています。

IPC:Promote

プロセスがスタンバイのプロモーションを待っています。

IPC:RecoveryConflictSnapshot

プロセスが、バキュームクリーンアップのリカバリの競合解決を待っています。

IPC:RecoveryConflictTablespace

プロセスは、表領域を削除するためのリカバリの競合解決を待っています。

IPC:RecoveryPause

プロセスがリカバリが再開されるのを待っています。

IPC:ReplicationOriginDrop

プロセスは、レプリケーションオリジンが非アクティブになるのを待っているため、削除できます。

IPC:ReplicationSlotDrop

プロセスは、レプリケーションスロットが非アクティブになるのを待っているため、ドロップできます。

IPC:SafeSnapshot

プロセスは、読み取り専用の遅延可能なトランザクションの有効なスナップショットを取得するのを待っています。

IPC:SyncRep

プロセスは、同期レプリケーション中にリモートサーバーからの確認を待っています。

IPC:XactGroupUpdate

プロセスは、パラレルオペレーションの終了時にグループリーダーがトランザクションステータスを更新するのを待っています。

Lock:advisory

バックエンドプロセスがアドバイザリロックを要求し、それを待っています。詳細については、「Lock:advisory」を参照してください。

Lock:extend

バックエンドプロセスは、リレーションを拡張できるように、ロックが解放されるのを待っています。このロックは、一度に 1 つのバックエンドプロセスだけのリレーションを拡張できるので必要です。詳細については、「Lock:extend」を参照してください。

Lock:frozenid

プロセスが更新を待っている`pg_database.datfrozenxid`そして`pg_database.datminmxid`。

Lock:object

プロセスは、非関係データベースオブジェクトのロックを取得するのを待っています。

Lock:page

プロセスは、リレーションのページのロックを取得するのを待っています。

Lock:Relation

バックエンドプロセスは、別のトランザクションによってロックされているリレーションのロック取得を待っています。詳細については、「Lock:Relation」を参照してください。

Lock:spectoken

プロセスが投機的挿入ロックを取得するのを待っています。

Lock:speculative token

プロセスが投機的挿入ロックを取得するのを待っています。

Lock:transactionid

トランザクションが行レベルのロックを待っています。詳細については、「Lock:transactionid」を参照してください。

Lock:tuple

バックエンドプロセスがタプルのロックを取得するのを待っている間、別のバックエンドプロセスが同じタプルで競合するロックを保持しています。詳細については、「Lock:tuple」を参照してください。

Lock:userlock

プロセスがユーザーロックを取得するのを待っています。

Lock:virtualxid

プロセスが仮想トランザクション ID ロックの取得を待っています。

LWLock:AddinShmemInit

プロセスは、共有メモリ内の拡張機能の領域割り当てを管理するのを待っています。

LWLock:AddinShmemInitLock

プロセスは、共有メモリ内の領域割り当ての管理を待っています。

LWLock:async

プロセスが非同期 (通知) バッファで I/O を待っています。

LWLock:AsyncCtlLock

プロセスが共有通知状態の読み取りまたは更新を待っています。

LWLock:AsyncQueueLock

プロセスが通知メッセージの読み取りまたは更新を待っています。

LWLock:AuroraOptimizedReadsCacheMapping

プロセスは Optimized Reads 階層型キャッシュ内のページにデータブロックを関連付けるのを待っています。

LWLock:AutoFile

プロセスが更新を待っている`postgresql.auto.conf`ファイルを開きます。

LWLock:AutoFileLock

プロセスが更新を待っている`postgresql.auto.conf`ファイルを開きます。

LWLock:Autovacuum

プロセスは autovacuum ワーカーの現在の状態の読み取りまたは更新を待っています。

Lwlock:AutovacuumLock

autovacuum ワーカーまたはランチャーは、自動バキュームワーカーの現在の状態の更新または読み取りを待っています。

Lwlock:AutovacuumSchedule

自動バキューム用に選択されたテーブルがまだバキューム処理が必要であることを確認するために、プロセスが待っています。

Lwlock:AutovacuumScheduleLock

プロセスは、バキューム用に選択したテーブルがまだバキューム処理が必要であることを確認するために待っています。

Lwlock:BackendRandomLock

プロセスが乱数の生成を待っています。

Lwlock:BackgroundWorker

プロセスは、バックグラウンドワーカーの状態の読み取りまたは更新を待っています。

Lwlock:BackgroundWorkerLock

プロセスは、バックグラウンドワーカーの状態の読み取りまたは更新を待っています。

Lwlock:BtreeVacuum

プロセスが B ツリーインデックスのバキューム関連情報の読み取りまたは更新を待っています。

Lwlock:BtreeVacuumLock

プロセスが B ツリーインデックスのバキューム関連情報の読み取りまたは更新を待っています。

LWLock:buffer_content

バックエンドプロセスは、共有メモリバッファの内容に対する軽量ロックの取得を待っています。詳細については、「LWLock:buffer_content (BufferContent)」を参照してください。

LWLock:buffer_mapping

バックエンドプロセスは、共有バッファプール内のバッファにデータブロックを関連付けるのを待っています。詳細については、「LWLock:buffer_mapping」を参照してください。

LWLock:BufferIO

バックエンドプロセスは、ページを共有メモリに読み込もうとしています。プロセスは、他のプロセスがそのページの I/O を完了するのを待っています。詳細については、「LWLock:BufferIO (IPC:BufferIO)」を参照してください。

Lwlock:Checkpoint

プロセスがチェックポイントのスタートを待っています。

Lwlock:CheckpointLock

プロセスがチェックポイントの実行を待っています。

Lwlock:CheckpointerComm

プロセスが管理を待っている`fsync`リクエスト。

Lwlock:CheckpointerCommLock

プロセスが管理を待っている`fsync`リクエスト。

Lwlock:clog

プロセスがクログ (トランザクションステータス) バッファの I/O を待っています。

Lwlock:CLogControlLock

プロセスは、トランザクションステータスの読み取りまたは更新を待っています。

Lwlock:CLogTruncationLock

プロセスの実行を待っている`txid_status`または、使用可能な最も古いトランザクション ID を更新します。

Lwlock:commit_timestamp

プロセスは、コミットタイムスタンプバッファでの I/O を待っています。

Lwlock:CommitTs

プロセスは、トランザクションコミットタイムスタンプに設定された最後の値の読み取りまたは更新を待っています。

Lwlock:CommitTsBuffer

プロセスは、コミットタイムスタンプのシンプルな最も長い時間使われていない (SLRU) バッファの I/O を待っています。

Lwlock:CommitTsControlLock

プロセスは、トランザクションのコミットタイムスタンプの読み取りまたは更新を待っています。

Lwlock:CommitTsLock

プロセスは、トランザクションタイムスタンプに設定された最後の値の読み取りまたは更新を待っています。

Lwlock:CommitTsSLRU

プロセスは、コミットタイムスタンプのシンプルな最も長い時間使われていない (SLRU) キャッシュにアクセスするのを待っています。

Lwlock:ControlFile

プロセスは、pg_control ファイルの読み取りまたは更新、または新しいログ先行書き込み (WAL) ファイルの作成を待っています。

Lwlock:ControlFileLock

プロセスは、制御ファイルの読み取りまたは更新、または新しいログ先行書き込み (WAL) ファイルの作成を待っています。

Lwlock:DynamicSharedMemoryControl

プロセスは、動的共有メモリ割り当て情報の読み取りまたは更新を待っています。

Lwlock:DynamicSharedMemoryControlLock

プロセスは、動的共有メモリ状態の読み取りまたは更新を待っています。

LWLock:lock_manager

バックエンドプロセスは、バックエンドプロセスのロックの追加または検査を待っています。または、パラレルクエリで使用されるロックグループに参加または終了するのを待っています。詳細については、「LWLock:lock_manager」を参照してください。

Lwlock:LockFastPath

プロセスは、プロセスの高速パスロック情報の読み取りまたは更新を待っています。

Lwlock:LogicalRepWorker

プロセスは、論理的なレプリケーションワーカーの状態の読み取りまたは更新を待っています。

Lwlock:LogicalRepWorkerLock

プロセスは、論理的なレプリケーションワーカーに対するアクションが完了するのを待っています。

Lwlock:multixact_member

プロセスは multixact_member バッファ上の I/O を待っています。

Lwlock:multixact_offset

プロセスは multixact オフセットバッファ上の I/O を待っています。

Lwlock:MultiXactGen

プロセスが共有 multixact 状態の読み取りまたは更新を待っています。

Lwlock:MultiXactGenLock

プロセスは、共有マルチシャクト状態の読み取りまたは更新を待っています。

Lwlock:MultiXactMemberBuffer

プロセスは、multixact メンバーのシンプルな最も長い時間使われていない (SLRU) バッファの I/O を待っています。詳細については、「LWLock:MultiXact」を参照してください。

Lwlock:MultiXactMemberControlLock

プロセスは、multixact メンバーマッピングの読み取りまたは更新を待っています。

Lwlock:MultiXactMemberSLRU

プロセスは、multixact メンバーのシンプルな最も長い時間使われていない (SLRU) キャッシュへのアクセスを待っています。詳細については、「LWLock:MultiXact」を参照してください。

Lwlock:MultiXactOffsetBuffer

プロセスは、multixact オフセットのシンプルな最も長い時間使われていない (SLRU) バッファの I/O を待っています。詳細については、「LWLock:MultiXact」を参照してください。

Lwlock:MultiXactOffsetControlLock

プロセスは、multixact オフセットマッピングの読み取りまたは更新を待っています。

Lwlock:MultiXactOffsetSLRU

プロセスは、multixact オフセットのシンプルな最も長い時間使われていない (SLRU) キャッシュへのアクセスを待っています。詳細については、「LWLock:MultiXact」を参照してください。

Lwlock:MultiXactTruncation

プロセスは、multixact 情報の読み取りまたは切り捨てを待っています。

Lwlock:MultiXactTruncationLock

プロセスは、multixact 情報の読み取りまたは切り捨てを待っています。

Lwlock:NotifyBuffer

プロセスは、NOTIFY メッセージのシンプルな最も長い時間使われていない (SLRU) バッファの I/O を待っています。

Lwlock:NotifyQueue

プロセスは NOTIFY メッセージの読み取りまたは更新を待っています。

Lwlock:NotifyQueueTail

プロセスが NOTIFY メッセージストレージの上限を更新するのを待っています。

Lwlock:NotifyQueueTailLock

プロセスが通知メッセージストレージの上限を更新するのを待っています。

Lwlock:NotifySLRU

プロセスは、NOTIFY メッセージのシンプルな最も長い時間使われていない (SLRU) キャッシュへのアクセスを待っています。

Lwlock:OidGen

プロセスは、新しいオブジェクト ID (OID) の割り当てを待っています。

Lwlock:OidGenLock

プロセスは、オブジェクト ID (OID) の割り当てまたは割り当てを待っています。

Lwlock:oldserxid

プロセスは oldserxid バッファ上の I/O を待っています。

Lwlock:OldSerXidLock

プロセスは、競合する直列化可能なトランザクションの読み取りまたは記録を待っています。

Lwlock:OldSnapshotTimeMap

プロセスは、古いスナップショット制御情報の読み取りまたは更新を待っています。

Lwlock:OldSnapshotTimeMapLock

プロセスは、古いスナップショット制御情報の読み取りまたは更新を待っています。

Lwlock:parallel_append

プロセスは、パラレル追加プランの実行中に次のサブプランを選択するのを待っています。

Lwlock:parallel_hash_join

プロセスは、パラレルハッシュプランの実行中に、メモリまたは更新カウンターのチャンクの割り当てまたは交換を待っています。

Lwlock:parallel_query_dsa

プロセスは、パラレルクエリの動的共有メモリ割り当てのロックを待っています。

Lwlock:ParallelAppend

プロセスは、パラレル追加プランの実行中に次のサブプランを選択するのを待っています。

Lwlock:ParallelHashJoin

プロセスは、パラレルハッシュ結合の計画実行中にワーカーの同期を待っています。

Lwlock:ParallelQueryDSA

プロセスは、パラレルクエリの動的共有メモリ割り当てを待っています。

Lwlock:PerSessionDSA

プロセスは、パラレルクエリの動的共有メモリ割り当てを待っています。

Lwlock:PerSessionRecordType

プロセスは、複合型に関するパラレルクエリの情報にアクセスするのを待っています。

Lwlock:PerSessionRecordTypmod

プロセスは、匿名レコードタイプを識別する型修飾子に関するパラレルクエリの情報へのアクセスを待っています。

Lwlock:PerXactPredicateList

プロセスは、パラレルクエリ中に現在の直列化可能トランザクションによって保持されている述語ロックのリストへのアクセスを待っています。

Lwlock:predicate_lock_manager

プロセスは、述語ロック情報の追加または検査を待っています。

Lwlock:PredicateLockManager

プロセスは、直列化可能なトランザクションで使用される述語ロック情報へのアクセスを待っています。

Lwlock:proc

プロセスは、高速パスロック情報の読み取りまたは更新を待っています。

Lwlock:ProcArray

プロセスは、共有プロセスごとのデータ構造にアクセスするのを待っています (通常は、スナップショットの取得やセッションのトランザクション ID のレポートなど)。

Lwlock:ProcArrayLock

プロセスは、スナップショットの取得を待っているか、トランザクションの最後にトランザクション ID をクリアしています。

Lwlock:RelationMapping

プロセスは、`pg_filenode.map` ファイル (特定のシステムカタログのファイルノード割り当てを追跡するために使用されます) の読み取りまたは更新を待っています。

Lwlock:RelationMappingLock

プロセスは、カタログからファイルノードへのマッピングを格納するために使用されるリレーションマップファイルの更新を待っています。

Lwlock:RelCacheInit

プロセスは、`pg_internal.init` ファイル (リレーションキャッシュ初期化ファイル) の読み取りまたは更新を待っています。

Lwlock:RelCacheInitLock

プロセスは、リレーションキャッシュ初期化ファイルの読み取りまたは書き込みを待っています。

Lwlock:replication_origin

プロセスは、レプリケーションの進行状況の読み取りまたは更新を待っています。

Lwlock:replication_slot_io

プロセスがレプリケーションスロットの I/O を待っています。

Lwlock:ReplicationOrigin

プロセスは、レプリケーションオリジンの作成、削除、または使用を待っています。

Lwlock:ReplicationOriginLock

プロセスは、レプリケーションオリジンの設定、削除、または使用を待っています。

Lwlock:ReplicationOriginState

プロセスは、1 つのレプリケーションオリジンの進行状況の読み取りまたは更新を待っています。

Lwlock:ReplicationSlotAllocation

プロセスがレプリケーションスロットの割り当てまたは解放を待っています。

Lwlock:ReplicationSlotAllocationLock

プロセスがレプリケーションスロットの割り当てまたは解放を待っています。

Lwlock:ReplicationSlotControl

プロセスは、レプリケーションスロットの状態の読み取りまたは更新を待っています。

Lwlock:ReplicationSlotControlLock

プロセスは、レプリケーションスロットの状態の読み取りまたは更新を待っています。

Lwlock:ReplicationSlotIO

プロセスがレプリケーションスロットの I/O を待っています。

Lwlock:SerialBuffer

プロセスは、直列化可能なトランザクションの競合のシンプルな最も長い時間使われていない (SLRU) バッファの I/O を待っています。

Lwlock:SerializableFinishedList

プロセスは、完成した直列化可能なトランザクションのリストへのアクセスを待っています。

Lwlock:SerializableFinishedListLock

プロセスは、完成した直列化可能なトランザクションのリストへのアクセスを待っています。

Lwlock:SerializablePredicateList

プロセスは、直列化可能なトランザクションによって保持されている述語ロックのリストへのアクセスを待っています。

Lwlock:SerializablePredicateLockListLock

プロセスは、直列化可能なトランザクションによって保持されているロックのリストに対する操作の実行を待っています。

Lwlock:SerializableXactHash

プロセスは、直列化可能なトランザクションに関する情報の読み取りまたは更新を待っています。

Lwlock:SerializableXactHashLock

プロセスは、直列化可能なトランザクションに関する情報を取得または保存するのを待っています。

Lwlock:SerialSLRU

プロセスは、直列化可能なトランザクションの競合のシンプルな最も長い時間使われていない (SLRU) キャッシュへのアクセスを待っています。

Lwlock:SharedTidBitmap

パラレルビットマップインデックススキャン中に、プロセスが共有タプル識別子 (TID) ビットマップへのアクセスを待っています。

Lwlock:SharedTupleStore

プロセスは、パラレルクエリ中に共有タプルストアへのアクセスを待っています。

Lwlock:ShmemIndex

プロセスは、共有メモリ内の領域を検索または割り当てるのを待っています。

Lwlock:ShmemIndexLock

プロセスは、共有メモリ内の領域を検索または割り当てるのを待っています。

Lwlock:SInvalRead

プロセスは、共有カタログの無効化キューからメッセージを取得するのを待っています。

Lwlock:SInvalReadLock

プロセスは、共有無効化キューからメッセージを取得または削除するのを待っています。

Lwlock:SInvalWrite

プロセスは、共有カタログの無効化キューにメッセージを追加するのを待っています。

Lwlock:SInvalWriteLock

プロセスが共有無効化キューにメッセージを追加するのを待っています。

Lwlock:subtrans

プロセスは、サブトランザクションバッファの I/O を待っています。

Lwlock:SubtransBuffer

プロセスは、サブトランザクションのシンプルな最も長い時間使われていない (SLRU) バッファの I/O を待っています。

LWLock:SubtransControlLock

プロセスは、サブトランザクション情報の読み取りまたは更新を待っています。

Lwlock:SubtransSLRU

プロセスは、サブトランザクションのシンプルな最も長い時間使われていない (SLRU) キャッシュへのアクセスを待っています。

Lwlock:SyncRep

プロセスは、同期レプリケーションの状態に関する情報の読み取りまたは更新を待っています。

Lwlock:SyncRepLock

プロセスは、同期レプリカに関する情報の読み取りまたは更新を待っています。

Lwlock:SyncScan

プロセスは、同期テーブルスキャンのスタート場所の選択を待っています。

Lwlock:SyncScanLock

プロセスは、同期スキャンのテーブル上でスキャンのスタート場所を取得するのを待っています。

Lwlock:TablespaceCreate

プロセスが表領域の作成または削除を待っています。

Lwlock:TablespaceCreateLock

プロセスが表領域の作成または削除を待っています。

Lwlock:tbm

プロセスは、ツリービットマップ (TBM) の共有イテレータロックを待っています。

Lwlock:TwoPhaseState

プロセスは、準備されたトランザクションの状態の読み取りまたは更新を待っています。

Lwlock:TwoPhaseStateLock

プロセスは、準備されたトランザクションの状態の読み取りまたは更新を待っています。

Lwlock:wal_insert

プロセスがメモリバッファへのログ先行書き込み (WAL) の挿入を待っています。

Lwlock:WALBufMapping

プロセスは WAL (ログ先行書き込み) バッファ内のページの置換を待っています。

Lwlock:WALBufMappingLock

プロセスは WAL (ログ先行書き込み) バッファ内のページの置換を待っています。

Lwlock:WALInsert

プロセスがメモリバッファへのログ先行書き込み (WAL) データの挿入を待っています。

Lwlock:WALWrite

プロセスは WAL (ログ先行書き込み) バッファがディスクに書き込まれるのを待っています。

Lwlock:WALWriteLockLwlock:WALWriteLock

プロセスは WAL (ログ先行書き込み) バッファがディスクに書き込まれるのを待っています。

Lwlock:WrapLimitsVacuum

プロセスは、トランザクション ID とマルチシャクト消費の制限の更新を待っています。

Lwlock:WrapLimitsVacuumLock

プロセスは、トランザクション ID とマルチシャクト消費の制限の更新を待っています。

Lwlock:XactBuffer

プロセスは、トランザクションステータスのシンプルな最も長い時間使われていない (SLRU) バッファの I/O を待っています。

Lwlock:XactSLRU

プロセスは、トランザクションステータスのシンプルな最も長い時間使われていない (SLRU) キャッシュへのアクセスを待っています。

Lwlock:XactTruncation

プロセスが pg_xact_status を実行するか、使用可能な最も古いトランザクション ID を更新するのを待っています。

Lwlock:XidGen

プロセスは、新しいトランザクション ID の割り当てを待っています。

Lwlock:XidGenLock

プロセスは、トランザクション ID の割り当てまたは割り当てを待っています。

Timeout:BaseBackupThrottle

アクティビティのスロットリング時に、ベースバックアップ中にプロセスが待っています。

Timeout:PgSleep

バックエンドプロセスが pg_sleep 関数を呼び出して、スリープタイムアウトの期限が切れるのを待っています。詳細については、「Timeout:PgSleep」を参照してください。

Timeout:RecoveryApplyDelay

遅延設定のため、プロセスはリカバリ中にログ先行書き込み (WAL) の適用を待っています。

Timeout:RecoveryRetrieveRetryInterval

どの出典 (pg_wal、アーカイブ、またはストリーミング) からもログ先行書き込み (WAL) データが使用できない場合、リカバリ中にプロセスが待っています。

Timeout:VacuumDelay

プロセスが、コストベースの真空遅延ポイントで待っています。

PostgreSQL の待機イベントの詳細なリストについては、PostgreSQL ドキュメントの「統計コレクター > 待機イベントテーブル」を参照してください。

Amazon Aurora PostgreSQL の更新

Amazon Aurora PostgreSQL エンジンのバージョンリリースと更新について以下に説明します。Aurora PostgreSQL エンジンをアップグレードする方法についても説明します。Aurora リリースの全般的な詳細については、「Amazon Aurora バージョン」を参照してください。

> **Tip**
>
> ブルー/グリーンデプロイを使用することで、DB クラスターのアップグレードに必要なダウンタイムを最小限に抑えることができます。詳細については、「データベースの更新にブルー/グリーンデプロイを使用する」を参照してください。

トピック

Amazon Aurora PostgreSQL のバージョンの識別

Amazon Aurora には、Aurora としては一般的で、すべての Aurora DB クラスターで使用できる一定の機能が含まれています。その他として Aurora には、Aurora がサポートする特定のデータベースエンジン固有の機能が含まれます。これらの機能は、各データベースエンジン (Aurora PostgreSQL など) を使用する Aurora DB クラスターでのみ使用できます。

Aurora データベースのリリースには、通常、データベースエンジンバージョン番号と Aurora バージョン番号の 2 つのバージョン番号があります。Aurora PostgreSQL リリースに Aurora バージョン番号がある場合、Amazon Aurora PostgreSQL リリースとエンジンのバージョン の一覧の中でエンジンバージョンの後に含まれています。

Aurora バージョン番号

Aurora バージョン番号では、*major.minor.patch* という命名法を使用しています。Aurora パッチバージョンには、リリース後にマイナーバージョンに追加された重要なバグ修正が含まれています。Amazon Aurora のメジャー、マイナー、パッチリリースの詳細については、「Amazon Aurora

メジャーバージョン」、「Amazon Aurora マイナーバージョン」、および「Amazon Aurora パッチバージョン」を参照してください。

次の SQL クエリを使用して、Aurora PostgreSQL DB インスタンスの Aurora バージョン番号を確認できます。

```
postgres=> SELECT aurora_version();
```

PostgreSQL バージョン 13.3、12.8、11.13、10.18、およびそれ以降のすべてのリリースからは、Aurora バージョン番号は PostgreSQL エンジンバージョンとより密接に整合するようになりました。例えば、Aurora PostgreSQL 13.3 DB クラスターをクエリすると、次の内容が返されます。

```
aurora_version
----------------
 13.3.1
(1 row)
```

Aurora PostgreSQL 10.14 DB クラスターなどの以前のリリースでは、次の例のようなバージョン番号が返されます。

```
aurora_version
----------------
 2.7.3
(1 row)
```

PostgreSQL エンジンのバージョン番号

PostgreSQL 10 以降、PostgreSQL データベースエンジンのバージョンでは *major*.*minor* という番号付け方法がすべてのリリースに使用されています。PostgreSQL 10.18、PostgreSQL 12.7、PostgreSQL 13.3 などの例があります。

PostgreSQL 10 より前のリリースでは、*major*.*major*.*minor* という番号付け方法が使用されていました。最初の 2 桁の数字がメジャーバージョン番号で、3 桁目の数字がマイナーバージョンを表します。例えば、PostgreSQL 9.6 がメジャーバージョンで、3 桁目の数字の 9.6.21 や 9.6.22 がマイナーバージョンを表します。

> **Note**
>
> PostgreSQL エンジンバージョン 9.6 はサポートされなくなりました。アップグレードするには、Amazon Aurora PostgreSQL DB クラスターのアップグレード を参照してください。

バージョンポリシーとリリースタイムラインについては、「Amazon Aurora メジャーバージョンが利用可能な期間」を参照してください。

PostgreSQL データベースエンジンのバージョン番号は、次の SQL クエリで確認できます。

```
postgres=> SELECT version();
```

Aurora PostgreSQL 13.3 DB クラスターの場合、結果は次のようになります。

```
version
-----------------------------------------------------------------------------------------------------------
 PostgreSQL 13.3 on x86_64-pc-linux-gnu, compiled by x86_64-pc-linux-gnu-gcc (GCC)
 7.4.0, 64-bit
(1 row)
```

Amazon Aurora PostgreSQL リリースとエンジンのバージョン

Amazon Aurora PostgreSQL 互換エディションのリリースは定期的に更新されています。更新はシステムメンテナンスの時間中に Aurora PostgreSQL DB クラスターに適用されます。更新が適用されるタイミングは、更新のタイプ、AWS リージョン、DB クラスターのメンテナンス期間の設定によって異なります。リストされているリリースの多くには、PostgreSQL のバージョン番号と Amazon Aurora バージョン番号の両方が含まれています。PostgreSQL バージョン 13.3、12.8、11.13、10.18 およびそれ以降のすべてのバージョンからは、Aurora のバージョン番号は使用されません。Aurora PostgreSQL データベースのバージョン番号を確認する方法は、「Amazon Aurora PostgreSQL のバージョンの識別」を参照してください。

エクステンションおよびモジュールの詳細については、「Amazon Aurora PostgreSQL のエクステンションバージョン」を参照してください。

> **Note**
>
> Amazon Aurora のバージョンポリシー、リリースタイムラインの詳細については、「Amazon Aurora メジャーバージョンが利用可能な期間」を参照してください。

Amazon Aurora のサポートの詳細については、「Amazon RDS のよくある質問」を参照してください。

AWS リージョン で使用できる PostgreSQL エンジンのバージョンを確認するには、AWS CLI コマンドの describe-db-engine-versions を使用します。例:

```
aws rds describe-db-engine-versions --engine aurora-postgresql --query '*[].
[EngineVersion]' --output text --region aws-region
```

AWS リージョン のリストについては、「Aurora PostgreSQL が利用可能なリージョン」を参照してください。

Aurora PostgreSQL で使用可能な PostgreSQL のバージョンの詳細については、「Aurora PostgreSQL のリリースノート」を参照してください。

Amazon Aurora PostgreSQL のエクステンションバージョン

Aurora PostgreSQL DB クラスターで使用するさまざまな PostgreSQL 拡張機能をインストールして設定できます。例えば、PostgreSQL pg_partman エクステンションを使用すると、テーブルパーティションの作成とメンテナンスを自動化できます。この機能と、Aurora PostgreSQL で利用できるその他の拡張機能の詳細については、「エクステンションと外部データラッパーの使用」を参照してください。

Aurora PostgreSQL でサポートされている PostgreSQL 拡張機能の詳細については、Aurora PostgreSQL のリリースノートの「Amazon Aurora PostgreSQL の拡張機能バージョン」を参照してください。

Amazon Aurora PostgreSQL DB クラスターのアップグレード

Amazon Aurora では、広範なテストの後にのみ AWS リージョン で PostgreSQL データベースエンジンの新しいバージョンが利用可能となります。Aurora PostgreSQL DB クラスターは、リージョンで利用可能になった時点で新しいバージョンにアップグレードできます。

DB クラスターが現在実行されている Aurora PostgreSQL のバージョンに応じて、新しいリリースへのアップグレードは、マイナーアップグレードまたはメジャーアップグレードのいずれかになります。例えば、Aurora PostgreSQL 11.15 DB クラスターを Aurora PostgreSQL 13.6 にアップグレードした場合は、メジャーバージョンアップグレードです。Aurora PostgreSQL 13.3 DB クラスターを Aurora PostgreSQL 13.7 にアップグレードした場合は、マイナーバージョンアップグレードです。次のトピックでは、両方のタイプのアップグレードを実行する方法について説明します。

目次

- AWS リージョン で使用可能なバージョンのリストを取得します。
- メジャーバージョンのアップグレードを実施する方法
  - 本番稼働用の DB クラスターの新しいメジャーバージョンへのアップグレードをテストする
  - Aurora PostgreSQL エンジンを新しいメジャーバージョンにアップグレードする
    - グローバルデータベースのメジャーアップグレード
- マイナーバージョンアップグレードを実行する前に
- マイナーバージョンのアップグレードとパッチの適用方法
  - マイナーリリースのアップグレードとダウンタイムなしのパッチ適用プロセス
  - Aurora PostgreSQL エンジンを新しいマイナーバージョンにアップグレードする
- PostgreSQL 拡張機能のアップグレード
- 代替のブルー/グリーンのアップグレードテクニック

Aurora PostgreSQL のアップグレードプロセスの概要

メジャーバージョンとマイナーバージョンのアップグレードの違いは、次のとおりです。

マイナーバージョンのアップグレードとパッチ

マイナーバージョンアップグレードおよびパッチには、既存のアプリケーションとの下位互換性がある変更のみが含まれます。マイナーバージョンのアップグレードとパッチは、Aurora PostgreSQL がテストして承認した後にのみ利用可能になります。

マイナーバージョンのアップグレードは、Aurora によって自動的に適用できます。新しい Aurora PostgreSQL DB クラスターを作成すると、[マイナーバージョンアップグレードの有効化] オプションが事前に選択されています。このオプションをオフにしない限り、スケジュールされたメンテナンス期間中には、マイナーバージョンのアップグレードが自動的に適用されます。自動マイナーバージョンアップグレード (AMVU) オプションと Aurora DB クラスターを使用できるように変更する方法については、「Aurora DB クラスターのマイナーバージョン自動アップグレード」を参照してください。

自動マイナーバージョンアップグレードオプションが Aurora PostgreSQL DB クラスターに設定されていない場合、Aurora PostgreSQL は新しいマイナーバージョンに自動的にアップグレードされません。代わりに、AWS リージョン で新しいマイナーバージョンがリリースされ、Aurora PostgreSQL DB クラスターが古いマイナーバージョンを実行している場合、Aurora はアップグレードを要求します。そのためには、クラスターのメンテナンスタスクにレコメンデーションを追加します。

パッチはアップグレードとはみなされず、自動的には適用されません。Aurora PostgreSQL では、Aurora PostgreSQL DB クラスターのメンテナンスタスクに推奨事項を追加して、パッチを適用するように求められます。詳細については、「マイナーバージョンのアップグレードとパッチの適用方法」を参照してください。

> **Note**
>
> セキュリティやその他の重要な問題を解決するパッチも、メンテナンスタスクとして追加されます。ただし、これらのパッチは必須です。保留中のメンテナンスタスクでセキュリティパッチが使用可能になったら、Aurora PostgreSQL DB クラスターにセキュリティパッチを適用してください。

アップグレードプロセスでは、クラスター内の各インスタンスが新しいバージョンにアップグレードされる際に、短時間停止する可能性があります。ただし、Aurora PostgreSQL バージョン 14.3.3、13.7.3、12.11.3、11.16.3、10.21.3、およびこれらのマイナーバージョンやより新しいメジャーバージョンのその他のリリース以降では、アップグレードプロセスではゼロダウンタイムパッチ適用 (ZDP) 機能が使用されます。この機能は停止を最小限に抑え、ほとんどの場合は完全に排除します。詳細については、「マイナーリリースのアップグレードとダウンタイムなしのパッチ適用プロセス」を参照してください。

> **Note**
>
> ZDP は、次の場合サポートされていません。
>
> - Aurora PostgreSQL DB クラスターが Aurora Serverless v1 に設定されている場合。
> - Aurora PostgreSQL DB クラスターがセカンダリ AWS リージョン の Aurora グローバルデータベースとして設定されている場合。
> - Aurora グローバルデータベースのリーダーインスタンスのアップグレード中。
> - OS パッチおよび OS アップグレード中。
>
> ZDP は Aurora Serverless v2 として設定されている Aurora PostgreSQL DB クラスターでサポートされています。

メジャーバージョンのアップグレード

マイナーバージョンのアップグレードやパッチとは異なり、Aurora PostgreSQL には自動メジャーバージョンアップグレードオプションはありません。新しいメジャー PostgreSQL バー

ジョンのアップグレードには、既存のアプリケーションとの下位互換性のないデータベースの変更が含まれる場合があります。新しい機能により、既存のアプリケーションが適切に動作しなくなることがあります。

Aurora PostgreSQL DB クラスター内の DB インスタンスをアップグレードする前に、問題を予防するため、「本番稼働用の DB クラスターの新しいメジャーバージョンへのアップグレードをテストする」で説明している手順に従うことを強くお勧めします。まず、以下の手順に従って、アプリケーションを新しいバージョンで実行できることを確認します。その後、Aurora PostgreSQL DB クラスターを新しいバージョンに手動でアップグレードできます。

アップグレードプロセスでは、クラスター内のすべてのインスタンスが新しいバージョンにアップグレードされる際に、短時間停止する可能性があります。事前計画プロセスにも時間がかかります。アップグレードタスクは、必ずクラスターのメンテナンス期間中、または運用が最小限のタイミングで実行することをお勧めします。詳細については、「メジャーバージョンのアップグレードを実施する方法」を参照してください。

> **Note**
>
> マイナーバージョンのアップグレードとメジャーバージョンのアップグレードの両方で、短期間の停止を伴う可能性があります。そのため、メンテナンス期間中、または使用率の低い時間帯にアップグレードを実行またはスケジュールすることを強くお勧めします。

Aurora PostgreSQL DB クラスターでは、オペレーティングシステムの更新が必要になる場合があります。これらのアップデートには glibc ライブラリの新しいバージョンが含まれることがあります。このような更新の際は、「Aurora PostgreSQL でサポートされる照合」で説明されているガイドラインに従うことをお勧めします。

AWS リージョン で使用可能なバージョンのリストを取得します。

次のように、describe-db-engine-versions AWS CLI コマンドを使用して AWS リージョン をクエリすることにより、Aurora PostgreSQL DB クラスターのアップグレードターゲットとして利用可能なすべてのエンジンバージョンのリストを取得できます。

Linux、macOS、Unix の場合:

```
aws rds describe-db-engine-versions \
  --engine aurora-postgresql \
```

```
  --engine-version version-number \
  --query 'DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}' \
  --output text
```

Windows の場合:

```
aws rds describe-db-engine-versions ^
  --engine aurora-postgresql ^
  --engine-version version-number ^
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" ^
  --output text
```

例えば、Aurora PostgreSQL バージョン 12.10 DB クラスターの有効なアップグレードターゲットを特定するには、次の AWS CLI のコマンドを実行します。

Linux、macOS、Unix の場合:

```
aws rds describe-db-engine-versions \
  --engine aurora-postgresql \
  --engine-version 12.10 \
  --query 'DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}' \
  --output text
```

Windows の場合:

```
aws rds describe-db-engine-versions ^
  --engine aurora-postgresql ^
  --engine-version 12.10 ^
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" ^
  --output text
```

この表で、Aurora PostgreSQL DB のさまざまなバージョンで使用できるメジャーバージョンとマイナーバージョンの両方のアップグレードターゲットを見つけることができます。

現在のソースバージョン	アップグレードターゲット
16.1	16
15.6	16
15.5	16 16 15
15.4	16 16 15 15
15.3	16 16 15 15 15
15.2	16 16 15 15 15 15
14.11	16 15
14.10	16 16 15 15
14.9	16 16 15 15 15 14 14
14.8	16 16 15 15 15 15 15 14 14 14
14.7	16 16 15 15 15 15 15 14 14 14 14
14.6	16 16 15 15 15 15 15 14 14 14 14 14
14.5	16 16 15 15 15 15 15 14 14 14 14 14 14
14.4	16 16 15 15 15 15 15 14 14 14 14 14 14 14
14.3	16 16 15 15 15 15 15 14 14 14 14 14 14 14 14

現在のソースバージョン	アップグレードターゲット
13.14	16 15 14
13.13	16 16 15 15 14 14
13.12	16 16 15 15 15 14 14 14
13.11	16 16 15 15 15 15 14 14 14 14
13.10	16 16 15 15 15 15 15 14 14 14 14 14 13 13 13 13
13.9	16 16 15 15 15 15 15 14 14 14 14 14 14 13 13 13
13.8	16 16 15 15 15 15 15 14 14 14 14 14 14 14 13 13 13 13 13 13
13.7	16 16 15 15 15 15 15 14 14 14 14 14 14 14 14 14 13 13 13 13 13 13 13
12.18	16 15 14 13
12.17	16 16 15 15 14 14 13
12.16	16 16 15 15 15 14 14 14 13 13 13
12.15	16 16 15 15 15 15 14 14 14 14 13 13 13 13
12.14	16 16 15 15 15 15 15 15 14 14 14 14 14 13 13 13 13 13 12
12.13	16 16 15 15 15 15 15 14 14 14 14 14 14 13 13 13 13 13 13 12 12 12 12
12.12	16 16 15 15 15 15 15 14 14 14 14 14 14 14 13 13 13 13 13 13 12 12 12 13 12 12 12
12.11	16 16 15 15 15 15 15 14 14 14 14 14 14 14 14 13 13 13 13 13 13 13 13 12 12 12 12

現在のソースバージョン	アップグレードターゲット
12.9	16 16 15 15 15 15 15 14 14 14 14 14 13 13 13 13 13 13 13 12 12 12 12 12 12 12
11.21	16 16 15 15 15 14 14 14 13 13 13 12 12
11.9	16 16 15 15 15 15 15 14 14 14 14 14 14 13 13 13 13 13 13 12 12 12 12 12 12 12 12 11.21

検討しているバージョンについて、クラスターの DB インスタンスクラスの可用性を常に確認してください。例えば、db.r4 は、Aurora PostgreSQL 13 ではサポートされていません。Aurora PostgreSQL DB クラスターが現在 db.r4 インスタンスクラスを使用している場合は、アップグレードする前に db.r5 に移動する必要があります。どれが Graviton2 ベースでどれがインテルベースであるかなど、DB インスタンスクラスの詳細については、「Aurora DB インスタンスクラス」を参照してください。

メジャーバージョンのアップグレードを実施する方法

メジャーバージョンのアップグレードには、以前のバージョンのデータベースと下位互換性のないデータベースの変更が含まれる可能性があります。新しいバージョンの新機能により、既存のアプリケーションが適切に動作しなくなることがあります。問題を回避するため、Amazon Aurora では、メジャーバージョンアップグレードは自動的に適用されません。むしろ、次の手順を実行して、メジャーバージョンのアップグレードを慎重に計画することをお勧めします。

1. 使用可能なターゲットのリストから必要なメジャーバージョンを、テーブル内のバージョンにリストされているターゲットから選択します。AWS CLI を使うことにより現在のバージョンの AWS リージョン で利用可能なバージョンの正確なリストを入手できます。詳細については、「AWS リージョン で使用可能なバージョンのリストを取得します。」を参照してください。

2. 新しいバージョンのトライアルデプロイで、アプリケーションが正常に動作することを確認します。完全なプロセスの詳細については、「本番稼働用の DB クラスターの新しいメジャーバージョンへのアップグレードをテストする」を参照してください。
3. トライアルデプロイでアプリケーションが正常に動作することを確認したら、クラスターをアップグレードできます。詳細については、「Aurora PostgreSQL エンジンを新しいメジャーバージョンにアップグレードする」を参照してください。

> **Note**
>
> 13.6 以降の Babelfish for Aurora PostgreSQL 13 ベースのバージョンから 14.6 以降の Aurora PostgreSQL 14 ベースのバージョンへのメジャーバージョンアップグレードを実行できます。Babelfish for Aurora PostgreSQL 13.4 と 13.5 は、メジャーバージョンアップグレードをサポートしていません。

次のように、describe-db-engine-versions AWS CLI コマンドを使用して AWS リージョン をクエリすることにより、Aurora PostgreSQL DB クラスターのメジャーバージョンアップグレードターゲットとして利用可能なエンジンバージョンのリストを取得できます。

Linux、macOS、Unix の場合:

```
aws rds describe-db-engine-versions \
  --engine aurora-postgresql \
  --engine-version version-number \
  --query 'DBEngineVersions[].ValidUpgradeTarget[?IsMajorVersionUpgrade == `true`].
{EngineVersion:EngineVersion}' \
  --output text
```

Windows の場合:

```
aws rds describe-db-engine-versions ^
  --engine aurora-postgresql ^
  --engine-version version-number ^
  --query "DBEngineVersions[].ValidUpgradeTarget[?IsMajorVersionUpgrade == `true`].
{EngineVersion:EngineVersion}" ^
  --output text
```

場合によっては、アップグレードするバージョンが現行のバージョンのターゲットではない場合があります。そのような場合は、versions table の情報を使用して、クラスターがターゲットの行に選択したターゲットを持つバージョンになるまで、マイナーバージョンのアップグレードを実行します。

本番稼働用の DB クラスターの新しいメジャーバージョンへのアップグレードをテストする

各新しいメジャーバージョンには、パフォーマンスを向上させるために設計されたクエリオプティマイザの機能強化が含まれています。ただし、ワークロードには、新しいバージョンでプランの実行を低下させるクエリが含まれる場合があります。そのため、本番環境でアップグレードする前に、パフォーマンスをテストして確認することをお勧めします。メジャーバージョンのアップグレード後の計画の安定性の確保 で説明されているように、クエリプラン管理 (QPM) 拡張機能を使用して、バージョン間でクエリプランの安定性を管理できます。

本番稼働用の Aurora PostgreSQL DB クラスターを新しいメジャーバージョンにアップグレードする前に、アップグレードをテストして、アプリケーションが正常に動作することを確認することを強く推奨します。

1. バージョン互換のパラメータグループを準備します。

   カスタム DB インスタンスまたは DB クラスターパラメータグループを使用している場合は、2 つのオプションから選択できます。

   a. 新しい DB エンジンバージョンのデフォルト DB インスタンス、DB クラスターパラメータグループ、またはその両方を指定します。

   b. 新しい DB エンジンバージョンの独自のカスタムパラメータグループを作成します。

   アップグレードリクエストの一部として DB インスタンスまたは DB クラスターの新しいパラメータグループを関連付ける場合は、パラメータを適用するために必ず、アップグレードの完了後にデータベースを再起動してください。パラメータグループの変更を適用するために DB インスタンスを再起動する必要がある場合、インスタンスのパラメータグループのステータスは `pending-reboot` と表示されます。インスタンスのパラメータグループのステータスは、コンソールまたは describe-db-instances や describe-db-clusters などの CLI コマンドを使用して確認できます。

2. サポートされていない使用の確認

   - アップグレードを試みる前に、すべての準備済みのトランザクションをコミットまたはロールバックします。次のクエリを使用して、開いている準備済みのトランザクションがインスタンスにないことを確認します。

     ```
     SELECT count(*) FROM pg_catalog.pg_prepared_xacts;
     ```

- アップグレードを試みる前に、使用されているすべての reg* データ型を削除します。`regtype` と `regclass` を除き、reg* データ型をアップグレードすることはできません。このデータ型は、pg_upgrade ユーティリティ (Amazon Aurora でアップグレードに使用される) で保持することはできません。このユーティリティの詳細については、PostgreSQL のドキュメントの「pg_upgrade」を参照してください。

  サポートされていない reg* データ型が使用されていないことを確認するには、データベースごとに次のクエリを使用します。

```
SELECT count(*) FROM pg_catalog.pg_class c, pg_catalog.pg_namespace n,
 pg_catalog.pg_attribute a
  WHERE c.oid = a.attrelid
      AND NOT a.attisdropped
      AND a.atttypid IN ('pg_catalog.regproc'::pg_catalog.regtype,
                         'pg_catalog.regprocedure'::pg_catalog.regtype,
                         'pg_catalog.regoper'::pg_catalog.regtype,
                         'pg_catalog.regoperator'::pg_catalog.regtype,
                         'pg_catalog.regconfig'::pg_catalog.regtype,
                         'pg_catalog.regdictionary'::pg_catalog.regtype)
      AND c.relnamespace = n.oid
      AND n.nspname NOT IN ('pg_catalog', 'information_schema');
```

- pgRouting 拡張機能がインストールされている Aurora PostgreSQL バージョン 10.18 以上の DB クラスターをアップグレードする場合には、バージョン 12.4 以上にアップグレードする前に拡張機能を削除してください。

  拡張子 `pg_repack` バージョン 1.4.3 がインストールされている Aurora PostgreSQL 10.x バージョンをアップグレードする場合には、より高いバージョンにアップグレードする前に拡張子を削除してください。

3. template1 と template0 のデータベースを確認してください。

   アップグレードを成功させるには、テンプレート 1 とテンプレート 0 のデータベースが存在し、テンプレートとしてリストされている必要があります。これをチェックするには、次のコマンドを使用します。

```
SELECT datname, datistemplate FROM pg_database;

datname    | datistemplate
-----------+---------------
template0  | t
```

```
rdsadmin   | f
template1  | t
postgres   | f
```

コマンド出力では、template1 データベースと template0 データベースの datistemplate 値は、t でなければなりません。

4. 論理的なレプリケーションスロットを削除します。

   Aurora PostgreSQL DB クラスターがいずれかの論理レプリケーションスロットを使用している場合は、アップグレードプロセスを続行できません。論理レプリケーションスロットは通常、AWS DMS を使用したデータの移行、またはデータベースからデータレイク、BI ツール、およびその他のターゲットへのテーブルのレプリケートなどの短期のデータの移行タスクに使用されます。アップグレードする前に、既存の論理レプリケーションスロットの目的を確認し、削除しても問題ないことを確認してください。次のクエリを使用して論理レプリケーションスロットを確認できます。

   ```
   SELECT * FROM pg_replication_slots;
   ```

   論理レプリケーションスロットがまだ使用されている場合は、それらを削除しないでください。また、その場合、アップグレードを続行することはできません。ただし、論理レプリケーションスロットが不要な場合は、次の SQL を使用して削除できます。

   ```
   SELECT pg_drop_replication_slot(slot_name);
   ```

   pglogical 拡張機能を使用するロジカルレプリケーションシナリオでも、パブリッシャーノードでメジャーバージョンアップグレードを正常に行うには、パブリッシャーノードからスロットを削除する必要があります。ただし、アップグレード後にサブスクライバーノードからレプリケーションプロセスを再開できます。詳細については、「メジャーアップグレード後の論理レプリケーションの再確立」を参照してください。

5. バックアップを実行します。

   アップグレードプロセスでは、アップグレード中に DB クラスターのスナップショットが作成されます。アップグレードプロセスの前に手動でもバックアップを行う場合の詳細については、「DB クラススナップショットの作成」を参照してください。

6. メジャーバージョンアップグレードを実行する前に、特定の拡張機能を、利用可能な最新バージョンにアップグレードします。更新する拡張機能は以下のとおりです。

   - pgRouting

- postgis_raster
- postgis_tiger_geocoder
- postgis_topology
- address_standardizer
- address_standardizer_data_us

現在インストールされている拡張機能ごとに以下のコマンドを実行します。

```
ALTER EXTENSION PostgreSQL-extension UPDATE TO 'new-version';
```

詳細については、「PostgreSQL 拡張機能のアップグレード」を参照してください。PostGIS のアップグレードの詳細については、「ステップ 6: PostGIS 拡張機能を更新する」を参照してください。

7. バージョン 11.x にアップグレードする場合は、メジャーバージョンアップグレードを実行する前に、サポートされていない拡張機能を削除してください。削除する拡張機能は以下のとおりです。
   - chkpass
   - tsearch2
8. ターゲットバージョンに応じて、unknown データ型を削除します。

   PostgreSQL バージョン 10 では、unknown データ型をサポートしていません。バージョン 9.6 のデータベースで unknown データ型を使用している場合、バージョン 10 にアップグレードすると次のようなエラーメッセージが表示されます。

   ```
   Database instance is in a state that cannot be upgraded: PreUpgrade checks failed:
   The instance could not be upgraded because the 'unknown' data type is used in user
    tables.
   Please remove all usages of the 'unknown' data type and try again."
   ```

   データベース内の unknown データ型を検索して、対象の列を削除したり、サポートされているデータ型に変更したりするには、各データベースに次の SQL コードを使用します。

   ```
   SELECT n.nspname, c.relname, a.attname
       FROM pg_catalog.pg_class c,
       pg_catalog.pg_namespace n,
       pg_catalog.pg_attribute a
       WHERE c.oid = a.attrelid AND NOT a.attisdropped AND
   ```

```
    a.atttypid = 'pg_catalog.unknown'::pg_catalog.regtype AND
    c.relkind IN ('r','m','c') AND
    c.relnamespace = n.oid AND
    n.nspname !~ '^pg_temp_' AND
    n.nspname !~ '^pg_toast_temp_' AND n.nspname NOT IN ('pg_catalog',
'information_schema');
```

9. リハーサルのアップグレードを実行します。

   プロダクションデータベースのアップグレードを試みる前に、プロダクションデータベースの複製でメジャーバージョンアップグレードをテストすることを強くお勧めします。複製されたテストインスタンスの実行計画を監視して、実行計画のリグレッションが発生していないかどうかを確認し、そのパフォーマンスを評価できます。テスト用の複製のインスタンスを作成するには、最新のスナップショットからデータベースを復元するか、データベースのクローンを作成します。詳細については、「スナップショットからの復元」または「Amazon Aurora DB クラスターのボリュームのクローン作成」を参照してください。

   詳細については、「Aurora PostgreSQL エンジンを新しいメジャーバージョンにアップグレードする」を参照してください。

10. プロダクションインスタンスをアップグレードします。

    リハーサルのメジャーバージョンアップグレードが成功したら、安心してプロダクションデータベースをアップグレードできます。詳細については、「Aurora PostgreSQL エンジンを新しいメジャーバージョンにアップグレードする」を参照してください。

> **Note**
>
> アップグレードプロセス中は、クラスターのバックアップ保持期間に 0 より大きい値が設定されている場合、Aurora PostgreSQL は、DB クラスターのスナップショットを作成します。このプロセス中、クラスターのポイントインタイム復元を実行することはできません。アップグレードのスタート前およびインスタンスの自動スナップショットの完了後に、後からポイントインタイムの復元を実行できます。ただし、以前のマイナーバージョンのポイントインタイム復元を実行することはできません。

進行中のアップグレードについては、Amazon RDS を使用して、pg_upgrade ユーティリティで生成される 2 つのログを表示することができます。表示できるのは `pg_upgrade_internal.log` および `pg_upgrade_server.log` です。これらのログのファイ

ル名には、Amazon Aurora によりタイムスタンプが追加されます。これらのログも、他のログと同様、表示できます。詳細については、「Amazon Aurora ログファイルのモニタリング」を参照してください。

11 PostgreSQL の拡張機能をアップグレードします。PostgreSQL のアップグレードプロセスでは、PostgreSQL の拡張機能はアップグレードされません。詳細については、「PostgreSQL 拡張機能のアップグレード」を参照してください。

メジャーバージョンアップグレードが完了したら、以下のことをお勧めします。

- ANALYZE 操作を実行して pg_statistic テーブルを更新します。これは、すべての PostgreSQL DB インスタンスのすべてのデータベースに対して行う必要があります。Optimizer の統計情報はメジャーバージョンのアップグレード中には転送されないため、パフォーマンスの問題を回避するためにすべての統計情報を再生成する必要があります。次のようにパラメータを指定せずにコマンドを実行して、現在のデータベース内のすべての標準テーブルの統計情報を生成します。

```
ANALYZE VERBOSE;
```

  VERBOSE フラグはオプションですが、使用することで進行状況を表示できます。詳細については、「PostgreSQL ドキュメント」の「ANALYZE」を参照してください。

> **Note**
>
> パフォーマンスの問題を回避するため、アップグレード後にシステムで ANALYZE を実行してください。

- PostgreSQL バージョン 10 にアップグレードした場合は、使用しているハッシュインデックスで REINDEX を実行してください。ハッシュインデックスはバージョン 10 で変更されたため、再構築する必要があります。無効なハッシュインデックスを見つけるには、ハッシュインデックスを含む各データベースに対して次の SQL を実行します。

```
SELECT idx.indrelid::regclass AS table_name,
   idx.indexrelid::regclass AS index_name
FROM pg_catalog.pg_index idx
   JOIN pg_catalog.pg_class cls ON cls.oid = idx.indexrelid
   JOIN pg_catalog.pg_am am ON am.oid = cls.relam
WHERE am.amname = 'hash'
AND NOT idx.indisvalid;
```

- アップグレードしたデータベースで同様のワークロードでアプリケーションをテストして、すべてが期待どおりに機能することを確認することをお勧めします。アップグレードが確認されたら、このテストインスタンスを削除できます。

Aurora PostgreSQL エンジンを新しいメジャーバージョンにアップグレードする

新しいメジャーバージョンへのアップグレードプロセスを開始するとき、Aurora PostgreSQL はクラスターに変更を加える前に Aurora DB クラスターのスナップショットを取得します。このスナップショットは、メジャーバージョンのアップグレード用にのみ作成され、マイナーバージョンのアップグレードでは作成されません。アップグレードプロセスが完了すると、このスナップショットは、RDS コンソールのスナップショットにリストされている手動スナップショットの中に表示されています。次の例のように、スナップショット名には、プレフィックス、Aurora PostgreSQL DB クラスターの名前、ソースバージョン、ターゲットバージョン、日付とタイムスタンプとして `preupgrade` が含まれます。

```
preupgrade-docs-lab-apg-global-db-12-8-to-13-6-2022-05-19-00-19
```

アップグレードの完了後、Aurora が作成して手動スナップショットリストに保存したスナップショットを使用して、必要に応じて DB クラスターを以前のバージョンに復元できます。

> **Tip**
>
> 一般に、スナップショットは Aurora DB クラスターをさまざまな時点に復元するためのさまざまな方法を提供します。詳細については、「DB クラスターのスナップショットからの復元」および「DB クラスターを指定の時点の状態に復元する」を参照してください。ただし、Aurora PostgreSQL は、以前のマイナーバージョンに復元するためのスナップショットを使用をサポートしていません。

メジャーバージョンのアップグレードプロセス中には、Aurora によってボリュームが割り当てられ、ソース Aurora PostgreSQL DB クラスターのクローンが作成されます。アップグレードが何らかの理由で失敗した場合、Aurora PostgreSQL はクローンを使用してアップグレードをロールバックします。ソースのボリュームのクローンが 15 個より多く割り当てられた後、後続のクローンはフルコピーになり、時間がかかります。これにより、アップグレードプロセスにかかる時間が延びる場合があります。Aurora PostgreSQL でアップグレードがロールバックされる場合は、次の点に注意してください。

- アップグレード中に割り当てられたクローンボリューム、およびその元となったボリュームの両方について、請求情報とメトリクスが表示される可能性があります。クラスターのバックアップ保持期間がアップグレードの時間を超えた時点で、Aurora PostgreSQL は余分なボリュームをクリーンアップします。
- そのクラスターの次のクロスリージョンスナップショットコピーは、増分コピーではなく、フルコピーになります。

クラスターを構成する DB インスタンスを安全にアップグレードするために、Aurora PostgreSQL では pg_upgrade ユーティリティを使用します。ライターのアップグレードが完了すると、各リーダーインスタンスは新しいメジャーバージョンにアップグレードされている間、短時間停止します。この PostgreSQL ユーティリティの詳細については、PostgreSQL のドキュメントの「pg_upgrade」を参照してください。

AWS Management Console、AWS CLI、または RDS API を使用することにより、Aurora PostgreSQL DB クラスターを新しいバージョンにアップグレードできます。

コンソール

DB クラスターのエンジンバージョンを変更するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[データベース] を選択して、アップグレードする DB クラスターを選択します。
3. [Modify] を選択します。[DB クラスターの変更] ページが表示されます。
4. [Engine version] (エンジンバージョン) で、新しいバージョンを選択します。
5. [続行] を選択して、変更の概要を確認します。
6. 変更をすぐに反映させるには、[Apply immediately] を選択します。このオプションを選択すると、停止状態になる場合があります。詳細については、「Amazon Aurora DB クラスターの変更」を参照してください。
7. 確認ページで、変更内容を確認します。正しい場合は、[クラスターの変更] を選択して変更を保存します。

   または、[戻る] を選択して変更を編集するか、[キャンセル] を選択して変更をキャンセルします。

AWS CLI

DB クラスターのエンジンバージョンをアップグレードするには、CLI の modify-db-cluster AWS CLI コマンドを使用します。以下のパラメータを指定します。

- `--db-cluster-identifier` - DB クラスターの名前。
- `--engine-version` - アップグレード先のデータベースエンジンのバージョン番号です。有効なエンジンバージョンの詳細については、AWS CLI の describe-db-engine-versions コマンドを参照してください。
- `--allow-major-version-upgrade` - `--engine-version` パラメータが DB クラスターの現在のメジャーバージョンとは異なるメジャーバージョンである場合に必須のフラグです。
- `--no-apply-immediately` - 次のメンテナンス時間中に変更を適用します。今すぐ変更を適用するには、`--apply-immediately` を使用します。

Example

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --engine-version new_version \
    --allow-major-version-upgrade \
    --no-apply-immediately
```

Windows の場合:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --engine-version new_version ^
    --allow-major-version-upgrade ^
    --no-apply-immediately
```

RDS API

DB クラスターのエンジンのバージョンをアップグレードするには、ModifyDBCluster オペレーションを使用します。以下のパラメータを指定します。

- `DBClusterIdentifier` - DB クラスターの名前、例えば *mydbcluster* です。

- `EngineVersion` - アップグレード先のデータベースエンジンのバージョン番号です。有効なエンジンバージョンについては、DescribeDBEngineVersions オペレーションを使用します。
- `AllowMajorVersionUpgrade` - `EngineVersion` パラメータが DB クラスターの現在のメジャーバージョンとは異なるメジャーバージョンである場合に必須のフラグです。
- `ApplyImmediately` - 変更をすぐに適用するか、次のメンテナンスウィンドウ中に適用するかを指定します。今すぐ変更を適用するには、値を `true` に設定します。次のメンテナンスウィンドウ中に変更を適用するには、値を `false` に設定します。

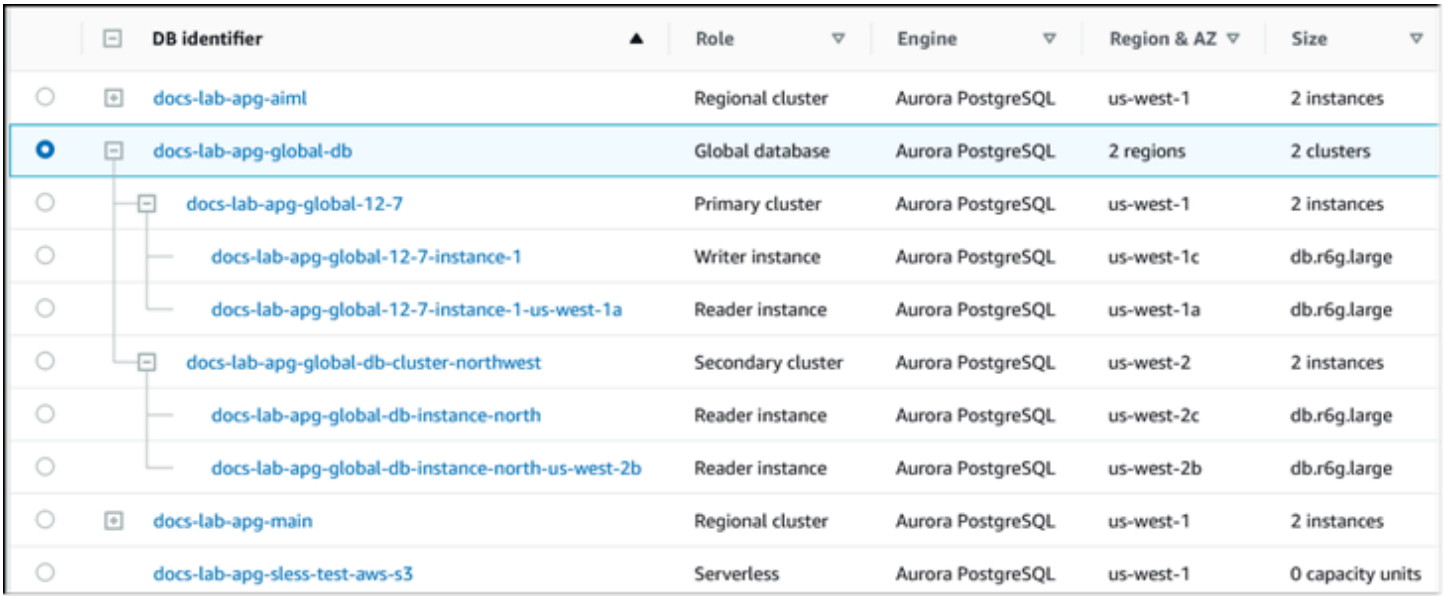
グローバルデータベースのメジャーアップグレード

Aurora グローバルデータベースクラスターの場合、アップグレードプロセスは Aurora グローバルデータベースを構成するすべての DB クラスターを同時にアップグレードします。これは、それぞれが同じ Aurora PostgreSQL バージョンを実行するようにするためです。また、システムテーブル、データファイル形式などの変更が、すべてのセカンダリクラスターに自動的にレプリケートされます。

グローバルデータベースクラスターを Aurora PostgreSQL の新しいメジャーバージョンにアップグレードするには、「本番稼働用の DB クラスターの新しいメジャーバージョンへのアップグレードをテストする」で説明されているように、アップグレードされたバージョンでアプリケーションをテストすることをお勧めします。本番稼働用の DB クラスターの新しいメジャーバージョンへのアップグレードをテストする の step 1. に詳述されているとおり、アップグレードの前に、Aurora グローバルデータベースの各 AWS リージョン に対して DB クラスターパラメータグループと DB パラメータグループの設定を必ず準備してください。

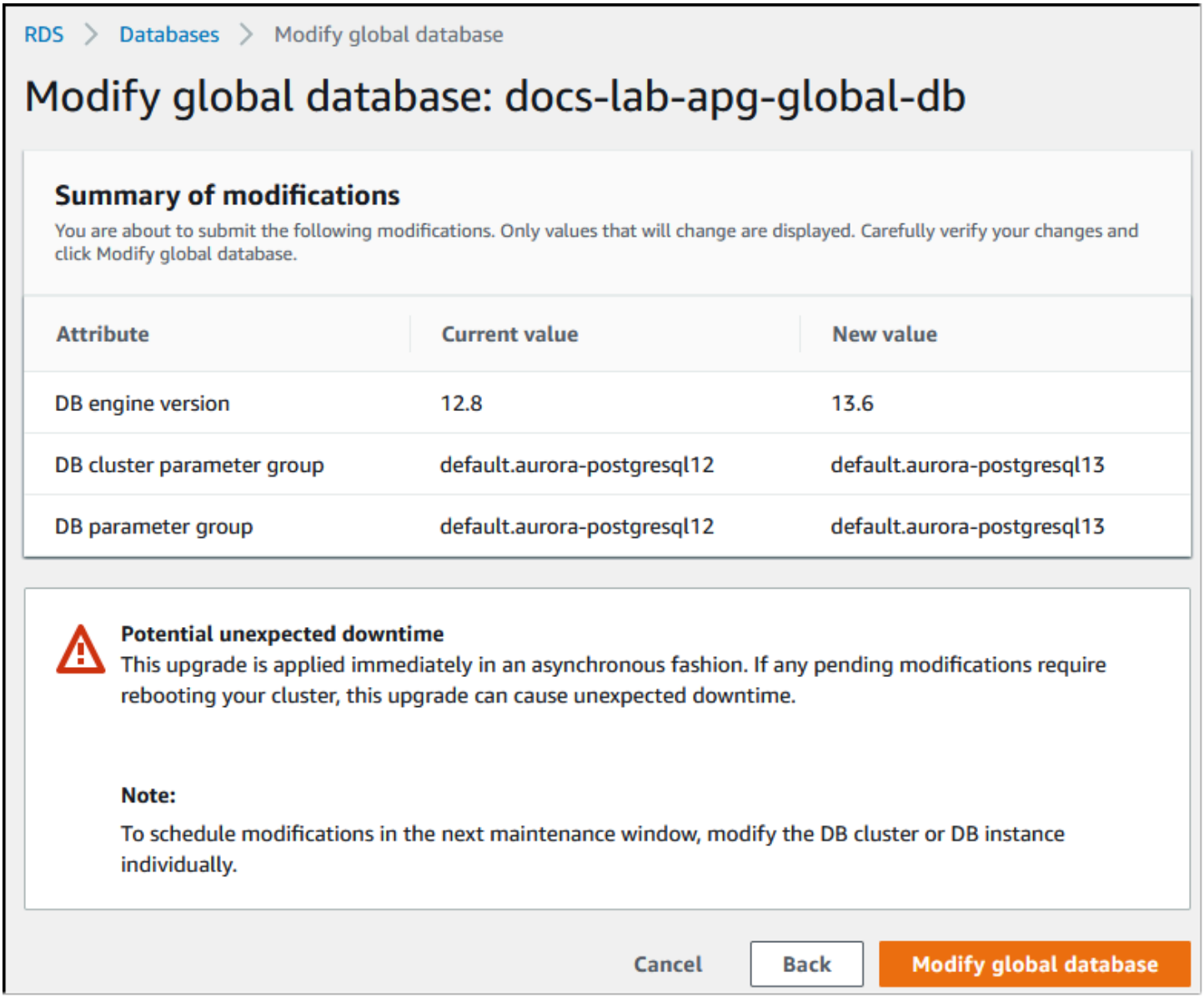
Aurora PostgreSQL グローバルデータベースクラスターに、`rds.global_db_rpo` パラメータに目標復旧時点 (RPO) が設定されている場合、アップグレードする前にパラメータをリセットしてください。RPO がオンになっている場合、メジャーバージョンのアップグレードプロセスは機能しません。デフォルトでは、このパラメータがオフになっています。Aurora PostgreSQL グローバルデータベースおよび RPO の詳細については、「Aurora PostgreSQL- ベースのグローバルデータベースの RPO (目標復旧時点) 管理」を参照してください。

新しいバージョンの試用版デプロイでアプリケーションが正常に実行できることを確認した場合は、アップグレードプロセスを開始できます。これを行うには、「Aurora PostgreSQL エンジンを新しいメジャーバージョンにアップグレードする」を参照してください。次の画像のように、RDS コンソールの [Databases] (データベース) リストから最上位の項目 [Global database] (グローバルデータベース) を必ず選択してください。



DB identifier	Role	Engine	Region & AZ	Size
docs-lab-apg-aiml	Regional cluster	Aurora PostgreSQL	us-west-1	2 instances
docs-lab-apg-global-db	Global database	Aurora PostgreSQL	2 regions	2 clusters
docs-lab-apg-global-12-7	Primary cluster	Aurora PostgreSQL	us-west-1	2 instances
docs-lab-apg-global-12-7-instance-1	Writer instance	Aurora PostgreSQL	us-west-1c	db.r6g.large
docs-lab-apg-global-12-7-instance-1-us-west-1a	Reader instance	Aurora PostgreSQL	us-west-1a	db.r6g.large
docs-lab-apg-global-db-cluster-northwest	Secondary cluster	Aurora PostgreSQL	us-west-2	2 instances
docs-lab-apg-global-db-instance-north	Reader instance	Aurora PostgreSQL	us-west-2c	db.r6g.large
docs-lab-apg-global-db-instance-north-us-west-2b	Reader instance	Aurora PostgreSQL	us-west-2b	db.r6g.large
docs-lab-apg-main	Regional cluster	Aurora PostgreSQL	us-west-1	2 instances
docs-lab-apg-sless-test-aws-s3	Serverless	Aurora PostgreSQL	us-west-1	0 capacity units

他の変更と同様に、プロンプトが表示されたらプロセスの続行を確認できます。



アップグレードプロセスを開始するには、コンソールではなく AWS CLI または RDS API を使用します。コンソールと同様、次のように、Aurora グローバルデータベースクラスターの構成要素ではなく、Aurora グローバルデータベースクラスターを操作します。

- modify-global-cluster AWS CLI コマンドを使用し、AWS CLI を使うことにより、Aurora グローバルデータベースのアップグレードを開始します。
- ModifyGlobalCluster API を使用して、アップグレードを開始します。

マイナーバージョンアップグレードを実行する前に

マイナーバージョンのアップグレード中のダウンタイムを低減するには、次のアクションを実行することをお勧めします。

- Aurora DB クラスターのメンテナンスは、トラフィックが少ない時間帯に実行する必要があります。メンテナンスウィンドウを適切に設定するには、Performance Insights を使用してこのような時間帯を特定します。Performance Insights については、「Amazon RDS での Performance Insights を使用したDB 負荷のモニタリング」を参照してください。DB クラスターのメンテナンスウィンドウの詳細については、「DB クラスターの適切なメンテナンスウィンドウの調整」を参照してください。
- エクスポネンシャルバックオフとジッターをサポートする AWS SDK を使用することが、ベストプラクティスです。詳細については、 ブログ投稿、「エクスポネンシャルバックオフとジッター」を参照してください。

マイナーバージョンのアップグレードとパッチの適用方法

マイナーバージョンのアップグレードとパッチは、厳格なテストの後にのみ AWS リージョン で利用可能となります。アップグレードとパッチをリリースする前に、Aurora PostgreSQL は、マイナーコミュニティバージョンのリリース後に発生する既知のセキュリティ問題、バグ、およびその他の問題により Aurora PostgreSQL フリートの全体的な安定性が損なわれないことをテストします。

Aurora PostgreSQL で新しいマイナーバージョンが利用できるようになるため、Aurora PostgreSQL DB クラスターを構成するインスタンスは、指定したメンテナンス期間中に自動的にアップグレードできます。これを行うには、Aurora PostgreSQL DB クラスターで [マイナーバージョン自動アップグレードの有効化] オプションがオンになっている必要があります。Aurora PostgreSQL DB クラスターを構成するすべての DB インスタンスで、マイナーアップグレードがクラスター全体に適用されるように、自動マイナーバージョンアップグレード (AmVU) オプションをオンにする必要があります。

> **Tip**
>
> Aurora PostgreSQL DB クラスターを構成するすべての PostgreSQL DB インスタンスに対して、[マイナーバージョン自動アップグレードの有効化] オプションが、オンになっていることを確認します。DB クラスター内のすべてのインスタンスを動作させるには、このオプションをオンにする必要があります。[マイナーバージョン自動アップグレード] の設定方法、およびクラスターレベルとインスタンスレベルで適用した場合にどのように機能するか

については、Aurora DB クラスターのマイナーバージョン自動アップグレード を参照してください。

次のクエリに describe-db-instances AWS CLIコマンドを使用することで、すべての Aurora PostgreSQL DB クラスターに対して、[マイナーバージョン自動アップグレードの有効化] オプションの値をチェックできます。

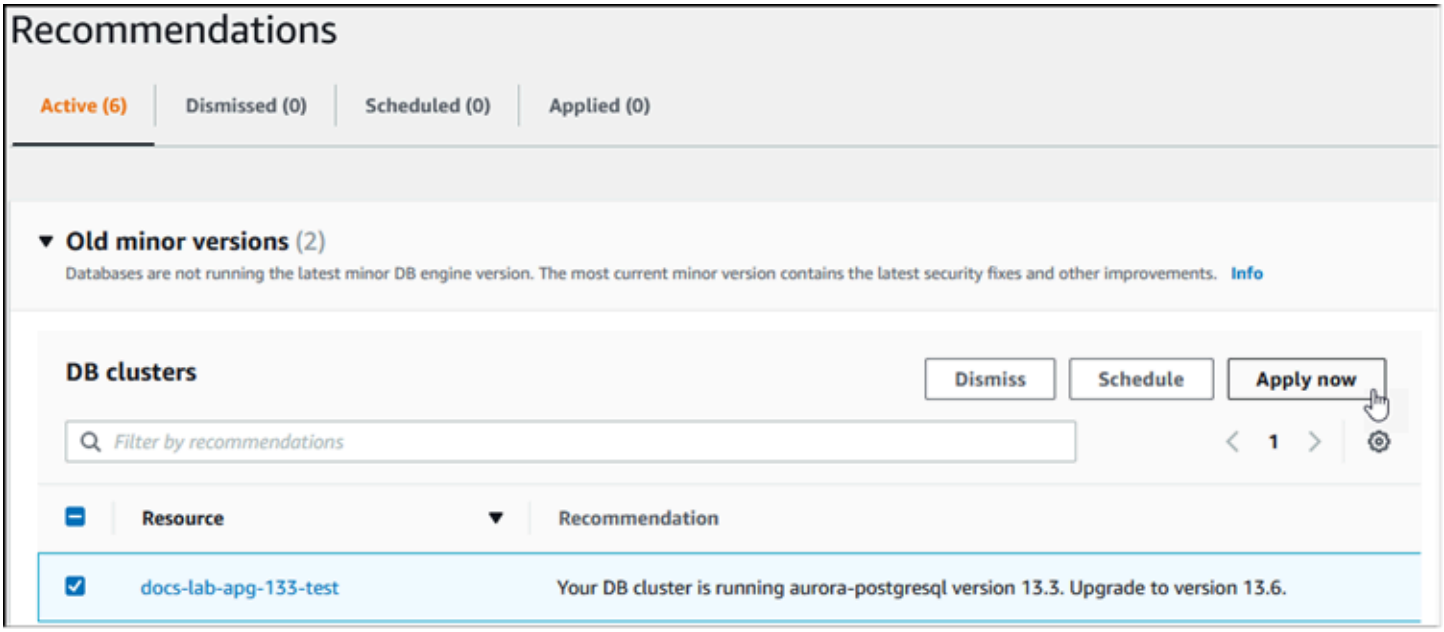
```
aws rds describe-db-instances \
  --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVer
```

このクエリにより、AutoMinorVersionUpgrade 設定のステータスに対して true または false 値を持つすべての Aurora DB クラスターとそのインスタンスのリストが返されます。表示されるコマンドは、AWS CLI がデフォルトの AWS リージョン に設定されていると仮定したものです。

[AmVU] オプションと Aurora DB クラスターを使用できるように変更する方法については、「Aurora DB クラスターのマイナーバージョン自動アップグレード」を参照してください。

Aurora PostgreSQL DB クラスターを新しいマイナーバージョンにアップグレードするには、メンテナンスタスクに応答するか、新しいバージョンを使用するようにクラスターを変更します。

RDS コンソールを使用して、レコメンデーションメニューを開くことにより、Aurora PostgreSQL DB クラスターで使用可能なアップグレードまたはパッチを特定することができます。そこには、古いマイナーバージョンなど、さまざまなメンテナンスの問題のリストを確認できます。本番環境に応じて、アップグレードをスケジュールするか、[今すぐ適用] を選択することですぐにアクションを実行することを選択できます。



パッチやマイナーバージョンアップグレードを手動で適用する方法など、Aurora DB クラスターのメンテナンス方法の詳細については、「Amazon Aurora DB クラスターのメンテナンス」を参照してください。

マイナーリリースのアップグレードとダウンタイムなしのパッチ適用プロセス

Aurora PostgreSQL DB クラスターのアップグレードには、停止の可能性が伴います。アップグレードプロセス中には、データベースがシャットダウンされます。データベースがビジー状態のときにアップグレードをスタートすると、DB クラスターが処理しているすべての接続とトランザクションが失われます。アップグレードを実行するためにデータベースがアイドル状態になるまで待機する場合は、長時間待機しなければならない場合があります。

ダウンタイムなしのパッチ適用 (ZDP) 機能により、アップグレードプロセスが改善されます。ZDP では、Aurora PostgreSQL DB クラスターへの影響を最小限に抑えながら、マイナーバージョンのアップグレードとパッチの両方を適用できます。ZDP は、Aurora PostgreSQL バージョンおよびこれらのマイナーバージョンおよび新しいメジャーバージョンのその他のリリースに、パッチまたは新しいマイナーバージョンアップグレードを適用する際に使用されます。つまり、これらのリリースのいずれかから新しいマイナーバージョンにアップグレードすると、ZDP が使用されます。

次の表に、ZDP が使用可能な Aurora MySQL バージョンと DB インスタンスクラスを示します。

Version	db.r* インスタンスクラス	db.t* インスタンスクラス	db.x* インスタンスクラス	db.serverless インスタンスクラス
10.21.0 以降の 10.21 バージョン	はい	はい	はい	該当なし
11.16.0 以降の 11.16 バージョン	はい	はい	はい	該当なし
11.17 以降のバージョン	はい	はい	はい	該当なし
12.11.0 以降の 12.11 バージョン	はい	はい	はい	該当なし
12.12 以降のバージョン	はい	はい	はい	該当なし
13.7.0 以降の 13.7 バージョン	はい	はい	はい	該当なし
13.8 以降のバージョン	はい	はい	はい	はい
14.3.1 以降の 14.3 バージョン	はい	はい	はい	該当なし
14.4.0 以降の 14.4 バージョン	はい	はい	はい	該当なし
14.5 以降のバージョン	はい	はい	はい	はい
15.3 以降のバージョン	はい	はい	はい	はい

ZDP は、Aurora PostgreSQL のアップグレードプロセス全体で、Aurora PostgreSQL DB クラスターへの現在のクライアント接続を維持することで機能します。ただし、以下の場合、ZDP が完了するまで接続は切断されます。

- 長期実行クエリまたはトランザクションが進行中である。
- データ定義言語 (DDL) ステートメントが実行中である。
- 一時テーブルまたはテーブルロックが使用中である。
- すべてのセッションが通知チャネルでリッスン中である。
- 「WITH HOLD」ステータスのカーソルが使用中である。
- TLSv1.3 または TLSv1.1 接続が使用中である。

ZDP によるアップグレードプロセス中、データベースエンジンはすべての新規トランザクションを一時停止するためのクワイエットポイントを探します。このアクションにより、パッチおよびアップグレードの際にデータベースが保護されます。トランザクションを一時停止してもアプリケーションがスムーズに実行されるように、コードに再試行ロジックを組み込むことをお勧めします。このアプローチにより、システムは短時間のダウンタイムを問題なく管理でき、アップグレード後に新しいトランザクションを再試行できます。

ZDP が正常に完了すると、接続停止のセッションを除いたアプリケーションセッションが保持され、アップグレードがまだ進行している間にデータベースエンジンが再起動します。データベースエンジンの再起動により、スループットが一時的に低下する可能性がありますが、これは通常数秒から約 1 分間程度で済みます。

場合によっては、ダウンタイムなしのパッチ適用 (ZDP) が成功しないこともあります。例えば、Aurora PostgreSQL DB クラスターまたはそのインスタンスが `pending` 状態にあるときにパラメータを変更すると、ZDP を中断する原因となります。

ZDP オペレーションのメトリックとイベントについては、コンソールの「イベント」ページを参照してください。イベントには、ZDP アップグレードの開始とアップグレードの完了が含まれます。このイベントでは、プロセスに要した時間、および再起動中に保持およびドロップされた接続の数を確認できます。詳細は、データベースのエラーログに表示されます。

Aurora PostgreSQL エンジンを新しいマイナーバージョンにアップグレードする

AWS CLI、または RDS API を使用することにより、Aurora PostgreSQL DB クラスターを新しいマイナーバージョンにアップグレードできます。アップグレードを実行する前に、メジャーバージョンのアップグレードする際に推奨する内容と同じベストプラクティスを実行することをお勧めします。

新しいメジャーバージョンと同様に、新しいマイナーバージョンでも、クエリ計画のリグレッションの原因となる修正などのオプティマイザの改善が行われている可能性があります。計画の安定性を確保するには、「メジャーバージョンのアップグレード後の計画の安定性の確保」で説明されているように、クエリ計画管理 (QPM) 拡張機能を使用することをお勧めします。

コンソール

Aurora PostgreSQL DB クラスターのエンジンバージョンをアップグレードするには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[データベース] を選択して、アップグレードする DB クラスターを選択します。
3. [Modify] を選択します。[DB クラスターの変更] ページが表示されます。
4. [Engine version] (エンジンバージョン) で、新しいバージョンを選択します。
5. [続行] を選択して、変更の概要を確認します。
6. 変更をすぐに反映させるには、[Apply immediately] を選択します。このオプションを選択すると、停止状態になる場合があります。詳細については、「Amazon Aurora DB クラスターの変更」を参照してください。
7. 確認ページで、変更内容を確認します。正しい場合は、[クラスターの変更] を選択して変更を保存します。

   または、[戻る] を選択して変更を編集するか、[キャンセル] を選択して変更をキャンセルします。

AWS CLI

DB クラスターのバージョンをアップグレードするには、次のパラメータに modify-db-cluster AWS CLI コマンドを使用します。

- `--db-cluster-identifier` — Aurora PostgreSQL DB クラスターの名前。
- `--engine-version` - アップグレード先のデータベースエンジンのバージョン番号です。有効なエンジンバージョンの詳細については、AWS CLI の describe-db-engine-versions コマンドを参照してください。
- `--no-apply-immediately` - 次のメンテナンス時間中に変更を適用します。今すぐ変更を適用するには、代わりに `--apply-immediately` を使用します。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --engine-version new_version \
    --no-apply-immediately
```

Windows の場合:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --engine-version new_version ^
    --no-apply-immediately
```

RDS API

DB クラスターのエンジンのバージョンをアップグレードするには、ModifyDBCluster オペレーションを使用します。以下のパラメータを指定します。

- `DBClusterIdentifier` - DB クラスターの名前、例えば *mydbcluster* です。
- `EngineVersion` - アップグレード先のデータベースエンジンのバージョン番号です。有効なエンジンバージョンについては、DescribeDBEngineVersions オペレーションを使用します。
- `ApplyImmediately` - 変更をすぐに適用するか、次のメンテナンスウィンドウ中に適用するかを指定します。今すぐ変更を適用するには、値を `true` に設定します。次のメンテナンスウィンドウ中に変更を適用するには、値を `false` に設定します。

PostgreSQL 拡張機能のアップグレード

Aurora PostgreSQL DB クラスターを新しいメジャーバージョンまたはマイナーバージョンにアップグレードしても、PostgreSQL 拡張機能が同時にアップグレードされるわけではありません。ほとんどの拡張機能については、メジャーバージョンまたはマイナーバージョンのアップグレード完了後にアップグレードします。ただし、場合によっては、Aurora PostgreSQL DB エンジンをアップグレードする前に拡張機能をアップグレードすることもあります。詳細については、本番稼働用の DB クラスターの新しいメジャーバージョンへのアップグレードをテストするの「list of extensions to update」を参照してください。

PostgreSQL 拡張機能をインストールするには、`rds_superuser` 権限が必要です。通常、特定の拡張機能の管理を容易にするために、`rds_superuser` が特定の拡張機能に対する権限を関連す

るユーザー (ロール) に委任します。つまり、Aurora PostgreSQL DB クラスター内のすべての拡張機能をアップグレードするタスクには、さまざまなユーザー (ロール) が含まれる可能性があります。スクリプトを使用してアップグレードプロセスを自動化する場合も、この点に注意してください。PostgreSQL の権限とロールの詳細については、「Amazon Aurora PostgreSQL でのセキュリティ」を参照してください。

> **Note**
>
> PostGIS 拡張機能の更新については、「PostGIS 拡張機能を使用した空間データの管理 (ステップ 6: PostGIS 拡張機能を更新する)」を参照してください。
> pg_repack 拡張機能を更新する場合、拡張機能をドロップしてアップグレードされた DB インスタンスに新しいバージョンを作成します。詳細については、「pg_repack ドキュメント」の「pg_repack installation」(pg_repack のインストール) を参照してください。

エンジンのアップグレード後に拡張機能を更新するには、ALTER EXTENSION UPDATE コマンドを使用します。

```
ALTER EXTENSION extension_name UPDATE TO 'new_version';
```

現在インストールされている拡張機能を一覧表示するには、次のコマンドで PostgreSQL の pg_extension カタログを使用します。

```
SELECT * FROM pg_extension;
```

インストールで使用可能な特定の拡張機能バージョンのリストを表示するには、次のコマンドで PostgreSQL の pg_available_extension_versions ビューを使用します。

```
SELECT * FROM pg_available_extension_versions;
```

代替のブルー/グリーンのアップグレードテクニック

状況によっては、古いクラスターからアップグレードされたクラスターへの即時の切り替えが最優先事項です。このような場合、古いクラスターと新しいクラスターを並べて実行するマルチステッププロセスを使用できます。ここでは、新しいクラスターが引き継ぐ準備ができるまで、古いクラスターから新しいクラスターにデータをレプリケートします。詳細については、「データベースの更新にブルー/グリーンデプロイを使用する」を参照してください。

Aurora PostgreSQL 長期サポート (LTS) リリース

新しい Aurora PostgreSQL バージョンは、それぞれ DB クラスターを作成またはアップグレードする際に一定期間使用できます。この期間後は、そのバージョンを使用するためにクラスターをアップグレードする必要があります。クラスターのアップグレードは、サポート期間終了の前にユーザーがを手動で行うことも、 Aurora PostgreSQL バージョンのサポートが終了する際に、Aurora により自動的に行わせることもできます。

Aurora により、適切な Aurora PostgreSQL バージョンが、長期サポート (LTS) のリリースとして指定されます。LTS リリースを使用するデータベースクラスターでは、非 LTS リリースを使用するクラスターと比べて、同じバージョンを長く使用することが可能で、アップグレードサイクルが少なくなります。LTS マイナーバージョンには、(パッチバージョンとして) バグ修正のみが含まれています。LTS バージョンには、導入後にリリースされた新機能は含まれません。

LTS マイナーバージョンで実行されている DB クラスターには、1 年に 1 回、LTS リリース用の最新パッチバージョンによるパッチ修正が行われます。このパッチ手法は、セキュリティと安定性に関する累積的な修正からのメリットを確実に適用するためのものです。セキュリティ関連など、適用すべき重要な修正が存在する場合には、より頻繁なパッチ修正を、LTS マイナーバージョンに対し行うことがあります。

> **Note**
>
> LTS マイナーバージョンの使用を、そのライフサイクルの間にわたり継続するには、DB インスタンスの [自動マイナーバージョンアップグレード] を無効にします。DB クラスターで、LTS マイナーバージョンによる自動的なアップグレードが行われないようにするには、Aurora クラスターのすべての DB インスタンスにおいて、[自動マイナーバージョンアップグレード] に [いいえ] を設定します。

ほとんどの Aurora PostgreSQL クラスターについては、LTS リリースを使用するのではなく、最新リリースへのアップグレードが推奨されます。これにより、マネージドサービスとして Aurora を利用して最新の機能とバグ修正にアクセスできます。LTS リリースは、以下のような特性を持つクラスターを対象としています。

- 稀に発生する重要なパッチ以外で、Aurora PostgreSQL アプリケーションのアップグレードのために、ダウンタイムを許容することができない場合。
- 各回の Aurora PostgreSQL データベースエンジンの更新において、クラスターおよび関連アプリケーションのテストサイクルが長い時間を要する場合。

- Aurora PostgreSQL クラスターのデータベースバージョンに、アプリケーションに必要なすべての DB エンジン機能とバグ修正が含まれている場合。

Aurora PostgreSQL の現在の LTS リリースは次のとおりです。

- PostgreSQL 14.6 2023 年 1 月 20 日にリリースされました。詳細については、Aurora PostgreSQL のリリースノートの「PostgreSQL 14.6」を参照してください。
- PostgreSQL 13.9。2023 年 1 月 20 日にリリースされました。詳細については、Aurora PostgreSQL のリリースノートの「PostgreSQL 13.9」を参照してください。
- PostgreSQL 12.9 2022 年 2 月 25 日にリリースされました。詳細については、Aurora PostgreSQL のリリースノートの「PostgreSQL 12.9」を参照してください。
- PostgreSQL 11.9 (Aurora PostgreSQL リリース 3.4)。2020 年 12 月 11 日にリリースされました。このバージョンの詳細については、Aurora PostgreSQL のリリースノートの「PostgreSQL 11.9, Aurora PostgreSQL リリース 3.4」を参照してください。

Aurora とデータベースエンジンのバージョンを特定する方法については、「Amazon Aurora PostgreSQL のバージョンの識別」を参照してください。

Amazon Aurora Global Database の使用

Amazon Aurora Global Database は複数の AWS リージョン にまたがり配置されます。これにより、低レイテンシーのグローバル読み取りを実現し、AWS リージョン 全体に影響が及ぶ可能性のある停止がまれに起きても、すばやい復旧を可能にします。Aurora Global Database には、1 つのリージョンにプライマリ DB クラスターがあり、異なるリージョンに最大 5 つのセカンダリ DB クラスターがあります。

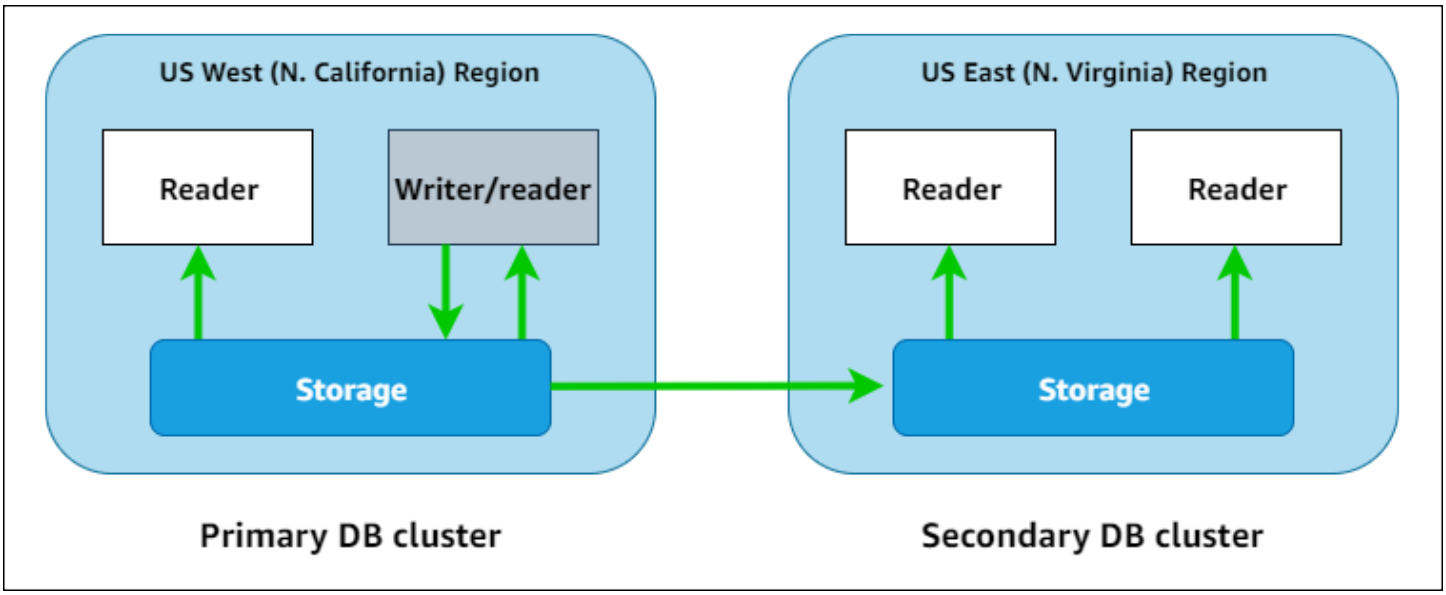
トピック

Amazon Aurora Global Database の概要

Amazon Aurora Global Database により、複数の AWS リージョン にまたがる単一の Aurora データベースを使用して、グローバルに分散したアプリケーションを実行できます。

Aurora Global Database は、データが書き込まれる 1 つのプライマリ AWS リージョン と、最大 5 つの読み取り専用セカンダリ AWS リージョン で構成されます。書き込みオペレーションは、プライマリ AWS リージョン 内のプライマリ DB クラスターに直接発行します。Aurora は、専用インフラストラクチャを使用しながら、通常 1 秒未満のレイテンシーで、データをセカンダリ AWS リージョン にレプリケートします。

次の図に、2 つの AWS リージョン にまたがる Aurora Global Database の例を示します。



各セカンダリクラスターは、読み取り専用のワークロードを処理するために 1 つまたは複数の Aurora レプリカ (読み取り専用の Aurora DB インスタンス) を追加することで、個別にスケールアップすることができます。

書き込みオペレーションはプライマリクラスターのみが実行します。書き込みオペレーションを実行するクライアントは、プライマリクラスターの DB クラスターエンドポイントに接続します。図に示すように、Aurora Global Database では、レプリケーションにはデータベースエンジンではなく、クラスターストレージボリュームを使用します。詳細については、「Amazon Aurora ストレージの概要」を参照してください。

Aurora Global Database は、ワールドワイドなフットプリントを持つアプリケーション向けに設計されています。読み取り専用セカンダリ DB クラスター (AWS リージョン) を使用することで、アプリケーションユーザーのさらに近くで読み取りオペレーションをサポートできます。書き込み転送機能を使用することで、Aurora Global Database をセカンダリのクラスターからプライマリクラスターにデータを送信するように構成することもできます。詳細については、「Amazon Aurora Global Database の書き込み転送を使用する」を参照してください。

プライマリ DB クラスターのリージョン変更にあたって、Aurora グローバルデータベースはシナリオに応じて、グローバルデータベーススイッチオーバーとグローバルデータベースフェイルオーバーの操作をサポートしています。

- 地域ローテーションなどの計画済みの運用手順には、グローバルデータベーススイッチオーバー(以前は「管理型計画フェイルオーバー」と呼ばれていました) を使用します。この機能を使用すると、正常な Aurora Global Database のプライマリクラスターを、データを損失せず、セカ

ンダリリージョンの 1 つに再配置できます。詳細については、「Amazon Aurora Global Database に対するスイッチオーバーの実行」を参照してください。

- プライマリリージョンの機能停止後に、Aurora Global Database を復旧するには、グローバルデータベースフェイルオーバーを使用します。この特徴量では、プライマリ DB クラスターを別のリージョンにフェイルオーバーします (クロスリージョンフェイルオーバー)。詳細については、「Aurora Global Database のマネージドフェイルオーバーを実行する」を参照してください。

Amazon Aurora Global Database の利点

Aurora Global Database を使用すると、次の利点を得ることができます。

- ローカルのレイテンシーによるグローバルな読み取り – 世界中にオフィスを持つ企業は、Aurora Global Database を使用することで、プライマリ AWS リージョン にある自社の主要な情報源を最新状態に保つことができます。他のリージョンにあるオフィスは、自社のリージョンにある情報にローカルのレイテンシーでアクセスすることができます。
- スケーラブルなセカンダリ Aurora DB クラスター – セカンダリクラスターは、読み取り専用のインスタンス (Aurora レプリカ) をセカンダリ AWS リージョン にさらに追加することでスケールできます。セカンダリクラスターは読み取り専用です。したがって、読み取り専用の Aurora レプリカインスタンスを、1 つの Aurora クラスターにつき、通常の 15 件ではなく最大 16 件サポートします。
- プライマリからセカンダリの Aurora DB クラスターへの迅速なレプリケーション-Aurora Global Database によるレプリケーションは、プライマリ DB クラスターのパフォーマンスにほとんど影響しません。DB インスタンスのリソースは、全面的にアプリケーションの読み取りおよび書き込みワークロードに当てられます。
- リージョン全体にわたる停止からの回復 – セカンダリクラスターを使用すると、従来のレプリケーションソリューションと比較して迅速 (低い RTO) かつ少ないデータ損失 (低い RPO) で、Aurora Global Database を新しいプライマリ AWS リージョン で使用できるようになります。

リージョンとバージョンの可用性

機能の可用性とサポートは、各 Aurora データベースエンジンの特定のバージョン、および AWS リージョン によって異なります。Aurora とグローバルデータベースでのバージョンとリージョン可用性については、「Aurora グローバルデータベースでサポートされているリージョンと DB エンジン」を参照してください。

Amazon Aurora Global Database の制限

現在、Aurora Global Database には以下の制限があります。

- Aurora Global Database を使用できるのは、一部の AWS リージョン、および Aurora MySQL と Aurora PostgreSQL の特定バージョンのみです。詳細については、「Aurora グローバルデータベースでサポートされているリージョンと DB エンジン」を参照してください。
- Aurora Global Database には、サポートされる Aurora DB インスタンスクラス、AWS リージョン の最大数などに関する特定の設定要件があります。詳細については、「Amazon Aurora Global Database の構成要件」を参照してください。
- MySQL 5.7 と互換性のある Aurora MySQL の場合、Aurora グローバルデータベースのスイッチオーバーにはバージョン 2.09.1 以降のマイナーバージョンが必要です。
- プライマリ DB クラスターとセカンダリ DB クラスターが同じメジャー、マイナー、パッチレベルのエンジンバージョンである場合にのみ、Aurora グローバルデータベースでマネージドクロスリージョンスイッチオーバーまたはフェイルオーバーを実行できます。ただし、マイナーエンジンバージョンが次のいずれかである場合、パッチレベルは異なる場合があります。

データベースエンジン	マイナーエンジンバージョン
Aurora PostgreSQL	• バージョン 14.5 以上 のマイナーバージョン • バージョン 13.8 以上 のマイナーバージョン • バージョン 12.12 以上 のマイナーバージョン • バージョン 11.17 以上 のマイナーバージョン

  詳細については、「マネージドクロスリージョンスイッチオーバーまたはフェイルオーバーに対するパッチレベルの互換性」を参照してください。
- Aurora Global Database は、現在、以下の Aurora 機能をサポートしていません。
  - Aurora Serverless v1
  - Aurora でのバックトラック

- グローバルデータベースで RDS プロキシ機能を使用する際の制限については、「グローバル データベースを使用した RDS Proxy の制約事項」を参照してください。
- マイナーバージョンの自動アップグレードは、Aurora Global Database の一部である Aurora MySQL クラスターや Aurora PostgreSQL クラスターには適用されません。なお、グローバルデータベースクラスターの一部である DB インスタンスに対してこの設定を指定できますが、その設定に効果はありません。
- Aurora Global Database は、現在、セカンダリ DB クラスターの Aurora Auto Scaling をサポートしていません。
- Aurora MySQL 5.7 を実行している Aurora グローバルデータベースでデータベースアクティビティストリームを使用するには、エンジンバージョンがバージョン 2.08 以降である必要があります。データベースアクティビティストリーミングについては、「データベースアクティビティストリームを使用した Amazon Aurora のモニタリング」を参照してください。
- 現在、Aurora Global Database のアップグレードには以下の制限があります。
  - Aurora グローバルデータベースのメジャーバージョンアップグレードを実行している間、グローバルデータベースクラスターにカスタムパラメータグループを適用できません。グローバルクラスターの各リージョンにカスタムパラメータグループを作成し、アップグレード後に手動でリージョンクラスターに適用します。
  - Aurora MySQL に基づく Aurora グローバルデータベースでは、`lower_case_table_names` パラメータがオンの場合、Aurora MySQL バージョン 2 からバージョン 3 へのインプレースアップグレードを実行できません。使用できる方法の詳細については、「メジャーバージョンのアップグレード」を参照してください。
  - Aurora PostgreSQL に基づく Aurora グローバルデータベースでは、目標復旧時点 (RPO) 機能がオンになっている場合、Aurora DB エンジンのメジャーバージョンアップグレードを実行できません。RPO 機能については、「Aurora PostgreSQL- ベースのグローバルデータベースの RPO (目標復旧時点) 管理」を参照してください。
  - Aurora MySQL に基づく Aurora グローバルデータベースでは、標準プロセスを使用して、バージョン 3.01 または 3.02 から 3.03 以降へのマイナーバージョンアップグレードを実行できません。使用するプロセスの詳細については、「エンジンのバージョンを変更して Aurora MySQL アップグレードする」を参照してください。

  Aurora Global Database のアップグレードについては、「Amazon Aurora Global Database のアップグレード」を参照してください。
- Aurora Global Database にある Aurora DB クラスターは、個別に停止またはスタートすることはできません。詳細については、「Amazon Aurora DB クラスターの停止と開始」を参照してください。

- セカンダリ Aurora DB クラスターにアタッチされた Aurora レプリカは、特定の場合に再起動することが可能です。プライマリ AWS リージョン のライター DB インスタンスが再起動またはフェイルオーバーすると、セカンダリリージョンにある Aurora レプリカも再起動します。このセカンダリクラスターは、その後すべてのレプリカがプライマリ DB クラスターのライターインスタンスと再び同期するまでは、使用できません。再起動またはフェイルオーバー時のプライマリクラスターの動作は、単一で非グローバルな DB クラスターの動作と同じです。詳細については、「Amazon Aurora でのレプリケーション」を参照してください。

  プライマリ DB クラスターに変更を加えるときは、必ず事前に、Aurora Global Database への影響を把握してください。詳細については、「予期しない停止からの Amazon Aurora Global Database の復旧」を参照してください。
- Aurora Global Database は現在、Amazon Aurora が DB クラスターの AWS KMS キーにアクセスできなくなった場合の `inaccessible-encryption-credentials-recoverable` ステータスをサポートしていません。このような場合、暗号化された DB クラスターはターミナルの `inaccessible-encryption-credentials` 状態になります。上記の状態についての詳細は、「DB クラスターステータスの表示」を参照してください。
- Aurora Global Database で実行されている Aurora PostgreSQL- ベースのDBクラスターには、以下の制限があります。
  - クラスターキャッシュ管理は、Aurora Global Database の一部である Aurora PostgreSQL DB クラスターではサポートされません。
  - Aurora Global Database のプライマリ DB クラスターが Amazon RDS PostgreSQL インスタンスのレプリカをベースとしている場合、セカンダリクラスターを作成することはできません。そのクラスターから、AWS Management Console、AWS CLI、または `CreateDBCluster` API オペレーションを使用してセカンダリを作成しようとしないでください。作成しようとするとタイムアウトし、セカンダリクラスターは作成されません。

プライマリと同じバージョンの Aurora DB エンジンを使用して、Aurora Global Database のセカンダリ DB クラスターを作成することを推奨します。詳細については、「Amazon Aurora Global Database の作成」を参照してください。

Amazon Aurora Global Database のスタート方法

Aurora Global Database の使用をスタートするには、まず、使用する Aurora DB エンジンと AWS リージョン を決定します。Aurora Global Database をサポートしているのは、特定の AWS リージョン にある Aurora MySQL と Aurora PostgreSQL のデータベースエンジンの、特定のバージョン

のみです。完全なリストについては、「Aurora グローバルデータベースでサポートされているリージョンと DB エンジン」を参照してください。

Aurora Global Database は、次のいずれかの方法で作成できます。

- 新しい Aurora DB クラスターと Aurora DB インスタンスを使用して新しい Aurora Global Database を作成する - これを行うには、Amazon Aurora Global Database の作成 のステップとおりに進めてください。プライマリ Aurora DB クラスターを作成した後、「AWS リージョン の Amazon Aurora Global Database への追加」の手順に従ってセカンダリ AWS リージョン を追加します。
- Aurora Global Database 機能をサポートする既存の Aurora DB クラスターを使用し、それに AWS リージョン を追加する – この操作は、既存の Aurora DB クラスターが Aurora グローバルモードをサポートする DB エンジンバージョンを使用しているか、グローバル互換性がある場合にのみ実行できます。DB エンジンのバージョンの中には、このモードが明確であるものと、そうでないものがあります。

  Aurora DB クラスターを選択した際に、AWS Management Console 上の [アクション] で [リージョンを追加] を選択できるかどうかを確認します。可能であれば、その Aurora DB クラスターを Aurora グローバルクラスターに使用できます。詳細については、「AWS リージョン の Amazon Aurora Global Database への追加」を参照してください。

Aurora Global Database を作成する前に、すべての構成要件を把握しておくことを推奨します。

トピック

- Amazon Aurora Global Database の構成要件
- Amazon Aurora Global Database の作成
- AWS リージョン の Amazon Aurora Global Database への追加
- セカンダリリージョンでのヘッドレス Aurora DB クラスターの作成
- Amazon Aurora Global Database のスナップショットの使用

Amazon Aurora Global Database の構成要件

Aurora Global Database は、2 つ以上の AWS リージョン にまたがり配置されます。プライマリ AWS リージョン は、1 つの書き込み Aurora DB インスタンスを持つ Aurora DB クラスターをサポートします。セカンダリ AWS リージョン は、Aurora レプリカ全体で構成される、読み取り専用の Aurora DB クラスターを実行します。セカンダリ AWS リージョン は最低 1 つが必要で、ま

た、Aurora Global Database では、セカンダリ AWS リージョン を最大 5 つ持つことができます。この表は、Aurora Global Database で許容されている Aurora DB クラスター、Aurora DB インスタンス、Aurora レプリカの最大数をリスト化したものです。

説明	プライマリ AWS リージョン	セカンダリ AWS リージョン
Aurora DB クラスター	1	5 (最大)
ライターインスタンス	1	0
読み取り専用インスタンス (Aurora レプリカ)、Aurora DB クラスターあたり	15 (最大数)	16 (総数)
読み取り専用インスタンス (最大許容数、指定されたセカンダリリージョンの実数)	15 - s	s = セカンダリ AWS リージョンの総数

Aurora Global Database を構成している Aurora DB クラスターには、以下の固有の要件があります。

- DB インスタンスクラスの要件-Aurora Global Database には、メモリを大量に消費するアプリケーションに最適化された DB インスタンスクラスが必要です。メモリ最適化 DB インスタンスクラスの詳細については、DB インスタンスクラスを参照してください。db.r5 以上のインスタンスクラスを使用することを推奨します。
- AWS リージョン の要件 – Aurora Global Database には、1 つの AWS リージョン に置かれた 1 つのプライマリ Aurora DB クラスターと、別のリージョンに置かれた 1 つ以上のセカンダリ Aurora DB クラスターが必要です。セカンダリ (読み取り専用) Aurora DB クラスターは最大 5 つ作成できます。それぞれが異なるリージョンに存在していなければなりません。つまり、Aurora Global Database にある 2 つの Aurora DB クラスターを、同じ AWS リージョン 内に置くことはできません。
- 命名の要件 – それぞれの Aurora DB クラスター用に選択する名前は、すべての AWS リージョンにおいて一意でなければなりません。異なる Aurora DB クラスターに、たとえそれらが別のリージョンに置かれていたとしても、同じ名前を付けることはできません。
- Aurora Serverless v2 の容量要件 — Aurora Serverless v2 のグローバルデータベースの場合、プライマリ AWS リージョン の DB クラスターに必要な最小容量は 8 ACU です。

このセクションの手順を実行するには、AWS アカウント が必要です。Amazon Aurora で作業を行うためのセットアップタスクを完了してください。詳細については、「Amazon Aurora の環境をセットアップする」を参照してください。Aurora DB クラスターを作成するための他の準備ステップも完了する必要があります。詳細については、「Amazon Aurora DB クラスターの作成」を参照してください。

Amazon Aurora Global Database の作成

場合によっては、グローバル互換性がある Aurora データベースエンジンを実行している既存の Aurora プロビジョニングされた DB クラスターがある場合があります。その場合は、そこに別の AWS リージョン を追加して、Aurora Global Database を作成できます。これを行うには、「AWS リージョン の Amazon Aurora Global Database への追加」を参照してください。

Aurora Global Database は、AWS Management Console、AWS CLI、または RDS API を使用し、次のステップに従って作成します。

コンソール

Aurora Global Database 作成の最初のステップは、Aurora グローバルデータベース機能をサポートしている AWS リージョン へのサインインです。詳細な一覧については、「Aurora グローバルデータベースでサポートされているリージョンと DB エンジン」を参照してください。

以下のステップの一環として、Aurora DB クラスターの Amazon VPC に基づいて Virtual Private Cloud (VPC) を選択します。独自の VPC を使用するには、事前に作成して、選択できるようにすることをお勧めします。同時に、関連するサブネットを作成し、必要に応じてサブネットグループとセキュリティグループを作成します。詳細については、「チュートリアル: DB インスタンス用の Amazon VPC を作成する」を参照してください。

Aurora DB クラスターの作成に関する全般情報については、「Amazon Aurora DB クラスターの作成」を参照してください。

Aurora Global Database を作成するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. [データベースの作成] を選択します。[Create database (データベースの作成)] ページで、次の操作を行います。

   - データベースの作成方法として、[Standard create (スタンダード作成)] を選択します。([簡易作成] を選択しないでください)。

- [エンジンオプション] セクションの Engine type で、該当するエンジンタイプとして、[Aurora (MySQL 互換)] または [Aurora (PostgreSQL 互換)] を選択します。

3. 引き続き、以下の手順のステップを使用して、Aurora グローバルデータベースを作成します。

Aurora MySQL を使用するグローバルデータベースの作成

以下のステップは、Aurora MySQL のすべてのバージョンに適用されます。

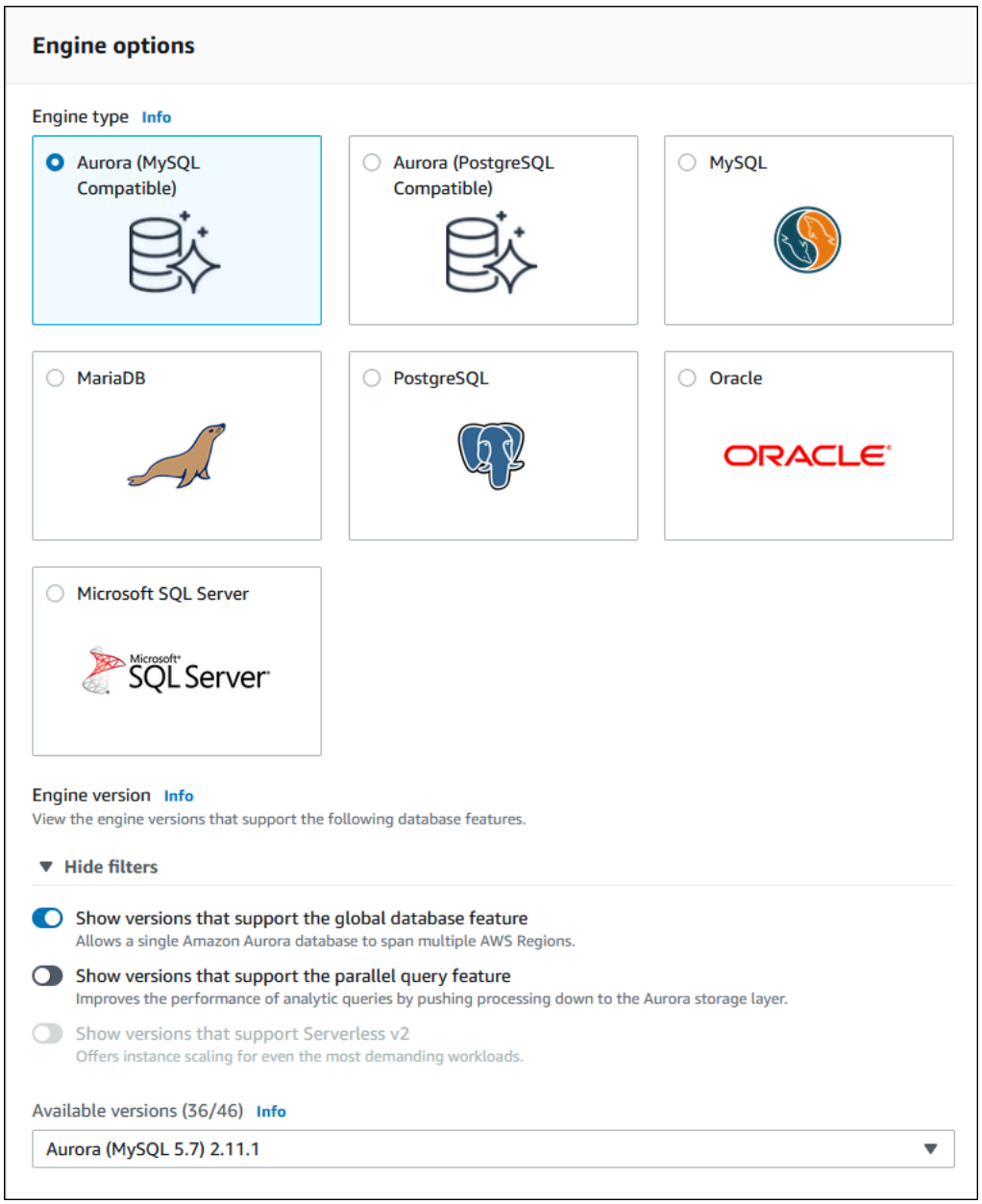
Aurora MySQL を使用して Aurora グローバルデータベースを作成するには

[データベースの作成] ページに記入します。

1. [Engine options (エンジンオプション)] で、以下の項目を選択します。

   a. [Show filters] (フィルターの表示) を展開し、[Show versions that support the global database feature] (グローバルデータベース機能をサポートするバージョンを表示する) をオンにします。

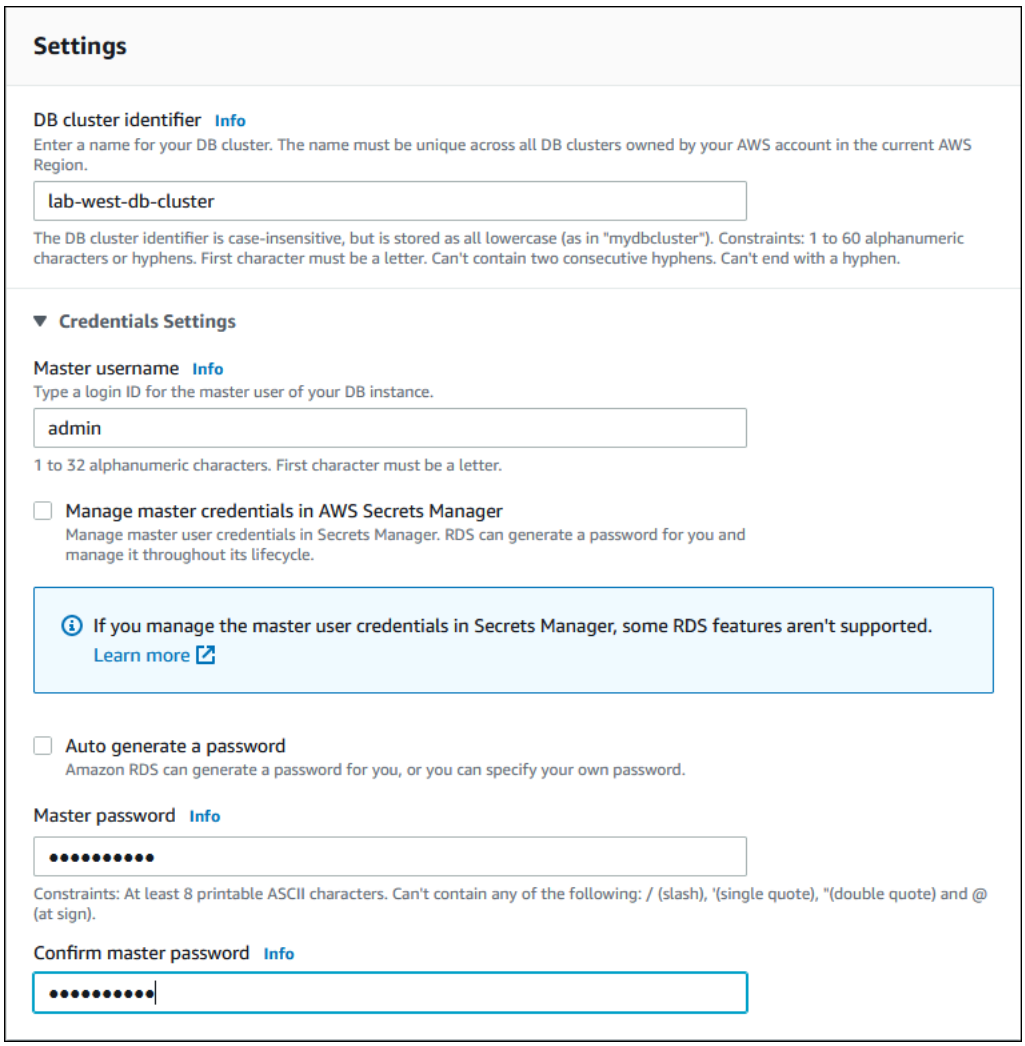
   b. [Engine version] (エンジンバージョン) では、Aurora Global Database に使用する Aurora MySQL のバージョンを選択します。



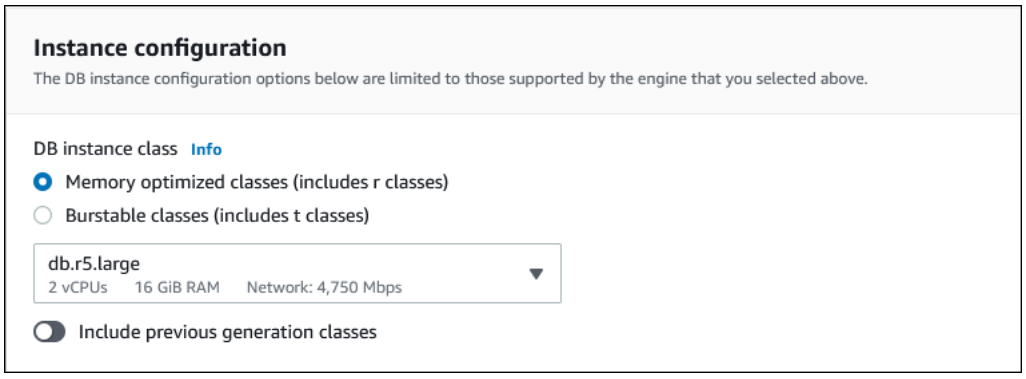
2. [テンプレート] では、[本番稼働] を選択します。あるいは、ユースケースに適していれば [Dev/Test (開発/テスト)] を選択することもできます。本番稼働環境では、開発/テストを使用しないでください。
3. [設定] では、以下の操作を行います。

   a. DB クラスター識別子にわかりやすい名前を入力します。Aurora Global Database の作成が完了すると、プライマリ DB クラスターはこの名前で識別されます。

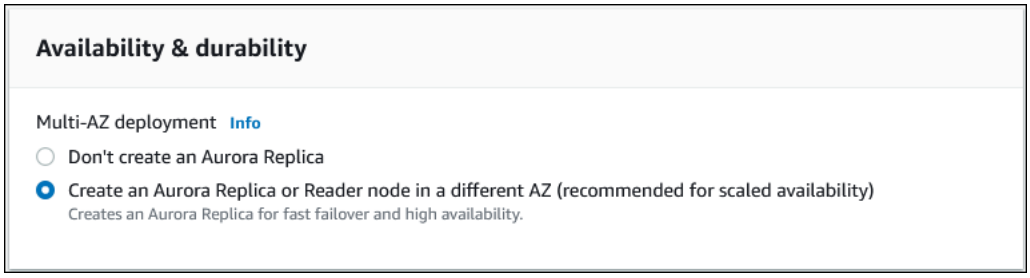
   b. DB インスタンスの `admin` ユーザーアカウント用に独自のパスワードを入力します。あるいは Aurora で生成します。[autogenerate a password (パスワードの自動生成)] をクリックすると、パスワードをコピーするためのオプションが表示されます。



4. [DB instance class (DB インスタンスクラス)] で、`db.r5.large` または別のメモリ最適化 DB インスタンスクラスを選択します。db.r5 以上のインスタンスクラスを使用することを推奨します。



5. [Availability & durability (可用性と耐久性の高い)] では、Aurora に、Aurora レプリカを異なるアベイラビリティーゾーン (AZ) に作成させるように、選択することを推奨します。Aurora レプリカをすぐに作成しないときは、後で作成する必要があります。



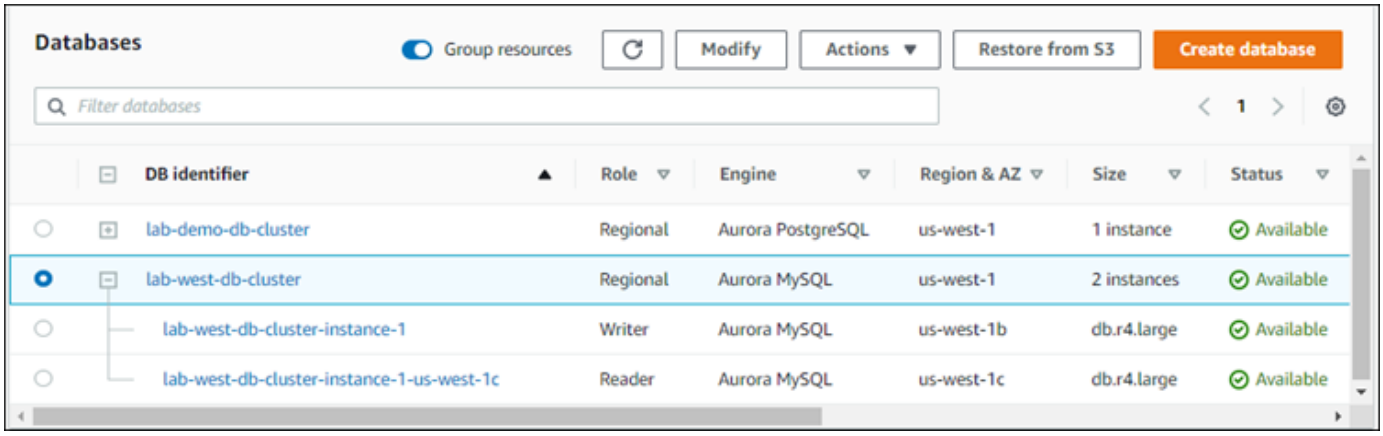
6. [接続] では、この DB インスタンスの仮想ネットワーク環境を定義している Amazon VPC に基づいた、Virtual Private Cloud (VPC) を選択します。デフォルトを選択するとこのタスクを簡略化できます。
7. [Database authentication (データベース認証)] の設定を完了します。このプロセスを簡略化するには、今すぐに [Password authentication (パスワード認証)] を選択して、AWS Identity and Access Management (IAM) のセットアップは後で行います。
8. [追加設定] では、以下の操作を行います。

   a. [初期のデータベース名] に名前を入力して、このクラスターのプライマリ Aurora DB インスタンスを作成します。これは、Aurora プライマリ DB クラスターの書き込みノードです。

      使用するカスタムパラメータグループが既にある場合を除いて、DB クラスターパラメータグループおよび DB パラメータグループは、デフォルトを選択したままにします。

   b. [Backtrack を有効にする] チェックボックスがオンになっている場合、そのチェックボックスをオフにします。Aurora Global Database は Backtrack をサポートしていません。[Additional configuration (追加構成)] では、その他デフォルトの設定を受け入れて構いません。
9. [データベースの作成] を選択します。

   Aurora が Aurora DB インスタンス、Aurora レプリカ、Aurora DB クラスターの作成プロセスを完了するまで、数分かかることがあります。Aurora DB クラスターが Aurora Global Database のプライマリ DB クラスターとして使用できる状態であるかは、ステータスによって確認できます。その場合は、次に示すように、ライターノードとレプリカノードの状態が [Available (使用可能)] になります。



プライマリ DB クラスターが利用可能になったら、セカンダリクラスターを追加して Aurora Global Database を作成します。確認するには、「AWS リージョン の Amazon Aurora Global Database への追加」のステップに従います。

Aurora PostgreSQL を使用するグローバルデータベースの作成

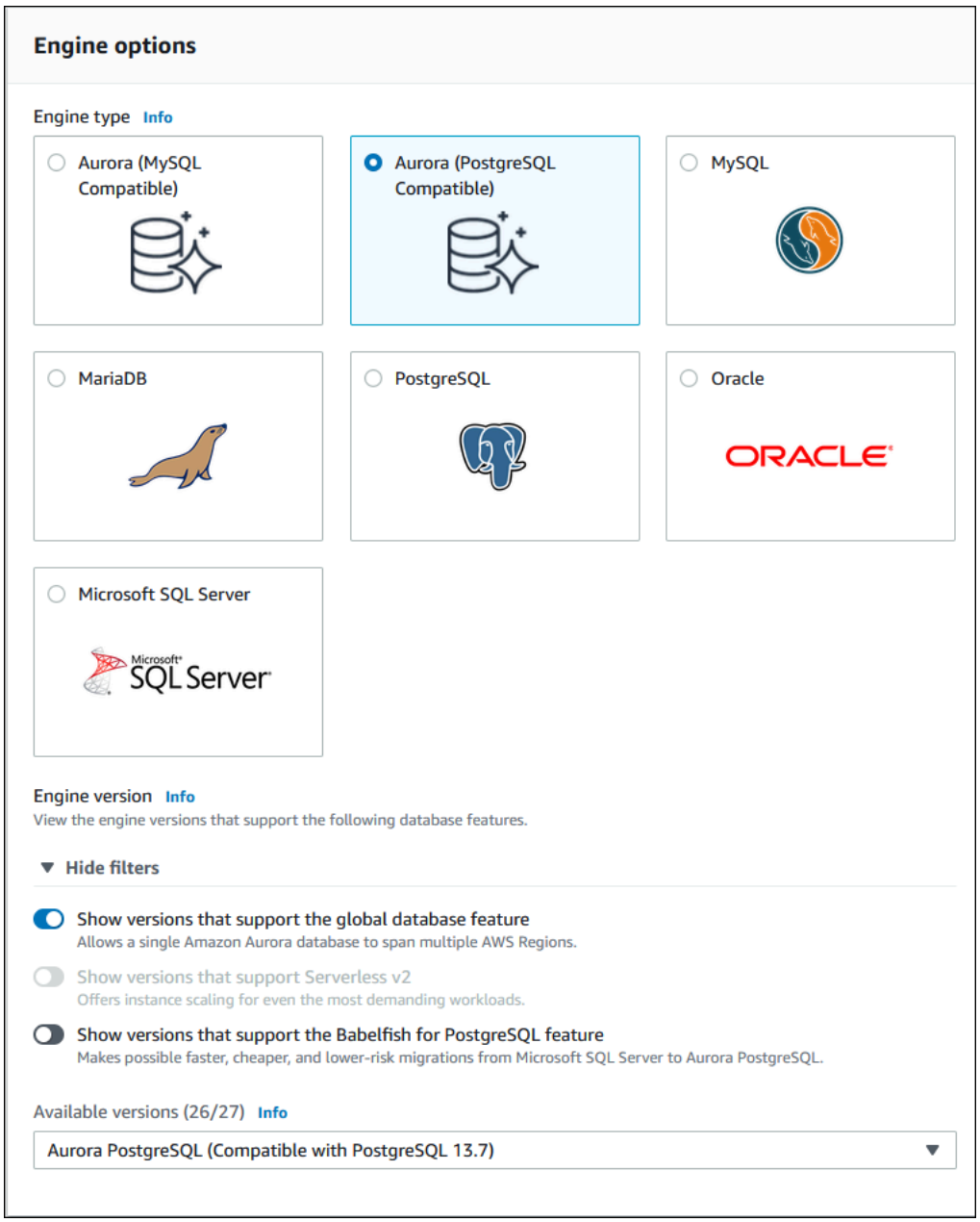
Aurora PostgreSQL を使用して Aurora グローバルデータベースを作成するには

[データベースの作成] ページニ記入します。

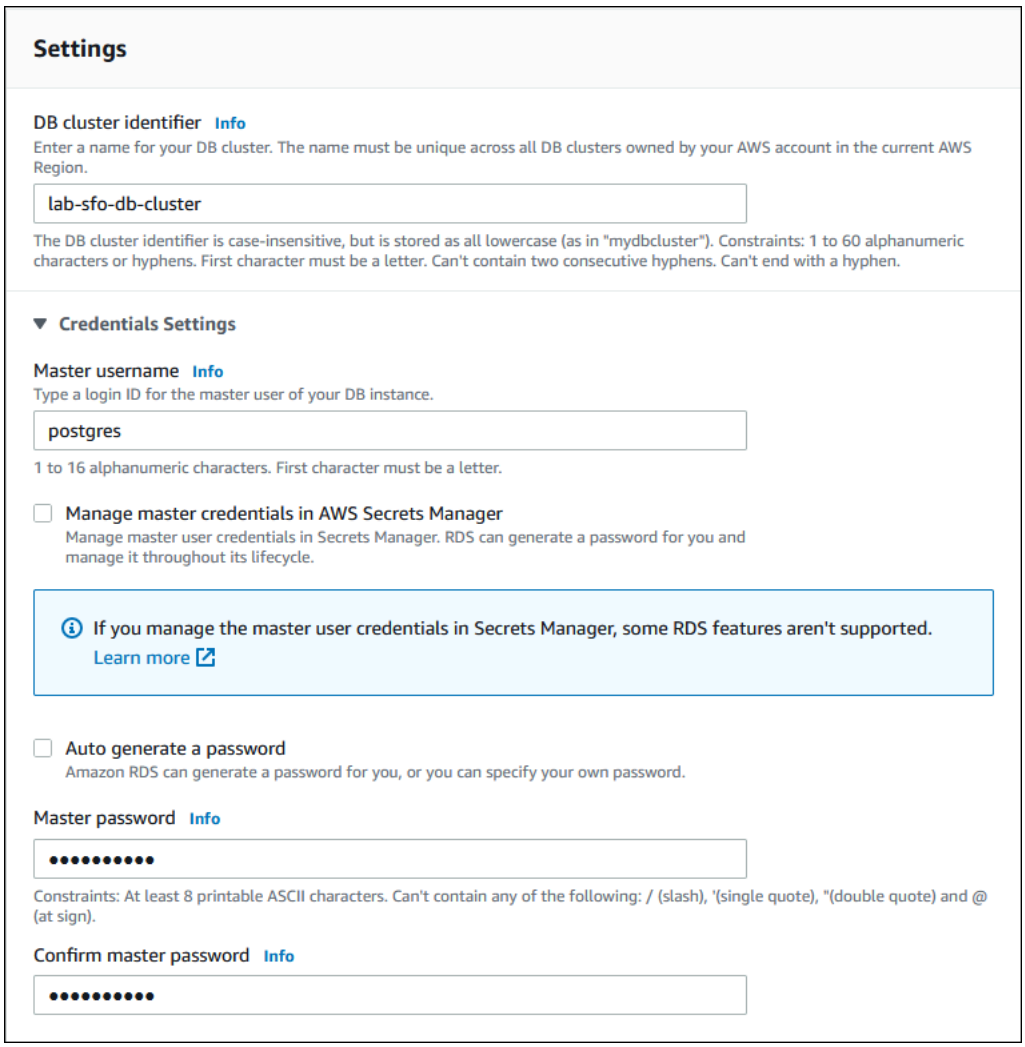
1. [Engine options (エンジンオプション)] で、以下の項目を選択します。

   a. [Show filters] (フィルターの表示) を展開し、[Show versions that support the global database feature] (グローバルデータベース機能をサポートするバージョンを表示する) をオンにします。

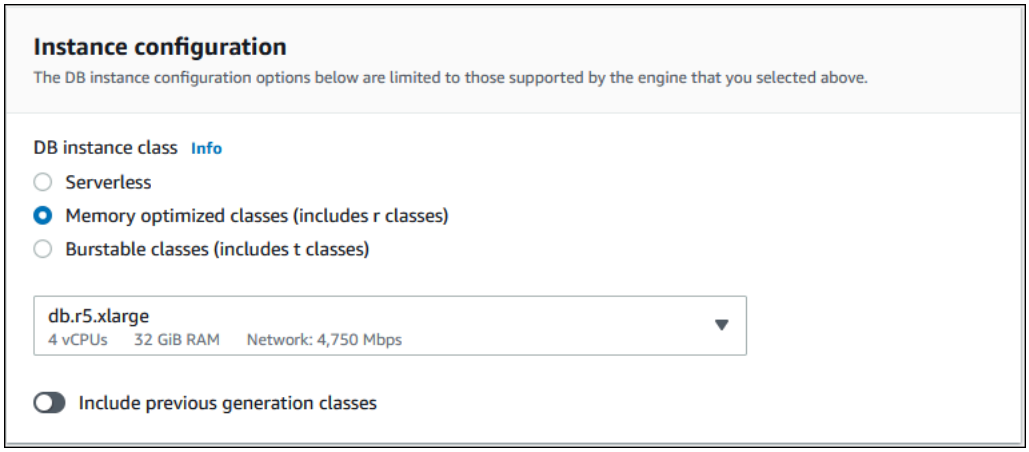
   b. [Engine version] (エンジンバージョン) では、Aurora Global Database に使用する Aurora PostgreSQL のバージョンを選択します。



2. [テンプレート] では、[本番稼働] を選択します。または、必要に応じて [Dev/Test (開発/テスト)] を選択することもできます。本番稼働環境では、開発/テストを使用しないでください。
3. [設定] では、以下の操作を行います。
   a. DB クラスター識別子にわかりやすい名前を入力します。Aurora Global Database の作成が完了すると、プライマリ DB クラスターはこの名前で識別されます。
   b. DB クラスターのデフォルト管理アカウント用に自分のパスワードを入力します。あるいは Aurora で生成します。[パスワードの自動生成] を選択すると、パスワードをコピーするオプションが表示されます。



4. [DB instance class (DB インスタンスクラス)] で、`db.r5.large` または別のメモリ最適化 DB インスタンスクラスを選択します。db.r5 以上のインスタンスクラスを使用することを推奨します。



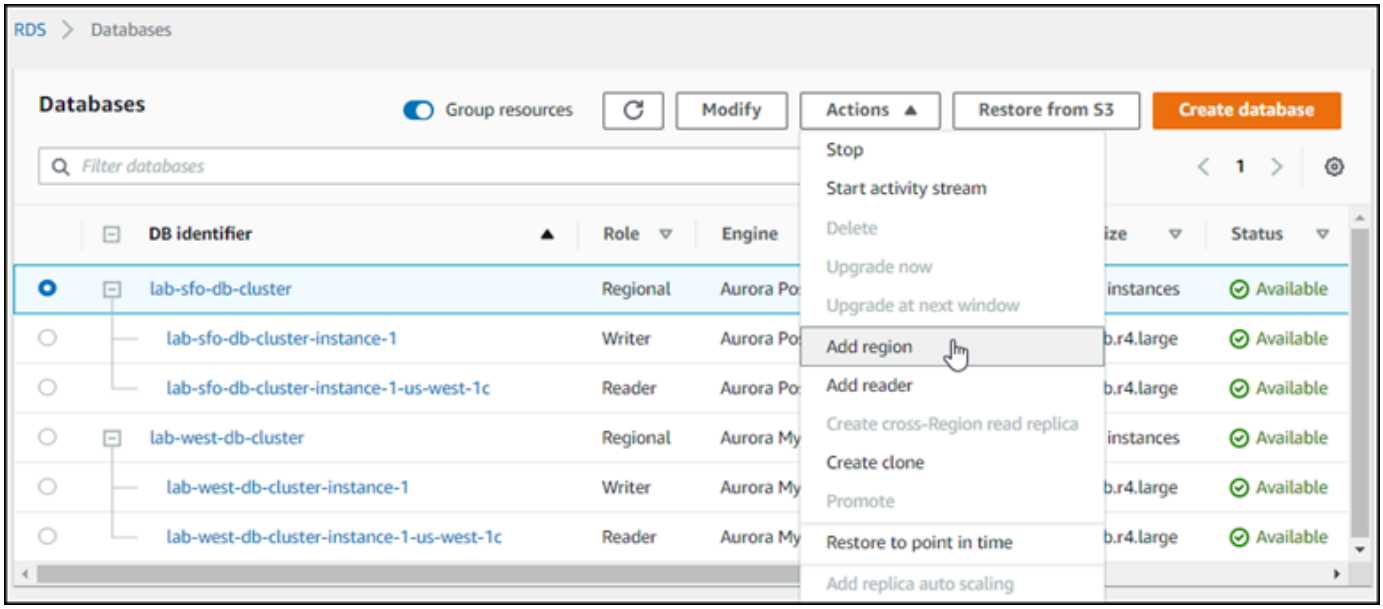
5. [可用性と耐久性の高い] では、Aurora に、Aurora レプリカを異なる AZ に作成させるように、選択することを推奨します。Aurora レプリカをすぐに作成しないときは、後で作成する必要があります。
6. [接続] では、この DB インスタンスの仮想ネットワーク環境を定義している Amazon VPC に基づいた、Virtual Private Cloud (VPC) を選択します。デフォルトを選択するとこのタスクを簡略化できます。
7. (オプション) [Database authentication] (データベース認証) の設定を完了します。パスワード認証は常に有効になっています。プロセスを簡略化するために、このセクションをスキップして、IAM パスワードと Kerberos 認証のセットアップを後で行うことができます。
8. [追加設定] では、以下の操作を行います。

   a. [初期のデータベース名] に名前を入力して、このクラスターのプライマリ Aurora DB インスタンスを作成します。これは、Aurora プライマリ DB クラスターの書き込みノードです。

      使用するカスタムパラメータグループが既にある場合を除いて、DB クラスターパラメータグループおよび DB パラメータグループは、デフォルトを選択したままにします。

   b. [Additional configuration] (追加設定) では、暗号化、ログのエクスポートなどのすべてで、デフォルトの設定を受け入れることができます。
9. [データベースの作成] を選択します。

   Aurora が Aurora DB インスタンス、Aurora レプリカ、Aurora DB クラスターの作成プロセスを完了するまで、数分かかることがあります。クラスターを使用する準備が整うと、Aurora DB クラスターと、書き込みおよびレプリカのノードは、すべて [Available] (利用可能) ステータスで表示されます。セカンダリを追加すると、これは Aurora Global Database のプライマリ DB クラスターになります。



プライマリ DB クラスターが作成されて利用可能になったら、「AWS リージョン の Amazon Aurora Global Database への追加」のステップに従ってセカンダリクラスターを 1 つまたは複数作成します。

AWS CLI

次の手順の AWS CLI コマンドで、次のタスクを実行します。

1. Aurora Global Database を作成し、名前を指定して、使用する Aurora データベースエンジンのタイプを指定します。
2. Aurora Global Database の Aurora DB クラスターを作成します。
3. クラスターの Aurora DB インスタンスを作成します。これは、グローバルデータベースのプライマリ Aurora DB クラスターです。
4. Aurora DB クラスターのセカンダリ DB インスタンスを作成します。これは、Aurora DB クラスターを完了するためのリーダーです。
5. 別のリージョンにセカンダリ Aurora DB クラスターを作成し、「AWS リージョン の Amazon Aurora Global Database への追加」のステップに従って Aurora Global Database に追加します。

Aurora データベースエンジンの手順に従います。

Aurora MySQL を使用するグローバルデータベースの作成

Aurora MySQL を使用して Aurora グローバルデータベースを作成するには

1. create-global-cluster CLI コマンドを使用して、AWS リージョン の名前、Aurora データベースエンジン、バージョンを渡します。

   Linux、macOS、Unix の場合:

   ```
   aws rds create-global-cluster --region primary_region  \
       --global-cluster-identifier global_database_id \
       --engine aurora-mysql \
       --engine-version version # optional
   ```

   Windows の場合:

   ```
   aws rds create-global-cluster ^
       --global-cluster-identifier global_database_id ^
       --engine aurora-mysql ^
       --engine-version version # optional
   ```

   これにより、名前 (識別子) と Aurora データベースエンジンだけの、「空の」Aurora Global Database が作成されます。Aurora Global Database が利用できるようになるまで、数分かかります。次のステップに進む前に、describe-global-clusters CLI コマンドを使用して利用可能かどうかを確認します。

   ```
   aws rds describe-global-clusters --region primary_region --global-cluster-
   identifier global_database_id
   ```

   Aurora Global Database が利用可能になったら、プライマリ Aurora DB クラスターを作成できます。

2. プライマリ Aurora DB クラスターを作成するときは、create-db-cluster CLI コマンドを使用します。`--global-cluster-identifier` パラメータを使用して、Aurora Global Database の名前を加えます。

   Linux、macOS、Unix の場合:

   ```
   aws rds create-db-cluster \
     --region primary_region \
   ```

```
  --db-cluster-identifier primary_db_cluster_id \
  --master-username userid \
  --master-user-password password \
  --engine aurora-mysql \
  --engine-version version \
  --global-cluster-identifier global_database_id
```

Windows の場合:

```
aws rds create-db-cluster ^
  --region primary_region ^
  --db-cluster-identifier primary_db_cluster_id ^
  --master-username userid ^
  --master-user-password password ^
  --engine aurora-mysql ^
  --engine-version version ^
  --global-cluster-identifier global_database_id
```

AWS CLI コマンドの describe-db-clusters を使用して、Aurora DB クラスターの準備が整ったことを確認します。特定の Aurora DB クラスターを選ぶときは、--db-cluster-identifier パラメータを使用します。あるいは、Aurora DB クラスターの名前をコマンドで除外すると、指定されたリージョンにあるすべての Aurora DB クラスターの詳細を取得できます。

```
aws rds describe-db-clusters --region primary_region --db-cluster-
identifier primary_db_cluster_id
```

レスポンスがクラスターに対して "Status": "available" と表示されたら利用できます。

3. プライマリ Aurora DB クラスターに DB インスタンスを作成します。これを行うには、create-db-instance CLI コマンドを使用します。コマンドに Aurora DB クラスターの名前を指定し、インスタンスの設定の詳細を指定します。コマンドは --master-username と --master-user-password のパラメータを Aurora DB クラスターから取得するので、それらをコマンドに渡す必要はありません。

   --db-instance-class では、db.r5.large などの、メモリ最適化クラスのもののみを使用できます。db.r5 以上のインスタンスクラスを使用することを推奨します。クラスの詳細については、DB インスタンスクラスを参照してください。

   Linux、macOS、Unix の場合:

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier db_instance_id \
  --engine aurora-mysql \
  --engine-version version \
  --region primary_region
```

Windows の場合:

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier db_instance_id ^
  --engine aurora-mysql ^
  --engine-version version ^
  --region primary_region
```

`create-db-instance` オペレーションが完了するまで時間がかかる場合があります。次に進む前に、ステータスをチェックして Aurora DB インスタンスが利用可能かどうかを確認します。

```
aws rds describe-db-clusters --db-cluster-identifier primary_db_cluster_id
```

コマンドが「利用可能」のステータスを返したら、プライマリ DB クラスターに新たな Aurora DB インスタンスを作成できます。これは、Aurora DB クラスターのリーダーインスタンス (Aurora レプリカ) です。

4. クラスターに別の Aurora DB インスタンスを作成するときは、`create-db-instance` CLI コマンドを使用します。

   Linux、macOS、Unix の場合:

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier replica_db_instance_id \
  --engine aurora-mysql
```

Windows の場合:

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier replica_db_instance_id ^
  --engine aurora-mysql
```

DB インスタンスが利用可能になると、書き込みノードからレプリカへのレプリケーションがスタートされます。次に進む前に、DB インスタンスが利用可能かどうかを describe-db-instances CLI コマンドを使用して確認します。

この時点で、書き込み DB インスタンスと Aurora レプリカを含むプライマリ Aurora DB クラスターを備えた、Aurora Global Database があります。ここで別のリージョンに読み取り専用の Aurora DB クラスターを追加し、Aurora Global Database を完成させます。これを行うには、「AWS リージョン の Amazon Aurora Global Database への追加」のステップに従います。

Aurora PostgreSQL を使用するグローバルデータベースの作成

次のコマンドを使用して Aurora Global Database の Aurora オブジェクトを作成する場合、各データベースが使用可能になるまでに数分かかる場合があります。任意のコマンドを完了した後、特定の Aurora オブジェクトのステータスをチェックして、ステータスが使用可能であることを確認します。

これを行うには、describe-global-clusters CLI コマンドを使用します。

```
aws rds describe-global-clusters --region primary_region
    --global-cluster-identifier global_database_id
```

Aurora PostgreSQL を使用して Aurora Global Database を作成するには

1. create-global-cluster CLI コマンドを使用します。

   Linux、macOS、Unix の場合:

   ```
   aws rds create-global-cluster --region primary_region  \
       --global-cluster-identifier global_database_id \
       --engine aurora-postgresql \
       --engine-version version # optional
   ```

Windows の場合:

```
aws rds create-global-cluster ^
    --global-cluster-identifier global_database_id ^
    --engine aurora-postgresql ^
    --engine-version version # optional
```

Aurora Global Database が利用可能になったら、プライマリ Aurora DB クラスターを作成できます。

2. プライマリ Aurora DB クラスターを作成するときは、`create-db-cluster` CLI コマンドを使用します。`--global-cluster-identifier` パラメータを使用して、Aurora Global Database の名前を加えます。

   Linux、macOS、Unix の場合:

```
aws rds create-db-cluster \
  --region primary_region \
  --db-cluster-identifier primary_db_cluster_id \
  --master-username userid \
  --master-user-password password \
  --engine aurora-postgresql \
  --engine-version version \
  --global-cluster-identifier global_database_id
```

   Windows の場合:

```
aws rds create-db-cluster ^
  --region primary_region ^
  --db-cluster-identifier primary_db_cluster_id ^
  --master-username userid ^
  --master-user-password password ^
  --engine aurora-postgresql ^
  --engine-version version ^
  --global-cluster-identifier global_database_id
```

   Aurora DB クラスターの準備が整っていることを確認します。以下のコマンドからのレスポンスが Aurora DB クラスターに対して `"Status": "available"` と表示されたら、次へ進めます。

```
aws rds describe-db-clusters --region primary_region --db-cluster-
identifier primary_db_cluster_id
```

3. プライマリ Aurora DB クラスターに DB インスタンスを作成します。これを行うには、create-db-instance CLI コマンドを使用します。

   Aurora DB クラスターの名前を --db-cluster-identifier パラメータに渡します。

   コマンドは --master-username と --master-user-password のパラメータを Aurora DB クラスターから取得するので、それらをコマンドに渡す必要はありません。

   --db-instance-class では、db.r5.large などの、メモリ最適化クラスのもののみを使用できます。db.r5 以上のインスタンスクラスを使用することを推奨します。クラスの詳細については、DB インスタンスクラスを参照してください。

   Linux、macOS、Unix の場合:

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier db_instance_id \
  --engine aurora-postgresql \
  --engine-version version \
  --region primary_region
```

   Windows の場合:

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier db_instance_id ^
  --engine aurora-postgresql ^
  --engine-version version ^
  --region primary_region
```

4. 次に進む前に、Aurora DB インスタンスのステータスを確認します。

```
aws rds describe-db-clusters --db-cluster-identifier primary_db_cluster_id
```

レスポンスで、Aurora DB インスタンスのステータスが「利用可能」と表示されていたら、プライマリ DB クラスターに新たな Aurora DB インスタンスを作成できます。

5. Aurora DB クラスターの Aurora レプリカを作成するときは、`create-db-instance` CLI コマンドを使用します。

   Linux、macOS、Unix の場合:

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier replica_db_instance_id \
  --engine aurora-postgresql
```

   Windows の場合:

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier replica_db_instance_id ^
  --engine aurora-postgresql
```

DB インスタンスが利用可能になると、書き込みノードからレプリカへのレプリケーションがスタートされます。次に進む前に、DB インスタンスが利用可能かどうかを `describe-db-instances` CLI コマンドを使用して確認します。

Aurora Global Database がありますが、その中にあるのは、書き込み DB インスタンスと Aurora レプリカで構成された Aurora DB クラスターを持つ、プライマリリージョンのみです。ここで別のリージョンに読み取り専用の Aurora DB クラスターを追加し、Aurora Global Database を完成させます。これを行うには、「AWS リージョン の Amazon Aurora Global Database への追加」のステップに従います。

RDS API

RDS API を使用して Aurora Global Database を作成するには、CreateGlobalCluster オペレーションを実行します。

AWS リージョン の Amazon Aurora Global Database への追加

Aurora Global Database には、プライマリ Aurora DB クラスターとは異なる AWS リージョン に 1 つ以上のセカンダリ Aurora DB クラスターが必要です。Aurora Global Database には、最大 5 つのセカンダリ DB クラスターをアタッチできます。Aurora Global Database に追加する各セカンダリ DB クラスターで、プライマリ DB クラスターに許容された Aurora レプリカの数を、1 つずつ減らします。

例えば、Aurora Global Database に 5 つのセカンダリリージョンがある場合、プライマリ DB クラスターが持てる Aurora レプリカの数は 15 ではなく 10 です。詳細については、「Amazon Aurora Global Database の構成要件」を参照してください。

プライマリ DB クラスター内の Aurora レプリカ (リーダーインスタンス) の数によって、追加できるセカンダリ DB クラスターの数が決まります。プライマリクラスターのリーダーインスタンスの数に、セカンダリクラスターの数を加えた合計が 15 を超えることはできません。例えば、プライマリクラスターにリーダーインスタンスが 14 あり、セカンダリクラスターが 1 つある場合、グローバルデータベースにこれ以上セカンダリクラスターを追加できません。

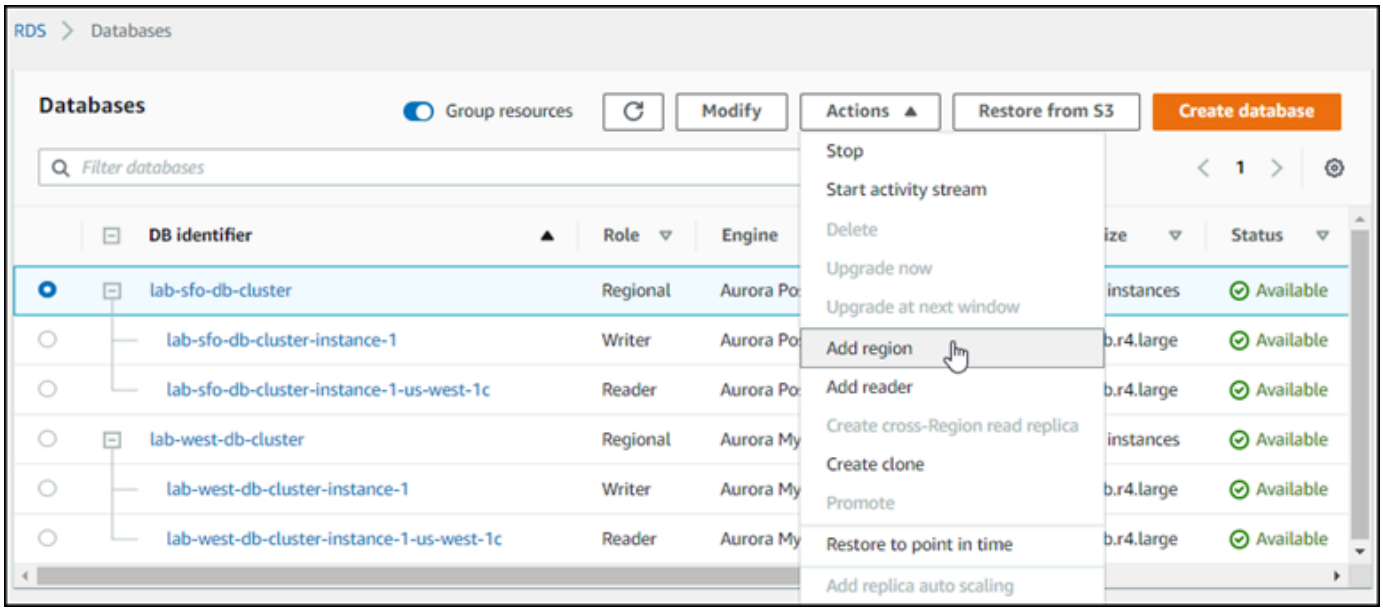
> **Note**
>
> Aurora MySQL バージョン 3 では、セカンダリクラスターを作成するときに、`lower_case_table_names` の値がプライマリクラスターの値と一致していることを確認します。この設定は、サーバーが識別子の大文字と小文字の区別を処理する方法に影響するデータベースパラメータです。データベースパラメータの詳細については、「「パラメータグループを使用する」」を参照してください。
> セカンダリクラスターを作成する際は、プライマリとセカンダリで同じ DB エンジンのバージョンを使用することをお勧めします。必要に応じて、プライマリをセカンダリと同じバージョンにアップグレードします。詳細については、「マネージドクロスリージョンスイッチオーバーまたはフェイルオーバーに対するパッチレベルの互換性」を参照してください。

コンソール

Aurora Global Database に AWS リージョン を追加するには

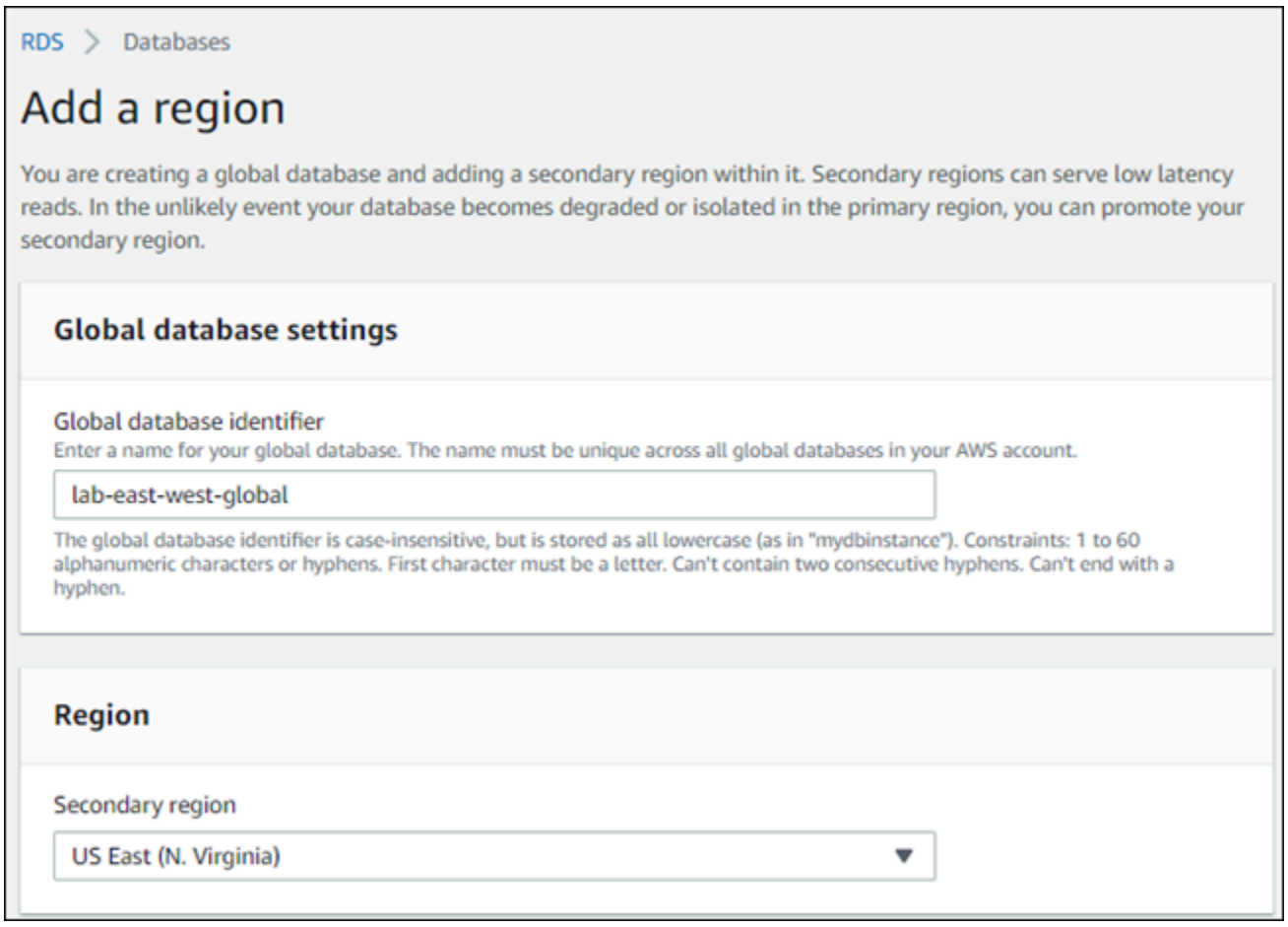
1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. AWS Management Console のナビゲーションペインで、[データベース] を選択します。

3. セカンダリ Aurora DB クラスターを必要とする Aurora Global Database を選択します。プライマリ Aurora DB クラスターが `Available` であることを確認します。
4. [アクション] で、[リージョンの追加] を選択します。



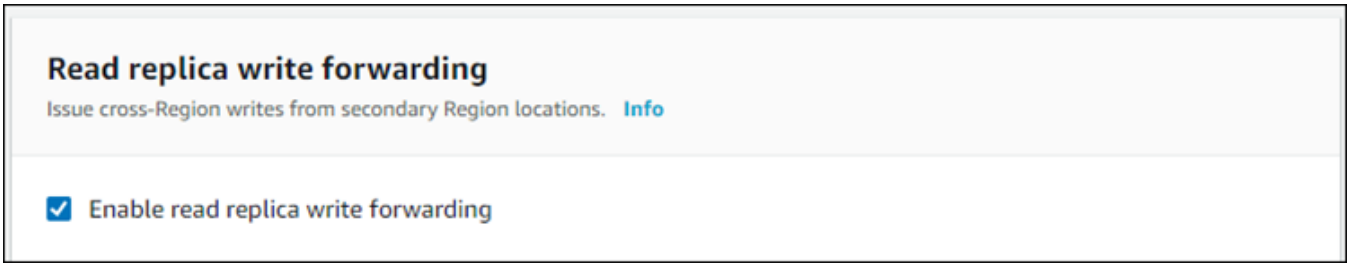
5. [Add a region] (リージョンの追加) ページで、セカンダリ AWS リージョン を選択します。

   同じ Aurora Global Database では、既にセカンダリ Aurora DB クラスターが存在する AWS リージョン を選択することはできません。また、プライマリ Aurora DB クラスターと同じリージョンにすることはできません。



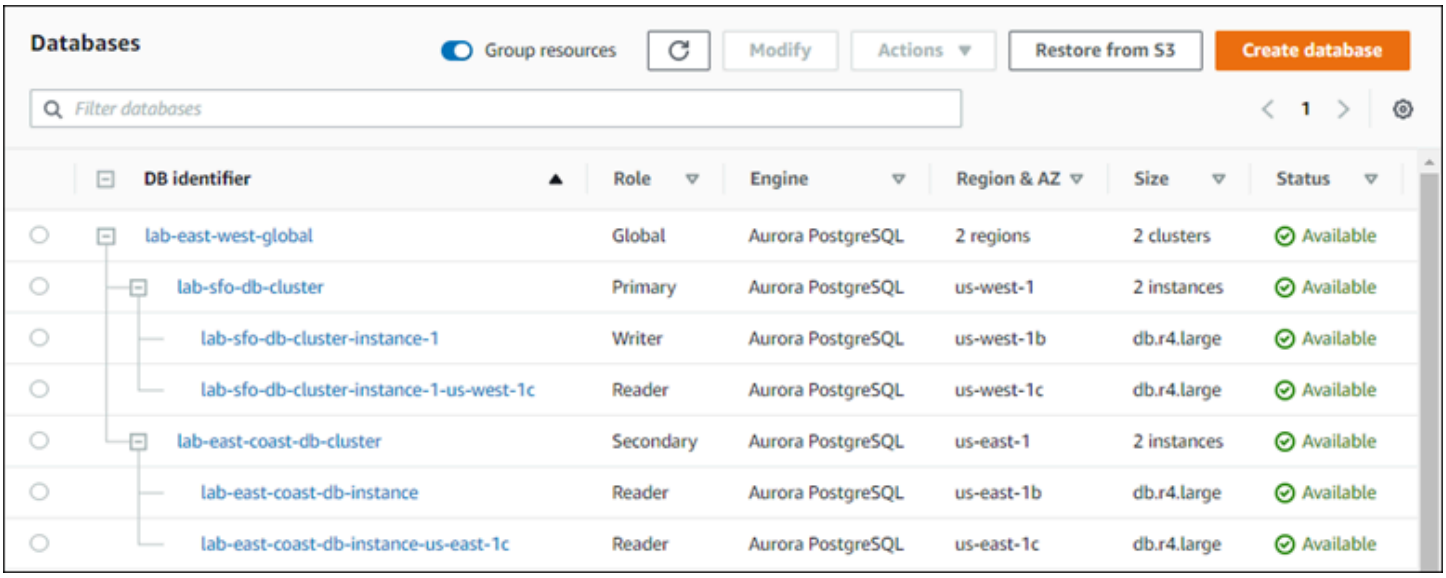
6. 新しい AWS リージョンのセカンダリ Aurora クラスターの残りのフィールドに入力してください。これらは、Aurora DB クラスターインスタンスと同じ設定オプションですが、以下のオプションは Aurora MySQL - ベースの Aurora Global Database にのみ適用されます。

   - リードレプリカの書き込み転送を有効にする - このオプションの設定では、Aurora Global Database のセカンダリ DB クラスターが書き込み操作をプライマリクラスターに転送します。詳細については、「Amazon Aurora Global Database の書き込み転送を使用する」を参照してください。



7. [リージョンの追加] を選択します。

Aurora Global Database にリージョンを追加すると、スクリーンショットに示すように、それらのリージョンが AWS Management Console の[データベース] に一覧表示されます。



AWS CLI

セカンダリ AWS リージョン を Aurora Global Database に追加するには

1. Aurora Global Database の名前 (`create-db-cluster`) を指定して、`--global-cluster-identifier` CLI コマンドを使用します。他のパラメータでは、以下の操作を行います。
2. `--region` で、Aurora プライマリリージョンのものとは異なる AWS リージョン を選択します。
3. `--engine` および `--engine-version` パラメータの特定の値を選択します。これらの値は、Aurora Global Database のプライマリ Aurora DB クラスターの値と同じです。
4. 暗号化されたクラスターの場合は、暗号化の `--source-region` としてプライマリ AWS リージョン を指定します。

次の例では、新しい Aurora DB クラスターを作成し、それを読み取り専用のセカンダリ Aurora DB クラスターとして Aurora Global Database にアタッチします。最後のステップでは、Aurora DB インスタンスを新しい Aurora DB クラスターに追加します。

Linux、macOS、Unix の場合:

```
aws rds --region secondary_region \
  create-db-cluster \
    --db-cluster-identifier secondary_cluster_id \
```

```
    --global-cluster-identifier global_database_id \
    --engine aurora-mysql|aurora-postgresql
    --engine-version version

aws rds --region secondary_region \
  create-db-instance \
    --db-instance-class instance_class \
    --db-cluster-identifier secondary_cluster_id \
    --db-instance-identifier db_instance_id \
    --engine aurora-mysql|aurora-postgresql
```

Windows の場合:

```
aws rds --region secondary_region ^
  create-db-cluster ^
    --db-cluster-identifier secondary_cluster_id ^
    --global-cluster-identifier global_database_id_id ^
    --engine aurora-mysql|aurora-postgresql ^
    --engine-version version

aws rds --region secondary_region ^
  create-db-instance ^
    --db-instance-class instance_class ^
    --db-cluster-identifier secondary_cluster_id ^
    --db-instance-identifier db_instance_id ^
    --engine aurora-mysql|aurora-postgresql
```

RDS API

RDS API を使用して新しい AWS リージョン を Aurora Global Database に追加するには、CreateDBCluster オペレーションを実行します。GlobalClusterIdentifier パラメータを使用して、既存のグローバルデータベースの識別子を指定します。

セカンダリリージョンでのヘッドレス Aurora DB クラスターの作成

Aurora Global Database では、プライマリとは異なる AWS リージョン に少なくとも 1 つのセカンダリ Aurora DB クラスターが必要でが、セカンダリクラスターにはヘッドレス設定を使用できます。ヘッドレスセカンダリ Aurora DB クラスターは、DB インスタンスがないクラスターのことです。この種類の構成では、Aurora Global Database の費用を削減できます。Aurora DB クラスターでは、コンピューティングとストレージは分離されています。DB インスタンスがない場合、コンピューティングに対する課金は発生せず、ストレージに対してのみが課金されます。正しく設定され

ていれば、ヘッドレスセカンダリのストレージボリュームはプライマリ Aurora DB クラスターと同期したままになります。

セカンダリクラスターを追加して、通常、Aurora Global Database の作成時に行うようにします。ただし、プライマリ Aurora DB クラスターがセカンダリへのレプリケーションをスタートした後、セカンダリ Aurora DB クラスターから Aurora 読み取り専用 DB を削除します。DB インスタンスがなくなったため、このセカンダリクラスターは「ヘッドレス」とみなされます。ただし、ストレージボリュームはプライマリ Aurora DB クラスターと同期されます。

> **Warning**
>
> Aurora PostgreSQL を使用して、セカンダリ AWS リージョン にヘッドレスクラスターを作成するには、AWS CLI または RDS API を使用してセカンダリ AWS リージョン を追加します。ステップをスキップして、セカンダリクラスターのリーダー DB インスタンスを作成します。現在、RDS コンソールでは、ヘッドレスクラスターの作成はサポートされていません。使用する CLI および API の手順については、AWS リージョン の Amazon Aurora Global Database への追加 を参照してください。
> グローバルデータベースが 13.4、12.8、または 11.13 未満のエンジンバージョンを使用している場合、セカンダリリージョンにリーダー DB インスタンスを作成し、その後、それを削除すると、プライマリリージョンのライター DB インスタンスで Aurora PostgreSQL バキューム問題が発生する可能性があります。この問題が発生した場合は、セカンダリリージョンのリーダー DB インスタンスを削除した後に、プライマリリージョンのライター DB インスタンスを再起動します。

ヘッドレスセカンダリ Aurora DB クラスターを Aurora Global Database に追加するには

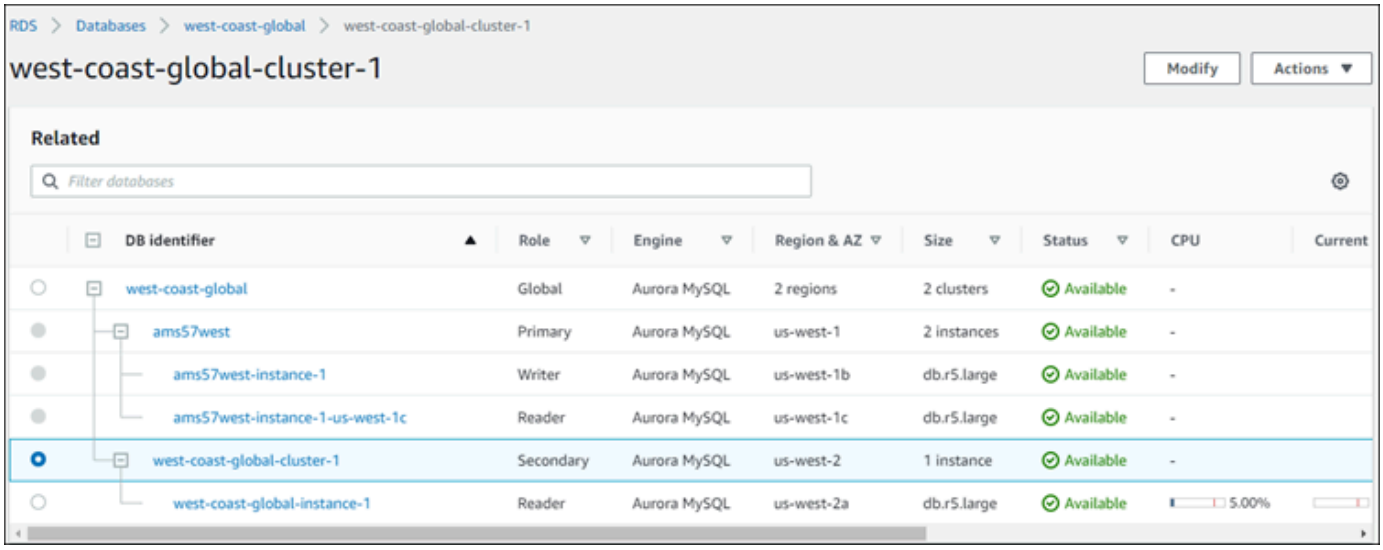
1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. AWS Management Console のナビゲーションペインで、[データベース] を選択します。
3. セカンダリ Aurora DB クラスターを必要とする Aurora Global Database を選択します。プライマリ Aurora DB クラスターが `Available` であることを確認します。
4. [アクション] で、[リージョンの追加] を選択します。
5. [Add a region] (リージョンの追加) ページで、セカンダリ AWS リージョン を選択します。

同じ Aurora Global Database では、既にセカンダリ Aurora DB クラスターが存在する AWS リージョン を選択することはできません。また、プライマリ Aurora DB クラスターと同じリージョンにすることはできません。

6. 新しい AWS リージョン リージョンのセカンダリ Aurora クラスターの残りのフィールドに必要事項を入力してください。これらは、任意の Aurora DB クラスターインスタンスと同じ設定オプションです。

   Aurora MySQL - ベースの Aurora Global Database の場合、[Enable read replica write forwarding (リードレプリカの書き込み転送を有効にする)] オプションを無視します。リーダーインスタンスを削除した後は、このオプションは機能しません。

7. [リージョンの追加] を選択します。Aurora Global Database にリージョンを追加すると、スクリーンショットに示すように、それらのリージョンが AWS Management Console の[データベース] に一覧表示されます。

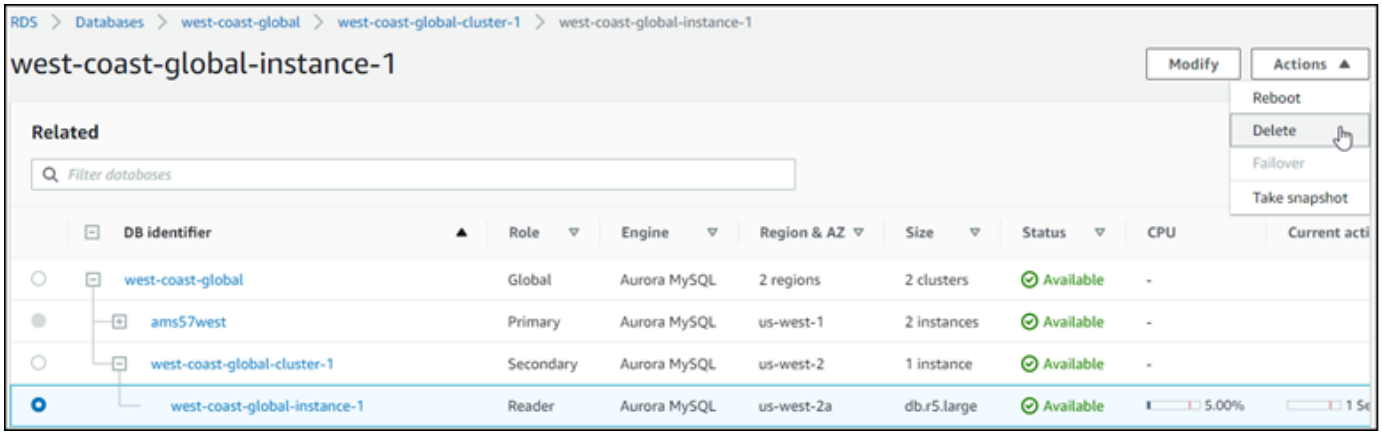


8. 続行する前に、AWS Management Console または AWS CLI を使用して、セカンダリ Aurora DB クラスターと、そのリーダーインスタンスのステータスを確認します。次に例を示します。

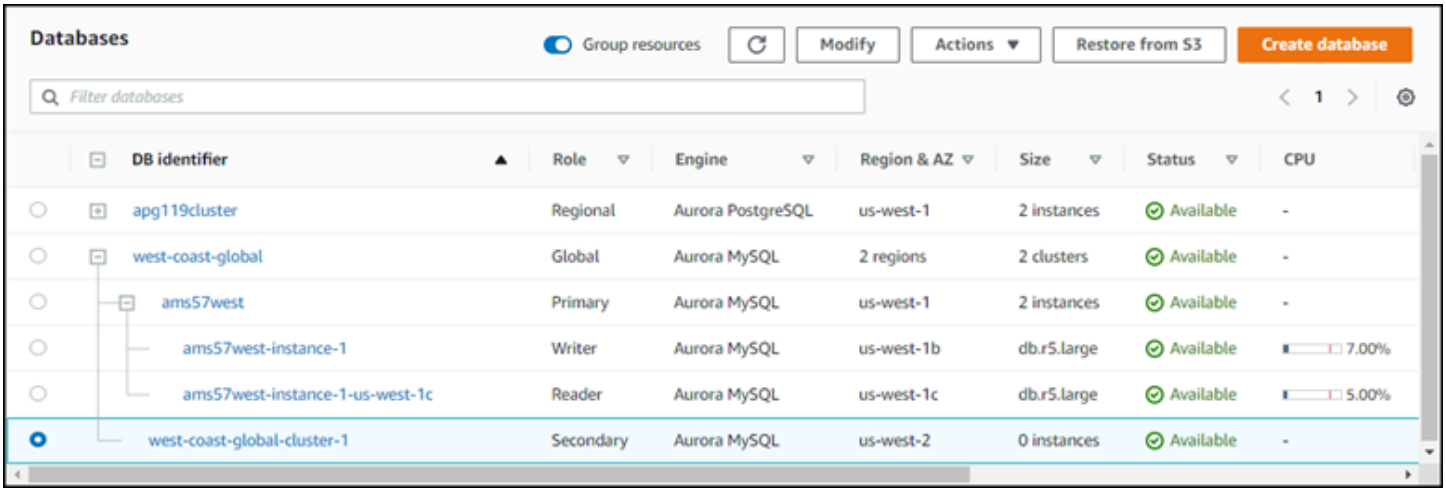
```
$ aws rds describe-db-clusters --db-cluster-identifier secondary-cluster-id --query
 '*[].[Status]' --output text
```

   新しく追加されたセカンダリ Aurora DB クラスターのステータスが [creating] から [available] に変わるまでに数分かかる場合があります。Aurora DB クラスターが使用可能になったら、リーダーインスタンスを削除できます。

9. セカンダリ Aurora DB クラスターでリーダーインスタンスを選択し、[Delete] (削除) をクリックします。



リーダーインスタンスを削除しても、セカンダリクラスターは Aurora Global Database の一部にとどまります。以下に示すように、インスタンスに関連付けられていません。



このヘッドレスセカンダリ Aurora DB クラスターを使用して、プライマリ AWS リージョン の予期しない停止が起きても Amazon Aurora Global Database を手動で復旧できます。

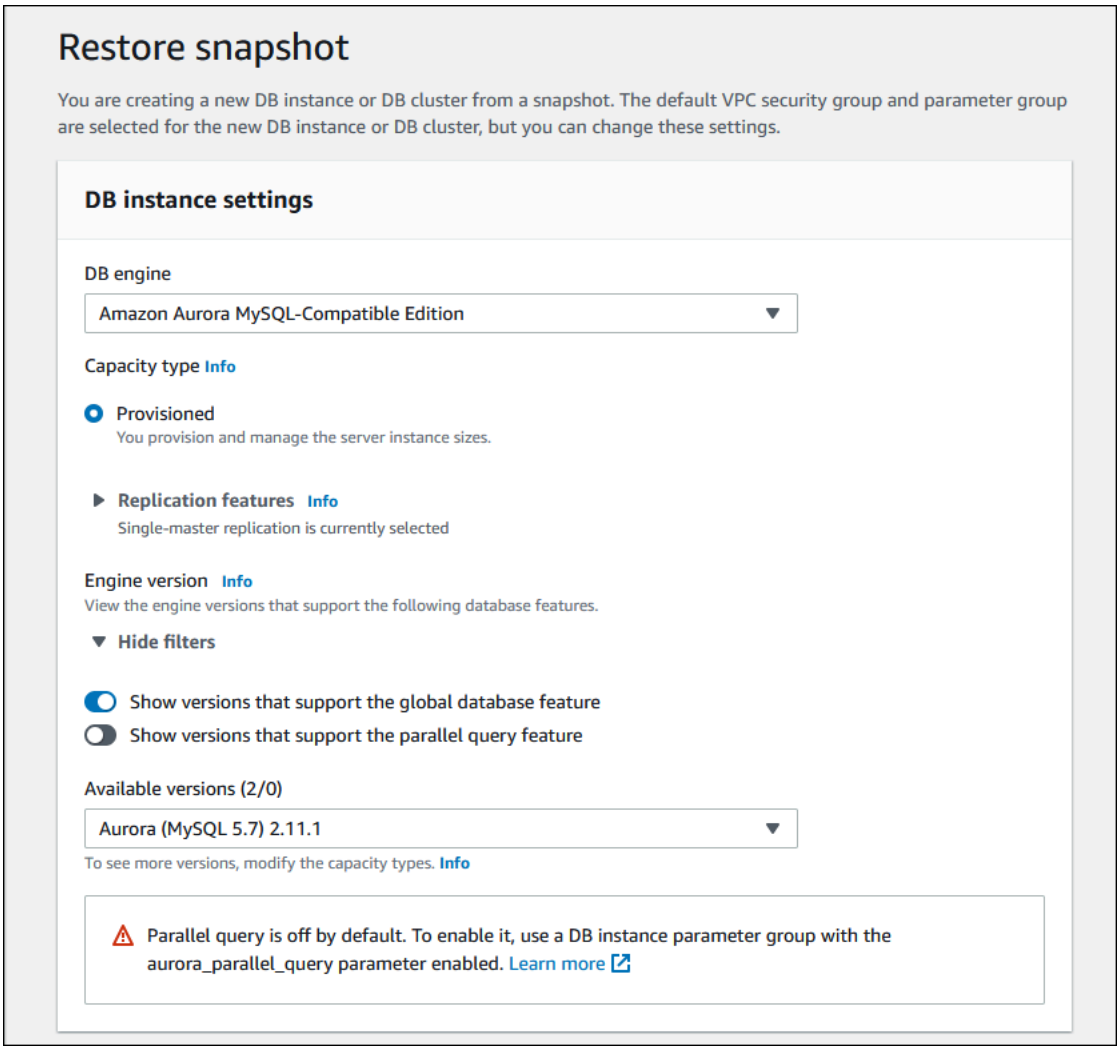
Amazon Aurora Global Database のスナップショットの使用

Aurora DB クラスターのスナップショットを復元するか、Amazon RDS DB インスタンスからスナップショットを復元することで、Aurora Global Database のスタート点として使用することができます。スナップショットを復元し、新しい Aurora プロビジョンド DB クラスターを同時に作成します。次に、復元した DB クラスターに別の AWS リージョン を追加して、それを Aurora Global Database に変換します。この方法でスナップショットを使って作成された Aurora DB クラスターは、すべて Aurora Global Database のプライマリクラスターになります。

使用するスナップショットは、`provisioned` か `serverless`Aurora DB クラスターから取得できます。

復元プロセス中は、スナップショットと同じ DB エンジンタイプを選択します。例えば、Aurora PostgreSQL を実行している Aurora Serverless DB クラスターから作成されたスナップショットを復元するとします。この場合、同じ Aurora DB エンジンとバージョンを使用して Aurora PostgreSQL DB クラスターを作成します。

復元された DB クラスターは、AWS リージョン が追加される際に、Aurora Global Database のプライマリクラスターのロールを引き継ぎます。このプライマリクラスターに含まれるすべてのデータは、Aurora Global Database に追加したセカンダリクラスターにレプリケートされます。



Amazon Aurora Global Database の管理

Aurora グローバルデータベースを構成する個別のクラスターに対して、ほとんどの管理オペレーションを実行できます。コンソールの [Databases (データベース)] ページで [Group related resources (グループ関連のリソース)] を選択すると、プライマリクラスターとセカンダリクラスターが、関連付けられたグローバルデータベースの下にグループとして表示されます。グローバルデータ

ベースの DB クラスターが実行されている AWS リージョン、Aurora DB エンジンとバージョン、およびその識別子を検索するには、[Configuration] (設定) タブを使用します。

クロスリージョンデータベースのフェイルオーバープロセスは、単一の Aurora DB クラスターではなく、Aurora グローバルデータベースでのみ使用できます。詳細については、「Amazon Aurora Global Database でスイッチオーバーまたはフェイルオーバーを使用する」を参照してください。

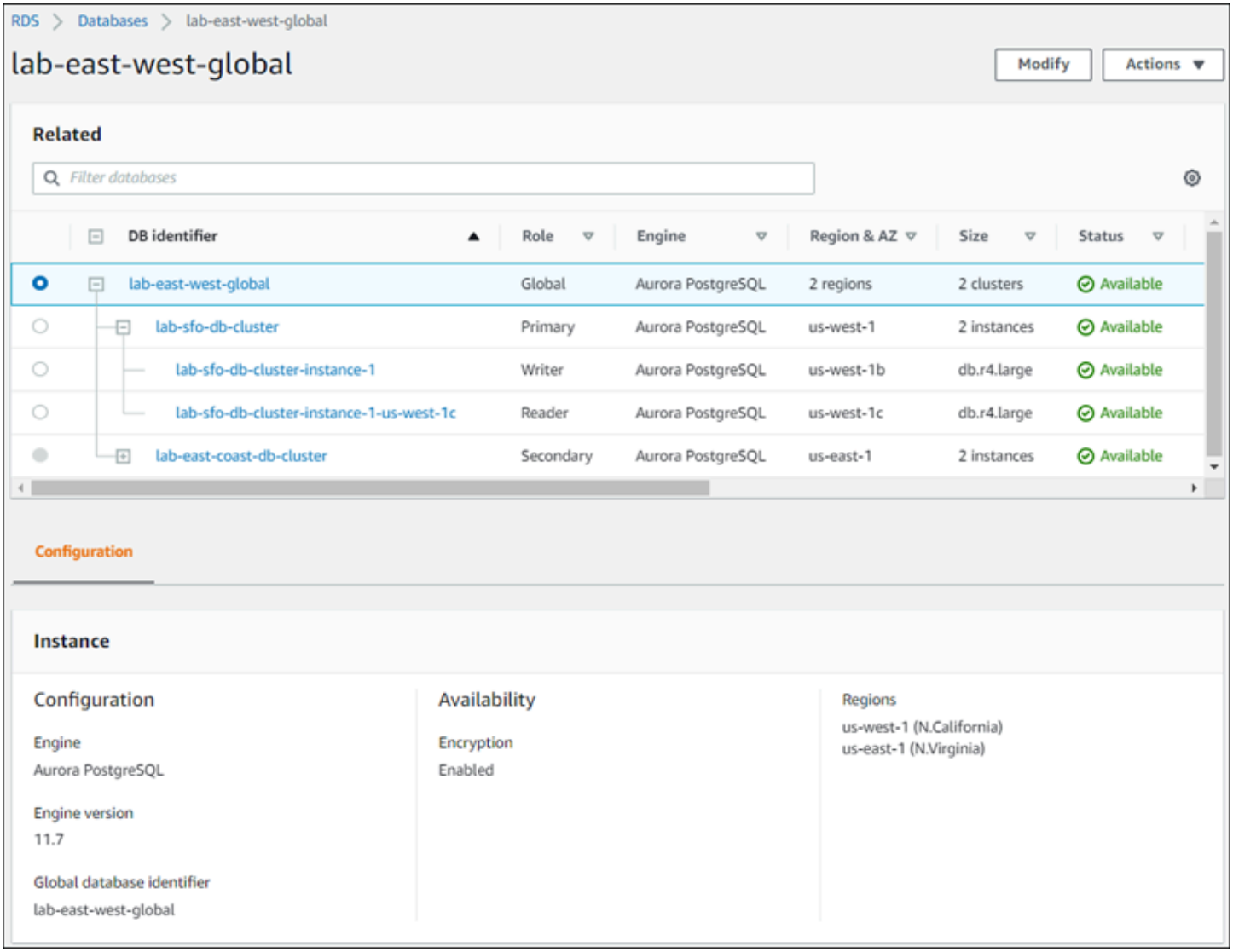
プライマリリージョンの予期しない停止から Aurora のグローバルデータベースを復元するには、予期しない停止からの Amazon Aurora Global Database の復旧 を参照してください。

トピック

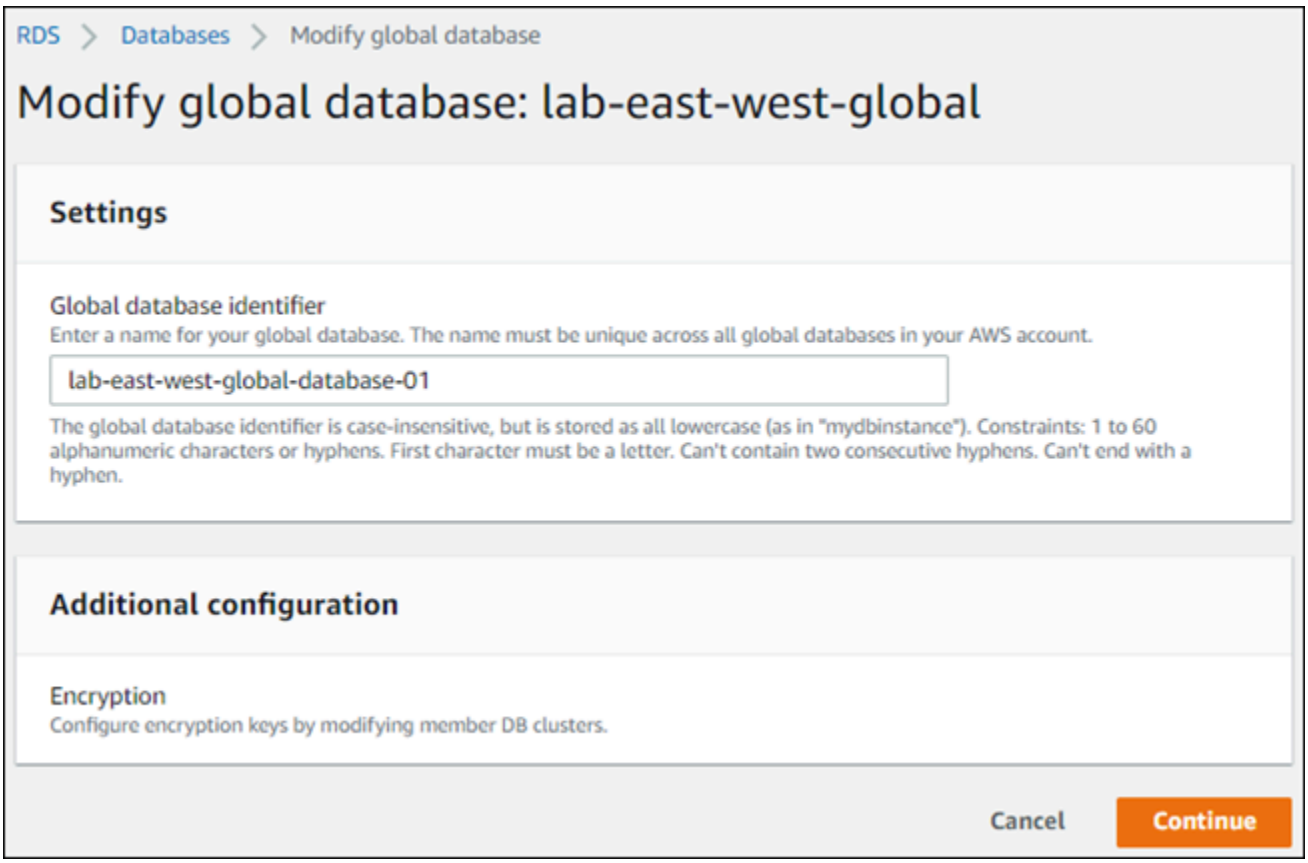
- Amazon Aurora Global Database の修正
- Aurora Global Database のパラメータの修正
- Amazon Aurora Global Database からのクラスターの削除
- Amazon Aurora Global Database の削除

Amazon Aurora Global Database の修正

AWS Management Console の [データベース] ページには、すべての Aurora Global Database が、それぞれのプライマリクラスターおよびセカンダリクラスターとともに表示されます。Aurora Global Database には、独自の構成設定があります。具体的には、次のスクリーンショットに示すように、プライマリクラスターとセカンダリクラスターに関連付けられた AWS リージョン があります。



Aurora Global Database に変更を加えると、次のスクリーンショットのように、変更をキャンセルする機会が与えられます。



[続行] を選択したら、変更を確認します。

Aurora Global Database のパラメータの修正

Aurora Global Database 内の各 Aurora クラスターで、Aurora DB クラスターパラメータグループを個別に設定することができます。ほとんどのパラメータの動作は、他の種類の Aurora クラスターと同じです。グローバルデータベース内のすべてのクラスター間で設定の一貫性を保つことをお勧めします。これにより、セカンダリクラスターをプライマリクラスターに昇格した場合に、予期しない動作の変更が生じることを回避できます。

例えば、別のクラスターがプライマリクラスターを肩代わりしたときに動作が変わらないように、タイムゾーンと文字セットに同じ設定を使用します。

`aurora_enable_repl_bin_log_filtering` と `aurora_enable_replica_log_compression` の構成設定は何の効果もありません。

Amazon Aurora Global Database からのクラスターの削除

Aurora DB クラスターは、いくつかの理由により、Aurora Global Database から削除できます。例えば、プライマリクラスターの性能低下や切断が生じて、Aurora DB クラスターを Aurora Global

Database から削除したい場合。その後、新しい Aurora Global Database を作成するために使用できるスタンドアロンのプロビジョニング Aurora DB クラスターになります。詳細については、「予期しない停止からの Amazon Aurora Global Database の復旧」を参照してください。

また、不要になった Aurora Global Database を削除するために、Aurora DB クラスターを削除したい場合も可能です。関連するすべての Aurora DB クラスターを削除 (デタッチ) した後、プライマリが最後に残るまで、Aurora Global Database を削除することはできません。詳細については、「Amazon Aurora Global Database の削除」を参照してください。

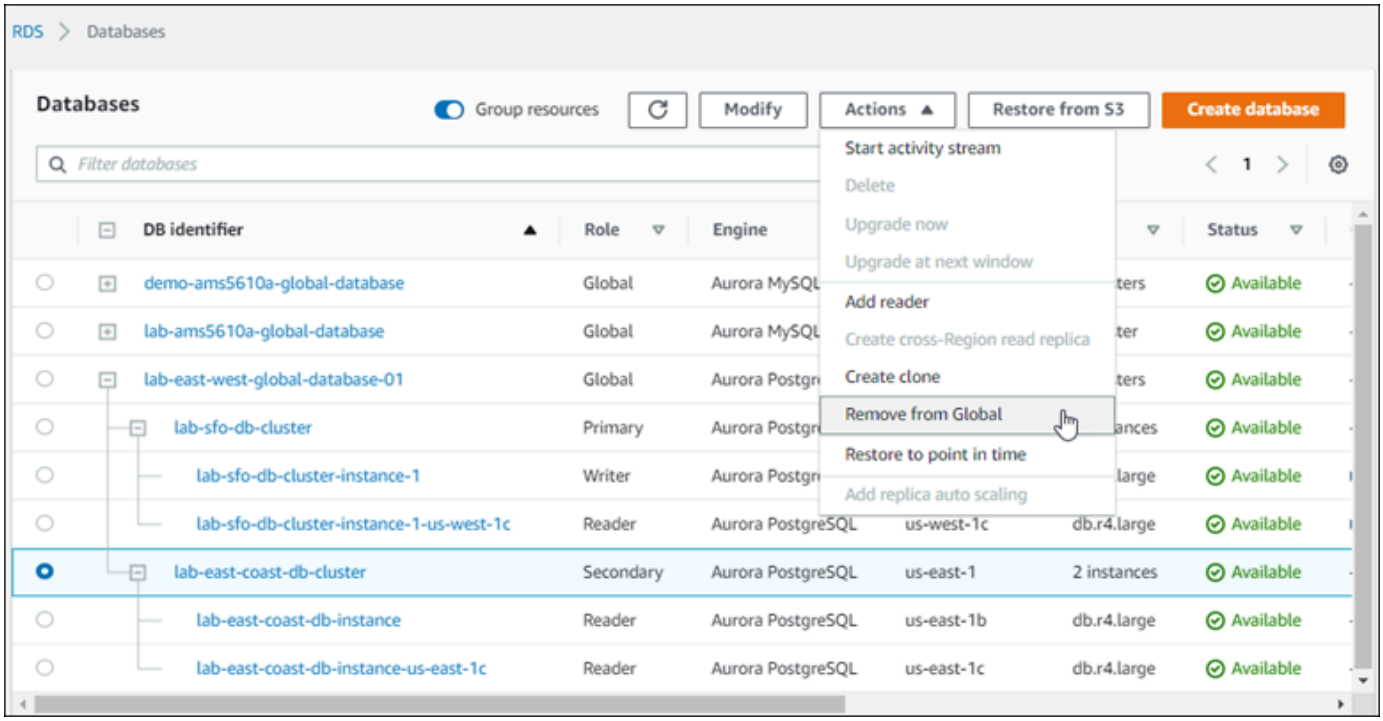
Aurora DB クラスターが Aurora Global Database からデタッチされると、プライマリデータベースと同期されなくなります。これは、完全な読み取り/書き込み機能を備えたスタンドアロンのプロビジョンド Aurora DB クラスターになります。

AWS Management Console、AWS CLI、または RDS API を使用すると、Aurora DB クラスターを Aurora Global Database から削除できます。

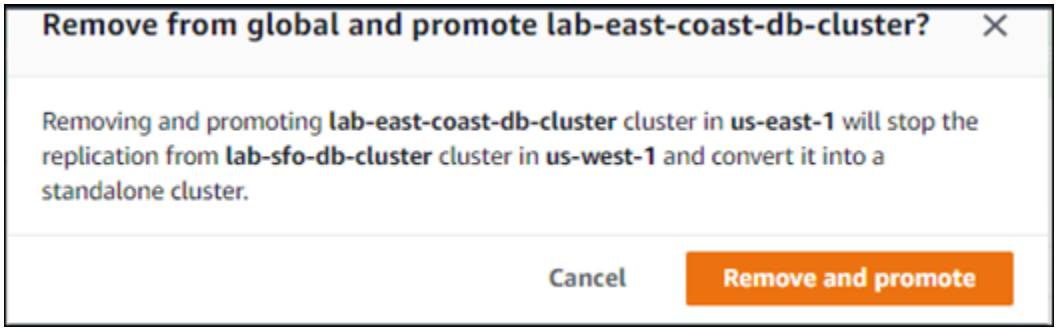
コンソール

Aurora Global Database から Aurora クラスターを解除するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. [データベース] ページでクラスターを選択します。
3. [アクション] で [グローバルから削除] を選択します。

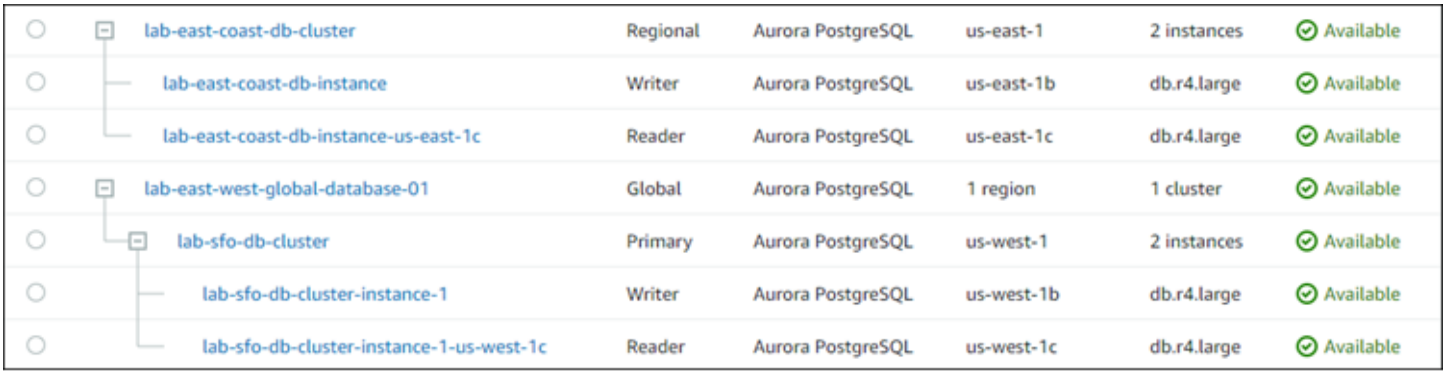


Aurora Global Database からセカンダリをデタッチするかどうかを確認するプロンプトが表示されます。



4. グローバルデータベースからクラスターを削除するには、[削除して昇格する] を選択します。

Aurora DB クラスターは、Aurora Global Database のセカンダリとして機能しなくなり、プライマリ DB クラスターと同期しなくなります。これは、完全な読み取り/書き込み機能を備えたスタンドアロン Aurora DB クラスターです。



すべてのセカンダリクラスターを解除または削除したら、次にプライマリクラスターを同じ方法で解除できます。すべてのセカンダリクラスターを削除するまで、Aurora Global Database からプライマリ Aurora DB クラスターをデタッチ (削除) することはできません。

Aurora Global Database は、リージョンと AZ が 0 のまま [Databases (データベース)] リストに残り続ける場合があります。今後この Aurora Global Database を使用しない場合は、削除できます。詳細については、「Amazon Aurora Global Database の削除」を参照してください。

AWS CLI

Aurora Global Database から Aurora クラスターを解除するには、次のパラメータを指定して、remove-from-global-cluster CLI コマンドを実行します。

- `--global-cluster-identifier`-Aurora Global Database の名前 (識別子)。
- `--db-cluster-identifier`-Aurora Global Database から削除する各 Aurora DB クラスターの名前。プライマリを削除する前に、すべてのセカンダリ Aurora DB クラスターを削除します。

以下の例では、まずセカンダリクラスターを削除し、次に Aurora Global Database からプライマリクラスターを削除します。

Linux、macOS、Unix の場合:

```
aws rds --region secondary_region \
  remove-from-global-cluster \
    --db-cluster-identifier secondary_cluster_ARN \
    --global-cluster-identifier global_database_id

aws rds --region primary_region \
  remove-from-global-cluster \
    --db-cluster-identifier primary_cluster_ARN \
```

```
    --global-cluster-identifier global_database_id
```

Aurora Global Database のセカンダリ AWS リージョン ごとに、`remove-from-global-cluster --db-cluster-identifier` *secondary_cluster_ARN* コマンドを繰り返し実行します。

Windows の場合:

```
aws rds --region secondary_region ^
  remove-from-global-cluster ^
    --db-cluster-identifier secondary_cluster_ARN ^
    --global-cluster-identifier global_database_id

aws rds --region primary_region ^
  remove-from-global-cluster ^
    --db-cluster-identifier primary_cluster_ARN ^
    --global-cluster-identifier global_database_id
```

Aurora Global Database のセカンダリ AWS リージョン ごとに、`remove-from-global-cluster --db-cluster-identifier` *secondary_cluster_ARN* コマンドを繰り返し実行します。

RDS API

RDS API を使用して Aurora Global Database から Aurora クラスターを解除するには、RemoveFromGlobalCluster アクションを実行します。

Amazon Aurora Global Database の削除

通常、Aurora Global Database にはビジネスクリティカルなデータが含まれているため、グローバルデータベースおよび関連付けられているクラスターを 1 回のステップで削除することはできません。Aurora Global Database を削除するには、以下の操作を行います。

- Aurora Global Database からすべてのセカンダリ DB クラスターを削除します。各クラスターは、スタンドアロン Aurora DB クラスターになります。この方法については、「Amazon Aurora Global Database からのクラスターの削除」を参照してください。
- 各スタンドアロン Aurora DB クラスターから、すべての Aurora レプリカを削除します。
- Aurora Global Database からプライマリ DB クラスターを削除します。これは、スタンドアロン Aurora DB クラスターになります。
- Aurora プライマリ DB クラスターから、まずすべての Aurora レプリカを削除し、次に書き込み DB インスタンスを削除します。

新しいスタンドアロン Aurora DB クラスターからライターインスタンスを削除すると、通常、Aurora DB クラスターと Aurora Global Database も削除されます。

全般的な情報については、「Aurora DB クラスターからの DB インスタンスの削除」を参照してください

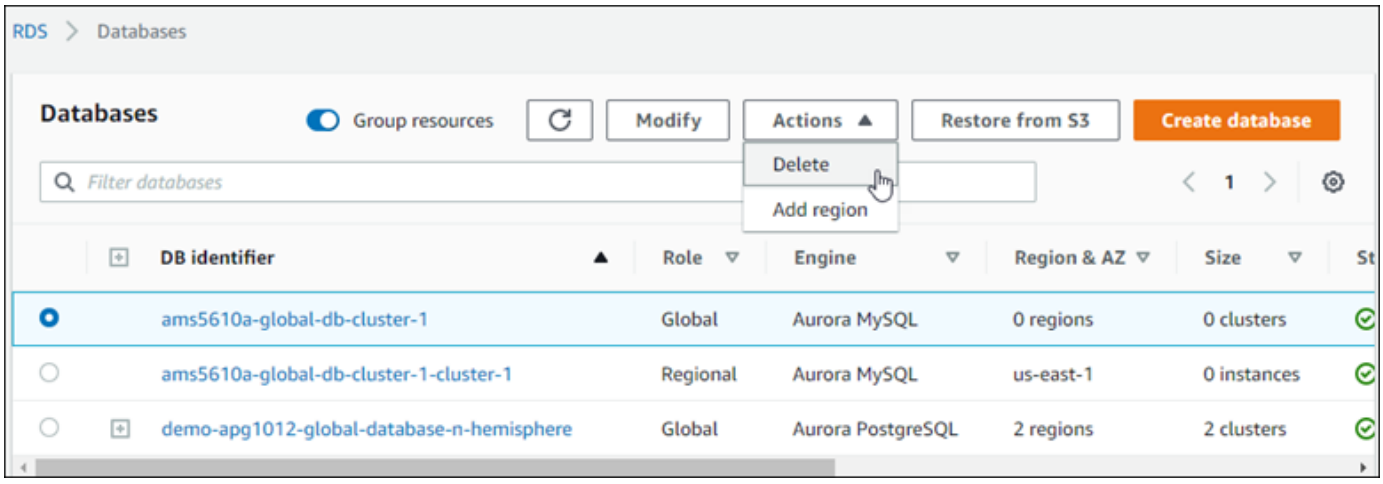
Aurora Global Database を削除するには AWS Management Console、AWS CLI、または RDS API を使用します。

コンソール

Aurora Global Database を削除するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. [データベース] を選択し、リストから削除する Aurora Global Database を探します。
3. Aurora Global Database からすべてのクラスターが削除されていることを確認します。Aurora Global Database では、リージョンと AZ が 0、クラスターのサイズが 0 と表示されるはずです。

   Aurora Global Database に Aurora DB クラスターが含まれている場合は、これを削除できません。必要に応じて、プライマリとセカンダリの Aurora DB クラスターを Aurora Global Database からデタッチします。詳細については、「Amazon Aurora Global Database からのクラスターの削除」を参照してください。
4. リストから Aurora Global Database を選択し、[アクション] メニューで [削除] を選択します。



AWS CLI

Aurora Global Database を削除するには、次の例のように、AWS リージョン の名前と Aurora Global Database の識別子とを指定しながら、delete-global-cluster CLI コマンドを実行します。

Linux、macOS、Unix の場合:

```
aws rds --region primary_region delete-global-cluster \
   --global-cluster-identifier global_database_id
```

Windows の場合:

```
aws rds --region primary_region delete-global-cluster ^
   --global-cluster-identifier global_database_id
```

RDS API

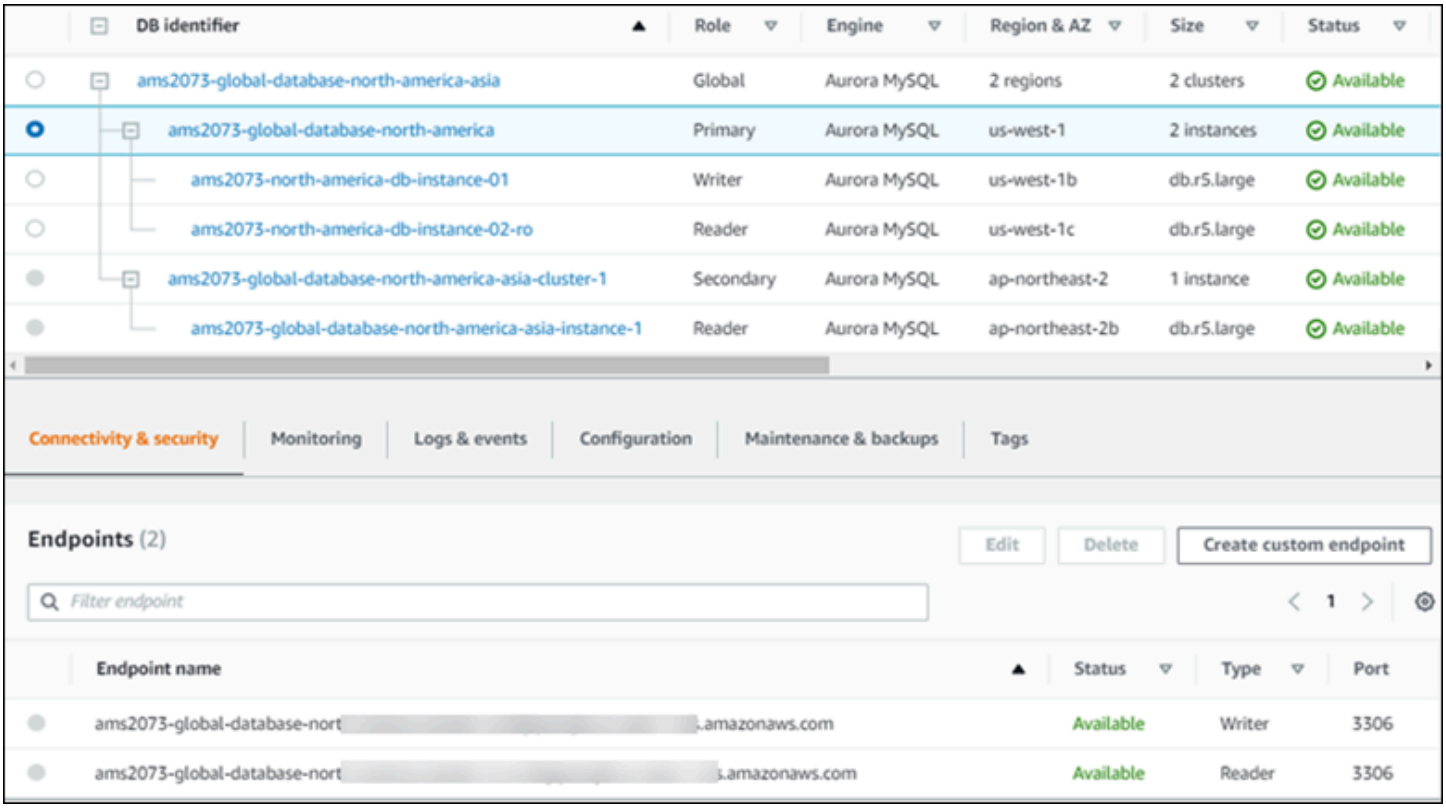
Aurora Global Database に含まれるクラスターを削除するときは、DeleteGlobalCluster API オペレーションを実行します。

Amazon Aurora Global Database への接続

Aurora Global Database への接続方法は、データベースへの書き込みと、データベースからの読み取りの、いずれを必要とするのかに応じて異なります。

- 読み取り専用のリクエストまたはクエリの場合、AWS リージョン にある Aurora クラスターのリーダーエンドポイントに接続します。
- データ操作言語 (DML) またはデータ定義言語 (DDL) のステートメントを実行するには、プライマリクラスターのクラスターエンドポイントに接続します。このエンドポイントは、アプリケーションとは異なる AWS リージョン に存在する場合があります。

コンソールで Aurora Global Database を表示すると、そのすべてのクラスターに関連付けられているすべての汎用エンドポイントを表示できます。次のスクリーンショットは、例を示しています。プライマリクラスターに関連付けられた単一のクラスターエンドポイントは、書き込みオペレーションに使用します。プライマリクラスターとセカンダリクラスターのリーダーエンドポイントは、読み取り専用クエリに使用します。レイテンシーを最小限にするときは、お使いの AWS リージョン か、最寄りの AWS リージョン の、いずれかにあるリーダーエンドポイントを選択します。次に Aurora MySQL の例を示します。



Amazon Aurora Global Database の書き込み転送を使用する

書き込み転送を使用すると Aurora Global Database で実行されているアプリケーションで管理すべきエンドポイントの数を、減らすことができます。書き込み転送を有効にすると、Aurora Global Database のセカンダリクラスターは、書き込みオペレーションを実行する SQL ステートメントをプライマリクラスターに転送することができます。プライマリクラスターが出典を更新し、結果の変更をすべてのセカンダリ AWS リージョンに反映させます。

書き込み転送設定により、セカンダリ AWS リージョンからプライマリリージョンに書き込みオペレーションを送信するための、独自のメカニズムを実装する必要がなくなります Aurora が、クロスリージョンネットワーキングの設定を処理します。また、Aurora は、各ステートメントに必要な、すべてのセッションとトランザクションのコンテキストを送信します。データは常に、初期にプライマリクラスターで変更され、次に Aurora Global Database のセカンダリクラスターにレプリケートされます。このようにして、プライマリクラスターは、すべてのデータの最新のコピーを持つ、常に信頼できる情報源となります。

トピック

- Aurora MySQL グローバルデータベースで書き込み転送を使用する
- Aurora PostgreSQL グローバルデータベースで書き込み転送を使用する

Aurora MySQL グローバルデータベースで書き込み転送を使用する

トピック

Aurora MySQL での書き込み転送を利用できるリージョンとバージョン

書き込み転送は、Aurora MySQL 2.08.1 以降のバージョンで、Aurora MySQL ベースのグローバルデータベースが利用可能なすべてのリージョンでサポートされています。

Aurora MySQL グローバルデータベースを利用できるバージョンとリージョンについては、「Aurora MySQL を使用した Aurora グローバルデータベース」を参照してください。

Aurora MySQL での書き込み転送の有効化

デフォルトでは、セカンダリクラスターを Aurora Global Database に追加すると、書き込み転送は有効になりません。

AWS Management Console を使用して書き込み転送を有効にするには、グローバルデータベースのリージョンを追加するときに、[リードレプリカの書き込み転送] の [グローバル書き込み転送を有効にする] チェックボックスをオンにします。既存のセカンダリクラスターの場合、クラスターを [グローバル書き込み転送を有効にする] に変更します。書き込み転送をオフにするには、リージョンの追加時またはセカンダリクラスターの変更時に、[グローバル書き込み転送を有効にする] チェックボックスをオフにします。

AWS CLI を使用して書き込み転送を有効にするには、`--enable-global-write-forwarding` オプションを使用します。このオプションは、`create-db-cluster` コマンドを使用して新しいセカンダリクラスターを作成するときに機能します。`modify-db-cluster` コマンドを使用して、既存のセカンダリクラスターを変更する場合にも機能します。グローバルデータベースでは、書き込み転

送をサポートする Aurora バージョンを使用する必要があります。これらの同じ CLI コマンドで --no-enable-global-write-forwarding オプションを使用すると、書き込み転送をオフにすることができます。

Amazon RDS API を使用して書き込み転送を有効にするには、EnableGlobalWriteForwarding パラメータを true に設定します。このパラメータは、CreateDBCluster オペレーションを使用して新しいセカンダリクラスターを作成するときに機能します。この操作は、ModifyDBCluster オペレーションを使用して既存のセカンダリクラスターを変更する場合にも機能します。グローバルデータベースでは、書き込み転送をサポートする Aurora バージョンを使用する必要があります。EnableGlobalWriteForwarding パラメータを false に設定することで、書き込み転送をオフにすることができます。

> **Note**
>
> データベースセッションで書き込み転送を使用するには、aurora_replica_read_consistency 構成パラメータの設定を指定します。書き込み転送機能を使用するすべてのセッションでこれを行います。このパラメータの詳細については、「Aurora MySQL での書き込み転送の分離と整合性」を参照してください。
> RDS プロキシ機能は aurora_replica_read_consistency 変数の SESSION 値をサポートしていません。この値を設定すると、予期しない動作が発生する可能性があります。

次の CLI の例は、書き込み転送を有効または無効にして Aurora Global Database を設定する方法を示しています。強調表示された項目は、Aurora Global Database のインフラストラクチャをセットアップするときに指定し、一貫性を保つために重要なコマンドとオプションを表しています。

次の例では、書き込み転送が有効になっている Aurora Global Database 、プライマリクラスター、およびセカンダリクラスターを作成します。ユーザー名、パスワード、プライマリとセカンダリの AWS リージョンは、自分で選択したものに置き換えます。

```
# Create overall global database.
aws rds create-global-cluster --global-cluster-identifier write-forwarding-test \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-1

# Create primary cluster, in the same AWS Region as the global database.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
  --db-cluster-identifier write-forwarding-test-cluster-1 \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --master-username user_name --master-user-password password \
```

```
  --region us-east-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-1 \
  --db-instance-identifier write-forwarding-test-cluster-1-instance-1 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-1 \
  --db-instance-identifier write-forwarding-test-cluster-1-instance-2 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-1

# Create secondary cluster, in a different AWS Region than the global database,
# with write forwarding enabled.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
  --db-cluster-identifier write-forwarding-test-cluster-2 \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-2 \
  --enable-global-write-forwarding

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-1 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-2

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-2 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-2
```

次の例は、前の例から続きます。書き込み転送が有効になっていないセカンダリクラスターを作成し、書き込み転送を有効にします。この例が終了すると、グローバルデータベース内のすべてのセカンダリクラスターで書き込み転送が有効になります。

```
# Create secondary cluster, in a different AWS Region than the global database,
# without write forwarding enabled.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
  --db-cluster-identifier write-forwarding-test-cluster-2 \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
```

```
  --region us-west-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-1 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-west-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-2 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-west-1

aws rds modify-db-cluster --db-cluster-identifier write-forwarding-test-cluster-2 \
  --region us-east-2 \
  --enable-global-write-forwarding
```

セカンダリクラスターで Aurora MySQL での書き込み転送が有効になっているかどうかの確認

セカンダリクラスターからの書き込み転送を使用できるかどうかを判断するには、クラスターに属性 `"GlobalWriteForwardingStatus": "enabled"` があるかどうかを確認します。

AWS Management Console で、クラスターの詳細ページの [設定] タブに、[グローバルリードレプリカの書き込み転送] のステータスが [有効] と表示されます。

すべてのクラスターのグローバル書き込み転送設定のステータスを表示するには、次の AWS CLI コマンドを実行します。

セカンダリクラスターには、書き込み転送がオンかオフかを示す値 `"enabled"` または `"disabled"` が表示されます。値 `null` は、そのクラスターで書き込み転送が使用できないことを示します。クラスターがグローバルデータベースの一部ではないか、セカンダリクラスターではなくプライマリクラスターです。書き込み転送をオンまたはオフにする処理中の場合、値は `"enabling"` または `"disabling"` になります。

Example

```
aws rds describe-db-clusters \
--query '*[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
```

```
[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    },
    {
        "GlobalWriteForwardingStatus": "disabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-2"
    },
    {
        "GlobalWriteForwardingStatus": null,
        "DBClusterIdentifier": "non-global-cluster"
    }
]
```

グローバル書き込み転送が有効になっているすべてのセカンダリクラスターを検索するには、次のコマンドを実行します。このコマンドは、クラスターのリーダーエンドポイントも返します。Aurora Global Database でセカンダリからプライマリへの書き込み転送を使用するときは、セカンダリクラスターのリーダーエンドポイントを使用します。

Example

```
aws rds describe-db-clusters --query 'DBClusters[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
 | [?GlobalWriteForwardingStatus == `enabled`]'
[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "ReaderEndpoint": "aurora-write-forwarding-test-replica-1.cluster-ro-
cnpexample.us-west-2.rds.amazonaws.com",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    }
]
```

Aurora MySQL での書き込み転送とアプリケーションおよび SQL の互換性

書き込み転送では、次の種類の SQL ステートメントを使用できます。

- INSERT、DELETE、および UPDATE などのデータ操作言語 (DML) ステートメント。書き込み転送で使用できるこれらのステートメントのプロパティには、以下で説明するように、いくつかの制限があります。
- SELECT ... LOCK IN SHARE MODE と SELECT FOR UPDATE ステートメント。

- PREPARE と EXECUTE ステートメント。

特定のステートメントは、書き込み転送機能を持つグローバルデータベースで使用すると、許可されないか、または古い結果を生成する可能性があります。したがって、セカンダリクラスターで EnableGlobalWriteForwarding の設定はデフォルトではオフになっています。オンにする前に、アプリケーションコードがこれらの制限の影響を受けていないことを確認してください。

書き込み転送で使用する SQL ステートメントには、次の制限が適用されます。場合によっては、書き込み転送がクラスターレベルで有効になっているセカンダリクラスターでステートメントを使用できます。この方法は、aurora_replica_read_consistency 設定パラメータによってセッション内で書き込み転送がオンになっていない場合に機能します。書き込み転送のために許可されていないステートメントを使用しようとすると、次の形式のエラーメッセージが表示されます。

```
ERROR 1235 (42000): This version of MySQL doesn't yet support 'operation with write
 forwarding'.
```

データ定義言語 (DDL)

プライマリクラスターに接続して、DDL ステートメントを実行します。リーダー DB インスタンスからは実行できません。

テンポラリテーブルのデータを使用した永続テーブルの更新

書き込み転送が有効になっているセカンダリクラスターでは、テンポラリテーブルを使用できます。ただし、ステートメントがテンポラリテーブルを参照している場合は、DML ステートメントを使用して永続テーブルを変更することはできません。例えば、テンポラリテーブルからデータを取る INSERT ... SELECT ステートメントを使用することはできません。テンポラリテーブルはセカンダリクラスターに存在し、プライマリクラスターでステートメントを実行するときには使用できません。

XA トランザクション

セッション内で書き込み転送が有効になっている場合、セカンダリクラスターで次のステートメントを使用することはできません。これらのステートメントは、書き込み転送が有効になっていないセカンダリクラスター、または aurora_replica_read_consistency 設定が空のセッションで使用できます。セッション内で書き込み転送を有効にする前に、コードでこれらのステートメントが使用されているかどうかを確認してください。

```
XA {START|BEGIN} xid [JOIN|RESUME]
XA END xid [SUSPEND [FOR MIGRATE]]
```

```
XA PREPARE xid
XA COMMIT xid [ONE PHASE]
XA ROLLBACK xid
XA RECOVER [CONVERT XID]
```

永続テーブルの LOAD ステートメント

書き込み転送が有効になっているセカンダリクラスターでは、次のステートメントを使用できません。

```
LOAD DATA INFILE 'data.txt' INTO TABLE t1;
        LOAD XML LOCAL INFILE 'test.xml' INTO TABLE t1;
```

セカンダリクラスターのテンポラリテーブルにデータをロードできます。ただし、永続テーブルを参照する LOAD ステートメントは、プライマリクラスターでのみ実行してください。

プラグインステートメント

書き込み転送が有効になっているセカンダリクラスターでは、次のステートメントを使用できません。

```
INSTALL PLUGIN example SONAME 'ha_example.so';
UNINSTALL PLUGIN example;
```

SAVEPOINT ステートメント

セッション内で書き込み転送が有効になっている場合、セカンダリクラスターで次のステートメントを使用することはできません。これらのステートメントは、書き込み転送が有効になっていないセカンダリクラスター、または aurora_replica_read_consistency 設定が空白のセッションで使用できます。セッション内で書き込み転送を有効にする前に、コードでこれらのステートメントが使用されているかどうかを確認してください。

```
SAVEPOINT t1_save;
ROLLBACK TO SAVEPOINT t1_save;
RELEASE SAVEPOINT t1_save;
```

Aurora MySQL での書き込み転送の分離と整合性

書き込み転送を使用するセッションでは、REPEATABLE READ 分離レベルのみを使用できます。セカンダリ READ COMMITTED リージョンの読み取り専用クラスターでも AWS 分離レベルを使用

できますが、その分離レベルは書き込み転送では機能しません。REPEATABLE READ および READ COMMITTED 分離レベルの詳細については、「Aurora MySQL の分離レベル」を参照してください。

セカンダリクラスターの読み取り整合性の程度を制御できます。読み取り整合性レベルは、一部またはすべての変更がプライマリクラスターからレプリケートされるように、各読み取りオペレーションの前にセカンダリクラスターが実行する待機時間を決定します。読み取り整合性レベルを調整して、セッションから転送されたすべての書き込みオペレーションが、後続のクエリの前にセカンダリクラスターに表示されるようにすることができます。また、この設定を使用して、セカンダリクラスターのクエリに、常にプライマリクラスターからの最新の更新が表示されるようにすることもできます。これは、他のセッションまたは他のクラスターによって送信されたものであっても同様です。アプリケーションでこの種類の動作を指定するには、セッションレベルのパラメータ aurora_replica_read_consistency の値を選択します。

> **Important**
>
> 書き込みを転送するセッションには、必ず aurora_replica_read_consistency パラメータを設定します。それ以外の場合、Aurora はそのセッションの書き込み転送を有効にしません。デフォルトでは、このパラメータに空の値があるため、このパラメータを使用する場合は特定の値を選択してください。aurora_replica_read_consistency パラメータは、書き込み転送が有効になっているセカンダリクラスターでのみ有効です。
> Aurora MySQL バージョン 2 および 3.04 より前のバージョン 3 の場合は、aurora_replica_read_consistency をセッション変数として使用します。Aurora MySQL バージョン 3.04 以降の場合、aurora_replica_read_consistency をセッション変数として、または DB クラスターパラメータとして使用できます。

aurora_replica_read_consistency パラメータには、EVENTUAL、SESSION、および GLOBAL の値を指定できます。

整合性レベルを上げると、アプリケーションは、AWS リージョン間で変更が反映されるのを待つ時間が長くなります。応答時間の短縮と、クエリを実行する前に他の場所で行われた変更が完全に使用可能であることのバランスを選択できます。

読み取り整合性を EVENTUAL に設定した場合、書き込み転送を使用するセカンダリ AWS リージョンのクエリでは、レプリケーションの遅延によりデータがわずかに古くなることがあります。同じセッションでの書き込みオペレーションの結果は、プライマリリージョンで書き込みオペレーションが実行され、現在のリージョンにレプリケートされるまで表示されません。クエリは、更新された結

果が使用可能になるのを待つことはありません。したがって、ステートメントのタイミングとレプリケーションの遅延の量に応じて、古いデータや更新されたデータが取得される可能性があります。

読み取り整合性を SESSION に設定した場合、書き込み転送を使用するセカンダリ AWS リージョンのすべてのクエリに、そのセッションで行われたすべての変更の結果が表示されます。トランザクションがコミットされているかどうかにかかわらず、変更が表示されます。必要に応じて、クエリは、転送された書き込みオペレーションの結果が現在のリージョンにレプリケートされるまで待っています。他のリージョンまたは現在のリージョン内の他のセッションで実行された書き込みオペレーションの結果が更新されるのを待つことはありません。

読み取り整合性を GLOBAL に設定した場合、セカンダリ AWS リージョンのセッションには、そのセッションによって行われた変更が表示されます。また、プライマリ AWS リージョンと他のセカンダリ AWS リージョンの両方のコミットされた変更もすべて表示されます。各クエリは、セッション遅延の量に応じて変化する期間を待つことがあります。クエリは、クエリがスタートされた時点の、プライマリクラスターからコミットされたすべてのデータでセカンダリクラスターが最新の状態になったときに実行されます。

書き込み転送に関連するすべてのパラメータの詳細については、「Aurora MySQL での書き込み転送の設定パラメータ」を参照してください。

書き込み転送の使用例

これらの例では、aurora_replica_read_consistency をセッション変数として使用しています。Aurora MySQL バージョン 3.04 以降の場合、aurora_replica_read_consistency をセッション変数として、または DB クラスターパラメータとして使用できます。

次の例では、プライマリクラスターが US East (N. Virginia) リージョンにあります。セカンダリクラスターは 米国東部 (オハイオ) リージョンにあります。この例は、INSERT ステートメントの後に SELECT ステートメントが続いて実行された場合の結果を示しています。aurora_replica_read_consistency 設定値によっては、ステートメントのタイミングによって結果が異なる場合があります。一貫性を高めるには、SELECT ステートメントを発行する前にしばらくお待ちください。または Aurora は、結果のレプリケーションが完了するまで自動的に待機してから、SELECT 処理を続行することができます。

この例では、eventual の読み取り整合性設定があります。INSERT ステートメントの直後に SELECT ステートメントを実行すると、COUNT(*) の値が戻されます。この値は、新しい行が挿入される前の行数を反映します。しばらくしてから SELECT を再度実行すると、更新された行数が返されます。SELECT ステートメントは待機しません。

```
mysql> set aurora_replica_read_consistency = 'eventual';
```

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)
mysql> insert into t1 values (6); select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.00 sec)
```

読み取り整合性設定が session の場合、INSERT の直後の SELECT ステートメントは、INSERT ステートメントからの変更が表示されるまで待機します。後続の SELECT ステートメントは待機しません。

```
mysql> set aurora_replica_read_consistency = 'session';
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.01 sec)
mysql> insert into t1 values (6); select count(*) from t1; select count(*) from t1;
Query OK, 1 row affected (0.08 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.37 sec)
+----------+
```

```
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.00 sec)
```

読み取り整合性設定を session に設定したまま、INSERT ステートメントの実行後に短い待機を行うと、次の SELECT ステートメントが実行されるまでに更新された行カウントが使用可能になります。

```
mysql> insert into t1 values (6); select sleep(2); select count(*) from t1;
Query OK, 1 row affected (0.07 sec)
+----------+
| sleep(2) |
+----------+
|        0 |
+----------+
1 row in set (2.01 sec)
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.00 sec)
```

読み取り整合性設定が global の場合、各 SELECT ステートメントは、クエリを実行する前に、ステートメントのスタート時刻時点のすべてのデータ変更が表示されるように待っています。各 SELECT ステートメントの待機時間は、プライマリクラスターとセカンダリクラスター間のレプリケーション遅延の量によって異なります。

```
mysql> set aurora_replica_read_consistency = 'global';
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.75 sec)
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
```

```
|        8 |
+----------+
1 row in set (0.37 sec)
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.66 sec)
```

Aurora MySQL での書き込み転送を使用したマルチパートステートメントの実行

DML ステートメントは、INSERT ... SELECT ステートメントや DELETE ... WHERE ステートメントなど、複数の部分から構成される場合があります。この場合、ステートメント全体がプライマリクラスターに転送され、そこで実行されます。

Aurora MySQL での書込み転送を使用したトランザクション

トランザクションがプライマリクラスターに転送されるかどうかは、トランザクションのアクセスモードによって異なります。SET TRANSACTION ステートメントまたは START TRANSACTION ステートメントを使用して、トランザクションのアクセスモードを指定できます。Aurora MySQL セッション可変 tx_read_only の値を変更することで、トランザクションアクセスモードを指定することもできます。このセッション値は、書き込み転送が有効になっているセカンダリクラスターに接続しているときにのみ変更できます。

長時間実行されるトランザクションがかなりの期間ステートメントを発行しない場合、アイドルタイムアウト期間を超える可能性があります。この期間のデフォルトは 1 分です。1 日まで増やすことができます。アイドルタイムアウトを超えたトランザクションは、プライマリクラスターによってキャンセルされます。次に送信するステートメントは、タイムアウトエラーを受け取ります。その後、Aurora はトランザクションをロールバックします。

このタイプのエラーは、書き込み転送が使用できなくなった場合に発生する可能性があります。例えば、プライマリクラスターを再起動するか、書き込み転送構成設定を無効にした場合、Aurora は書き込み転送を使用するすべてのトランザクションをキャンセルします。

Aurora MySQL での書き込み転送の設定パラメータ

Aurora クラスターのパラメータグループには、書き込み転送機能の設定が含まれています。これらはクラスターパラメータであるため、各クラスターのすべての DB インスタンスは、これらの可変に

同じ値を持ちます。これらのパラメータの詳細を次の表にまとめ、表に続いて使用上の注意を記載してください。

名前	スコープ	タイプ	デフォルト値	有効な値
aurora_fwd_master_idle_timeout (Aurora MySQL バージョン 2)	グローバル	符号なし整数	60	1-86,400
aurora_fwd_master_max_connections_pct (Aurora MySQL バージョン 2)	グローバル	符号なし長整数	10	0-90
aurora_fwd_writer_idle_timeout (Aurora MySQL バージョン 3)	グローバル	符号なし整数	60	1-86,400
aurora_fwd_writer_max_connections_pct (Aurora MySQL バージョン 3)	グローバル	符号なし長整数	10	0-90
aurora_replica_read_consistency	セッション	列挙型	null	EVENTUAL, SESSION, GLOBAL

セカンダリクラスターからの着信書き込みリクエストを制御するには、プライマリクラスターで次の設定を使用します。

- aurora_fwd_master_idle_timeout、 aurora_fwd_writer_idle_timeout: プライマリクラスターがセカンダリクラスターから転送された接続でアクティビティを終了するまで待機する秒数。この期間を超えてセッションがアイドル状態のままである場合、Aurora はセッションをキャンセルします。
- aurora_fwd_master_max_connections_pct、aurora_fwd_writer_max_connections_pct: リーダーから転送されたクエリを処理するためにライター DB インスタンスで使用できるデータベース接続の上限。これは、プライマリクラスター内の書き込み DB インスタンスの max_connections 設定のパーセンテージで表されます。例えば、max_connections が800 回、aurora_fwd_master_max_connections_pctまたはaurora_fwd_writer_max_connections_pctが 10 回 の場合、書き込みは最大 80 回 の

同時転送セッションを許可します。これらの接続は、max_connections 設定によって管理される同じ接続プールから取得されます。

この設定は、1 つ以上のセカンダリクラスターで書き込み転送が有効になっている場合に、プライマリクラスターにのみ適用されます。この値を小さくしても、既存の接続は影響を受けません。Aurora は、セカンダリクラスターから新しい接続を作成するときに、この新しい設定値を参照します。デフォルト値は 10 で、max_connections 値の 10% を表します。セカンダリクラスターのいずれかでクエリ転送を有効にする場合、セカンダリクラスターからの書き込みオペレーションを正常に実行するには、この設定に 0 以外の値を指定する必要があります。値が 0 の場合、書き込みオペレーションは、メッセージ ER_CON_COUNT_ERROR とともにエラーコード Not enough connections on writer to handle your request を受け取ります。

aurora_replica_read_consistency パラメータは、書き込み転送を有効にするセッションレベルのパラメータです。各セッションでそれを使用します。読み取り整合性レベルには、EVENTUAL、SESSION、または GLOBAL を指定できます。整合性レベルの詳細については、Aurora MySQL での書き込み転送の分離と整合性 を参照してください。このパラメータには、次の規則が適用されます。

- これはセッションレベルのパラメータです。デフォルト値は '' (空) です。
- 書き込み転送は、aurora_replica_read_consistency が EVENTUAL、SESSION、または GLOBAL に設定されている場合にのみセッションで使用できます。このパラメータは、書き込み転送が有効で、Aurora Global Database 内にあるセカンダリクラスターのリーダーインスタンスでのみ関係します。
- マルチステートメントトランザクション内では、この可変を設定する (空の場合) または設定解除する (既に設定されている場合) ことはできません。ただし、トランザクション中に、有効な値 (EVENTUAL、SESSION、GLOBAL) から別の値 (EVENTUAL、SESSION、GLOBAL) に変更することは可能です。
- セカンダリクラスターで書き込み転送が有効になっていない場合、可変は SET に指定できません。
- プライマリクラスターでセッション可変を設定しても、効果はありません。プライマリクラスターでこの可変を変更しようとすると、エラーが発生します。

Aurora MySQL での書き込み転送の Amazon CloudWatch メトリクス

次の Amazon CloudWatch メトリクスと Aurora MySQL ステータス変数は、1 つ以上のセカンダリクラスターで書き込み転送を使用するときに、プライマリクラスターに適用されます。これらのメトリクスはすべて、プライマリクラスターのライター DB インスタンスで測定されます。

CloudWatch メトリクス	Aurora MySQL ステータス変数	単位	説明
`ForwardingMasterDMLLatency`	–	ミリ秒	書き込み DB インスタンスで転送された各 DML ステートメントを処理する平均時間。 セカンダリクラスターが書き込みリクエストを転送する時間や、変更をセカンダリクラスターに複製する時間は含まれません。 Aurora MySQL バージョン 2 の場合。
`ForwardingMasterDMLThroughput`	–	1 秒あたりのカウント数	この書き込み DB インスタンスによって 1 秒間に処理される、転送された DML ステートメントの数。 Aurora MySQL バージョン 2 の場合。
`ForwardingMasterOpenSessions`	`Aurora_fwd_master_open_sessions`	Count (カウント)	書き込み DB インスタンスで転送されたセッションの数。

CloudWatch メトリクス	Aurora MySQL ステータス変数	単位	説明
			Aurora MySQL バージョン 2 の場合。
–	`Aurora_fwd_master_dml_stmt_count`	Count (カウント)	この書き込み DB インスタンスに転送された DML ステートメントの合計数。 Aurora MySQL バージョン 2 の場合。
–	`Aurora_fwd_master_dml_stmt_duration`	マイクロ秒	この書き込み DB インスタンスに転送された DML ステートメントの合計期間。 Aurora MySQL バージョン 2 の場合。
–	`Aurora_fwd_master_select_stmt_count`	Count (カウント)	この書き込み DB インスタンスに転送された `SELECT` ステートメントの総数。 Aurora MySQL バージョン 2 の場合。
–	`Aurora_fwd_master_select_stmt_duration`	マイクロ秒	この書き込み DB インスタンスに転送された `SELECT` ステートメントの合計期間。 Aurora MySQL バージョン 2 の場合。

CloudWatch メトリクス	Aurora MySQL ステータス変数	単位	説明
ForwardingWriterDMLLatency	–	ミリ秒	書き込み DB インスタンスで転送された各 DML ステートメントを処理する平均時間。 セカンダリクラスターが書き込みリクエストを転送する時間や、変更をセカンダリクラスターに複製する時間は含まれません。 Aurora MySQL バージョン 3 の場合。
ForwardingWriterDMLThroughput	–	1 秒あたりのカウント数	この書き込み DB インスタンスによって 1 秒間に処理される、転送された DML ステートメントの数。Aurora MySQL バージョン 3 の場合。
ForwardingWriterOpenSessions	Aurora_fwd_writer_open_sessions	Count (カウント)	書き込み DB インスタンスで転送されたセッションの数。Aurora MySQL バージョン 3 の場合。

CloudWatch メトリクス	Aurora MySQL ステータス変数	単位	説明
–	`Aurora_fwd_writer_dml_stmt_count`	Count (カウント)	この書き込み DB インスタンスに転送された DML ステートメントの合計数。 Aurora MySQL バージョン 3 の場合。
–	`Aurora_fwd_writer_dml_stmt_duration`	マイクロ秒	この書き込み DB インスタンスに転送された DML ステートメントの合計期間。
–	`Aurora_fwd_writer_select_stmt_count`	Count (カウント)	この書き込み DB インスタンスに転送された SELECT ステートメントの総数。 Aurora MySQL バージョン 3 の場合。
–	`Aurora_fwd_writer_select_stmt_duration`	マイクロ秒	この書き込み DB インスタンスに転送された SELECT ステートメントの合計期間。 Aurora MySQL バージョン 3 の場合。

次の CloudWatch メトリクスと Aurora MySQL ステータス変数は、各セカンダリクラスターに適用されます。これらのメトリクスは、書き込み転送が有効になっているセカンダリクラスターのリーダー DB インスタンスで測定されます。

CloudWatch メトリクス	Aurora MySQL ステータス変数	単位	説明
ForwardingReplicaDMLLatency	–	ミリ秒	レプリカ上で転送された DML の平均応答時間。
ForwardingReplicaDMLThroughput	–	1 秒あたりのカウント数	1 秒あたりに処理された転送 DML ステートメントの数。
ForwardingReplicaOpenSessions	Aurora_fwd_replica_open_sessions	Count (カウント)	リーダー DB インスタンスで書き込み転送を使用しているセッションの数。
ForwardingReplicaReadWaitLatency	–	ミリ秒	リーダー DB インスタンス上の SELECT ステートメントがプライマリクラスターに追いつくのを待機する平均待機時間。 クエリを処理する前にリーダー DB インスタンスが待機する程度は、aurora_replica_read_consistency 設定によって異なります。
ForwardingReplicaReadWaitThroughput	–	1 秒あたりのカウント数	書き込みを転送しているすべてのセッションで、1 秒間処理した SELECT ステートメントの総数。

CloudWatch メトリクス	Aurora MySQL ステータス変数	単位	説明
ForwardingReplicaSelectLatency	(–)	ミリ秒	転送済みSELECTレイテンシー、SELECTモニタリング期間内に転送されたすべてのステートメントの平均値。
ForwardingReplicaSelectThroughput	–	1 秒あたりのカウント数	モニタリング期間内の 1 秒あたりの転送済み SELECT スループット。
–	Aurora_fwd_replica_dml_stmt_count	Count (カウント)	このリーダー DB インスタンスから転送された DML ステートメントの合計数。
–	Aurora_fwd_replica_dml_stmt_duration	マイクロ秒	このリーダー DB インスタンスから転送された DML ステートメントの合計期間。
–	Aurora_fwd_replica_errors_session_limit	Count (カウント)	以下のエラー条件のいずれかが原因でプライマリクラスターが拒否したセッションの数。 • writer full • 転送中のステートメントが多すぎます。

CloudWatch メトリクス	Aurora MySQL ステータス変数	単位	説明
–	`Aurora_fwd_replica_read_wait_count`	Count (カウント)	このリーダー DB インスタンスでの書き込み後の読み取り待機の合計数。
–	`Aurora_fwd_replica_read_wait_duration`	マイクロ秒	このリーダー DB インスタンスの読み取り整合性設定による待機時間の合計。
–	`Aurora_fwd_replica_select_stmt_count`	Count (カウント)	このリーダー DB インスタンスから転送された SELECT ステートメントの合計数。
–	`Aurora_fwd_replica_select_stmt_duration`	マイクロ秒	このリーダー DB インスタンスから転送された SELECT ステートメントの合計期間。

Aurora PostgreSQL グローバルデータベースで書き込み転送を使用する

トピック

- Aurora PostgreSQL での書き込み転送を利用できるリージョンとバージョン
- Aurora PostgreSQL での書き込み転送の有効化
- セカンダリクラスターで Aurora PostgreSQL での書き込み転送が有効になっているかどうかの確認
- 書き込み転送とアプリケーションおよび Aurora PostgreSQL の互換性
- Aurora PostgreSQL での書き込み転送の分離と整合性
- Aurora PostgreSQL での書き込み転送を使用したマルチパートステートメントの実行
- Aurora PostgreSQL での書き込み転送の設定パラメータ

- Aurora PostgreSQL での書き込み転送の Amazon CloudWatch メトリクス
- Aurora PostgreSQL での書き込み転送のイベントを待機する

Aurora PostgreSQL での書き込み転送を利用できるリージョンとバージョン

書き込み転送は、Aurora PostgreSQL バージョン 15.4 以降のマイナーバージョンと、バージョン 14.9 以降のマイナーバージョンでサポートされます。書き込み転送は、Aurora PostgreSQL ベースのグローバルデータベースが利用可能なすべてのリージョンで利用できます。

Aurora PostgreSQL グローバルデータベースで利用できるバージョンとリージョンについては、「Aurora PostgreSQL を使用した Aurora グローバルデータベース」を参照してください。

Aurora PostgreSQL での書き込み転送の有効化

デフォルトでは、セカンダリクラスターを Aurora Global Database に追加すると、書き込み転送は有効になりません。セカンダリ DB クラスターの書き込み転送は、作成中または作成後にいつでも有効にできます。必要に応じて、後で無効にすることができます。書き込み転送を有効または無効にしても、ダウンタイムや再起動は発生しません。

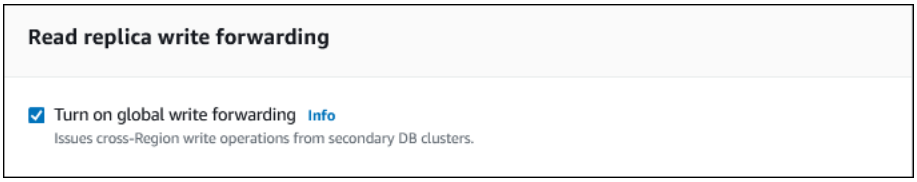
コンソール

コンソールでは、セカンダリ DB クラスターを作成または変更するときに、書き込み転送を有効または無効にすることができます。

セカンダリ DB クラスターの作成時に書き込み転送を有効または無効にする

新しいセカンダリ DB クラスターを作成するときは、[リードレプリカの書き込み転送] の [グローバル書き込み転送を有効にする] チェックボックスをオンにして、書き込み転送を有効にします。または、チェックボックスをオフにしてこの機能を無効にします。セカンダリ DB クラスターを作成するには、「Amazon Aurora DB クラスターの作成」の DB エンジンの手順に従ってください。

次のスクリーンショットは、[グローバル書き込み転送を有効にする] チェックボックスがオンになっている[リードレプリカの書き込み転送] セクションを示しています。



セカンダリ DB クラスターの修正時に書き込み転送を有効または無効にする

コンソールでは、セカンダリ DB クラスターを修正して書き込み転送を有効または無効にすることができます。

コンソールでセカンダリ DB クラスターの書き込み転送を有効または無効にするには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. [データベース] をクリックします。
3. セカンダリ DB クラスターを選択し、[変更] を選択します。
4. [リードレプリカの書き込み転送] セクションで、[グローバル書き込み転送を有効にする] チェックボックスをオンまたはオフにします。
5. Continue (続行) をクリックします。
6. [変更をスケジュール] で、[すぐに適用] を選択します。次に [スケジュールされたメンテナンスウィンドウで適用] を選択すると、Aurora ではこの設定が無視され、書き込み転送が直ちにオンになります。
7. [クラスタークラスターの変更] を選択します。

AWS CLI

AWS CLI を使用して書き込み転送を有効にするには、`--enable-global-write-forwarding` オプションを使用します。このオプションは、create-db-cluster コマンドを使用して新しいセカンダリクラスターを作成するときに機能します。modify-db-cluster コマンドを使用して、既存のセカンダリクラスターを変更する場合にも機能します。グローバルデータベースでは、書き込み転送をサポートする Aurora バージョンを使用する必要があります。これらの同じ CLI コマンドで `--no-enable-global-write-forwarding` オプションを使用することで、書き込み転送をオフにすることができます。

以下の手順では、AWS CLI を使用してグローバルクラスター内のセカンダリ DB クラスターの書き込み転送を有効または無効にする方法について説明します。

既存のセカンダリ DB クラスターの書き込み転送を有効または無効にするには

- modify-db-cluster AWS CLI コマンドを呼び出して以下の値を指定します。
  - `--db-cluster-identifier` - DB クラスターの名前。

- オンにする場合は `--enable-global-write-forwarding`、オフにする場合は `--no-enable-global-write-forwarding`。

次の例では、DB クラスター `sample-secondary-db-cluster` の書き込み転送を有効にします。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-secondary-db-cluster \
    --enable-global-write-forwarding
```

Windows の場合:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-secondary-db-cluster ^
    --enable-global-write-forwarding
```

RDS API

Amazon RDS API を使用して書き込み転送を有効にするには、`EnableGlobalWriteForwarding` パラメータを `true` に設定します。このパラメータは、CreateDBCluster オペレーションを使用して新しいセカンダリクラスターを作成するときに機能します。この操作は、ModifyDBCluster オペレーションを使用して既存のセカンダリクラスターを変更する場合にも機能します。グローバルデータベースでは、書き込み転送をサポートする Aurora バージョンを使用する必要があります。`EnableGlobalWriteForwarding` パラメータを `false` に設定することで、書き込み転送をオフにすることができます。

セカンダリクラスターで Aurora PostgreSQL での書き込み転送が有効になっているかどうかの確認

セカンダリクラスターからの書き込み転送を使用できるかどうかを判断するには、クラスターに属性 `"GlobalWriteForwardingStatus": "enabled"` があるかどうかを確認します。

AWS Management Console では、クラスターの詳細ページの [設定] タブに `Read replica write forwarding` が表示されます。すべてのクラスターのグローバル書き込み転送設定のステータスを表示するには、次の AWS CLI コマンドを実行します。

セカンダリクラスターには、書き込み転送がオンかオフかを示す値 "enabled" または "disabled" が表示されます。値 null は、そのクラスターで書き込み転送が使用できないことを示します。クラスターがグローバルデータベースの一部ではないか、セカンダリクラスターではなくプライマリクラスターです。書き込み転送をオンまたはオフにする処理中の場合、値は "enabling" または "disabling" になります。

Example

```
aws rds describe-db-clusters --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    },
    {
        "GlobalWriteForwardingStatus": "disabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-2"
    },
    {
        "GlobalWriteForwardingStatus": null,
        "DBClusterIdentifier": "non-global-cluster"
    }
]
```

グローバル書き込み転送が有効になっているすべてのセカンダリクラスターを検索するには、次のコマンドを実行します。このコマンドは、クラスターのリーダーエンドポイントも返します。Aurora Global Database でセカンダリからプライマリへの書き込み転送を使用するときは、セカンダリクラスターのリーダーエンドポイントを使用します。

Example

```
aws rds describe-db-clusters --query 'DBClusters[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
 | [?GlobalWriteForwardingStatus == `enabled`]'
[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "ReaderEndpoint": "aurora-write-forwarding-test-replica-1.cluster-ro-
cnpexample.us-west-2.rds.amazonaws.com",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    }
```

```
]
```

書き込み転送とアプリケーションおよび Aurora PostgreSQL の互換性

特定のステートメントは、書き込み転送機能を持つグローバルデータベースで使用すると、許可されないか、または古い結果を生成する可能性があります。また、ユーザー定義関数とユーザー定義プロシージャはサポートされていません。したがって、セカンダリクラスターで EnableGlobalWriteForwarding の設定はデフォルトではオフになっています。オンにする前に、アプリケーションコードがこれらの制限の影響を受けていないことを確認してください。

書き込み転送では、次の種類の SQL ステートメントを使用できます。

- INSERT、DELETE、および UPDATE などのデータ操作言語 (DML) ステートメント。
- SELECT FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE } ステートメント
- PREPARE と EXECUTE ステートメント
- このリストにあるステートメントを含む EXPLAIN ステートメント

書き込み転送では、次の種類の SQL ステートメントはサポートされていません。

- データ定義言語 (DDL) ステートメント
- ANALYZE
- CLUSTER
- COPY
- カーソル — カーソルはサポートされていないため、書き込み転送を使用する前に必ずカーソルを閉じてください。
- GRANT|REVOKE|REASSIGN OWNED|SECURITY LABEL
- LOCK
- SAVEPOINT ステートメント
- SELECT INTO
- SET CONSTRAINTS
- TRUNCATE
- VACUUM

Aurora PostgreSQL での書き込み転送の分離と整合性

書き込み転送を使用するセッションでは、REPEATABLE READ および READ COMMITTED 分離レベルのみを使用できます。ただし、SERIALIZABLE 分離レベルはサポートされていません。

セカンダリクラスターの読み取り整合性の程度を制御できます。読み取り整合性レベルは、一部またはすべての変更がプライマリクラスターからレプリケートされるように、各読み取りオペレーションの前にセカンダリクラスターが実行する待機時間を決定します。読み取り整合性レベルを調整して、セッションから転送されたすべての書き込みオペレーションが、後続のクエリの前にセカンダリクラスターに表示されるようにすることができます。また、この設定を使用して、セカンダリクラスターのクエリに、常にプライマリクラスターからの最新の更新が表示されるようにすることもできます。これは、他のセッションまたは他のクラスターによって送信されたものであっても同様です。アプリケーションでこの種類の動作を指定するには、セッションレベルのパラメータ apg_write_forward.consistency_mode の適切な値を選択します。apg_write_forward.consistency_mode パラメータは、書き込み転送が有効になっているセカンダリクラスターでのみ有効です。

> **Note**
>
> apg_write_forward.consistency_mode パラメータには、SESSION、EVENTUAL、GLOBAL および OFF の値を指定できます。デフォルトでは、値は SESSION に設定されます。値を OFF に設定すると、セッションでの書き込み転送が無効になります。

整合性レベルを上げると、アプリケーションは、AWS リージョン間で変更が反映されるのを待つ時間が長くなります。応答時間の短縮と、クエリを実行する前に他の場所で行われた変更が完全に使用可能であることのバランスを選択できます。

整合性モードを設定できるたびに、次のような効果が得られます。

- SESSION – 書き込み転送を使用するセカンダリ AWS リージョンのすべてのクエリに、そのセッションで行われたすべての変更の結果が表示されます。トランザクションがコミットされているかどうかにかかわらず、変更が表示されます。必要に応じて、クエリは、転送された書き込みオペレーションの結果が現在のリージョンにレプリケートされるまで待っています。他のリージョンまたは現在のリージョン内の他のセッションで実行された書き込みオペレーションの結果が更新されるのを待つことはありません。

- EVENTUAL - 書き込み転送を使用するセカンダリ AWS リージョンのクエリでは、レプリケーションの遅延によりデータがわずかに古くなることがあります。同じセッションでの書き込みオペレーションの結果は、プライマリリージョンで書き込みオペレーションが実行され、現在のリージョンにレプリケートされるまで表示されません。クエリは、更新された結果が使用可能になるのを待つことはありません。したがって、ステートメントのタイミングとレプリケーションの遅延の量に応じて、古いデータや更新されたデータが取得される可能性があります。
- GLOBAL — セカンダリ AWS リージョンのセッションには、そのセッションによって行われた変更が表示されます。また、プライマリ AWS リージョンと他のセカンダリ AWS リージョンの両方のコミットされた変更もすべて表示されます。各クエリは、セッション遅延の量に応じて変化する期間を待つことがあります。クエリは、クエリがスタートされた時点の、プライマリクラスターからコミットされたすべてのデータでセカンダリクラスターが最新の状態になったときに実行されます。
- OFF — セッションで書き込み転送が無効です。

書き込み転送に関連するすべてのパラメータの詳細については、「Aurora PostgreSQL での書き込み転送の設定パラメータ」を参照してください。

Aurora PostgreSQL での書き込み転送を使用したマルチパートステートメントの実行

DML ステートメントは、INSERT ... SELECT ステートメントや DELETE ... WHERE ステートメントなど、複数の部分から構成される場合があります。この場合、ステートメント全体がプライマリクラスターに転送され、そこで実行されます。

Aurora PostgreSQL での書き込み転送の設定パラメータ

Aurora クラスターのパラメータグループには、書き込み転送機能の設定が含まれています。これらはクラスターパラメータであるため、各クラスターのすべての DB インスタンスは、これらの可変に同じ値を持ちます。これらのパラメータの詳細を次の表にまとめ、表に続いて使用上の注意を記載してください。

名前	スコープ	タイプ	デフォルト値	有効値
apg_write_forward.connect_timeout	セッション	秒	30	0-2147483647
apg_write_forward.consistency_mode	セッション	enum	セッション	SESSION, EVENTUAL,

名前	スコープ	タイプ	デフォルト値	有効値
				GLOBAL, OFF
apg_write_forward.idle_in_transaction_session_timeout	セッション	ミリ秒	86400000	0-2147483647
apg_write_forward.idle_session_timeout	セッション	ミリ秒	300000	0-2147483647
apg_write_forward.max_forwarding_connections_percent	グローバル	整数	25	1–100

apg_write_forward.max_forwarding_connections_percent パラメータは、リーダーから転送されたクエリを処理するために使用できるデータベース接続スロットの上限です。これは、プライマリクラスター内の書き込み DB インスタンスの max_connections 設定のパーセンテージで表されます。例えば、max_connections が 800、apg_write_forward.max_forwarding_connections_percent が 10 の場合、書き込みは最大 80 の同時転送セッションを許可します。これらの接続は、max_connections 設定によって管理される同じ接続プールから取得されます。この設定は、セカンダリクラスターで少なくとの 1 つの書き込み転送が有効になっている場合に、プライマリクラスターにのみ適用されます。

セカンダリクラスターでは以下の設定を使用します。

- apg_write_forward.consistency_mode — セカンダリクラスターの読み取り整合性の程度を制御するセッションレベルのパラメータ。有効な値は SESSION、EVENTUAL、GLOBAL、または OFF です。デフォルトでは、値は SESSION に設定されます。値を OFF に設定すると、セッションでの書き込み転送が無効になります。整合性レベルの詳細については、Aurora PostgreSQL での書き込み転送の分離と整合性 を参照してください。このパラメータは、書き込み転送が有効で、Aurora Global Database 内にあるセカンダリクラスターのリーダーインスタンスでのみ関係します。
- apg_write_forward.connect_timeout — セカンダリクラスターがプライマリクラスターへの接続を確立するまでに待機する最大秒数。値が 0 の場合、無期限に待機することになります。
- apg_write_forward.idle_in_transaction_session_timeout – プライマリクラスターが開いているセッションのあるセカンダリクラスターから転送された接続でそれを閉じるまでアク

ティビティで待機するミリ秒数。トランザクションでこの期間を超えてセッションがアイドル状態のままである場合、Aurora はセッションを終了します。0 の値は、タイムアウトを無効にします。

- apg_write_forward.idle_session_timeout - プライマリクラスターがセカンダリクラスターから転送された接続でアクティビティを終了するまで待機するミリ秒数。この期間を超えてセッションがアイドル状態のままである場合、Aurora はセッションを終了します。0 の値は、タイムアウトを無効にします。

Aurora PostgreSQL での書き込み転送の Amazon CloudWatch メトリクス

次の Amazon CloudWatch メトリクスは、1 つ以上のセカンダリクラスターで書き込み転送を使用する場合、プライマリクラスターに適用されます。これらのメトリクスはすべて、プライマリクラスターのライター DB インスタンスで測定されます。

CloudWatch メトリクス	単位と説明
AuroraForwardingWriterDMLThroughput	1 秒あたりのカウント数。この書き込み DB インスタンスによって 1 秒間に処理される、転送された DML ステートメントの数。
AuroraForwardingWriterOpenSessions	カウント。転送されたクエリを処理しているこのライター DB インスタンスで開いているセッションの数。
AuroraForwardingWriterTotalSessions	カウント。このライター DB インスタンスで転送されたセッションの合計数。

次の CloudWatch メトリクスは、各セカンダリクラスターに適用されます。これらのメトリクスは、書き込み転送が有効になっているセカンダリクラスターのリーダー DB インスタンスで測定されます。

CloudWatch メトリクス	単位と説明
AuroraForwardingReplicaCommitThroughput	1 秒あたりのカウント数。このレプリカが 1 秒あたりに転送したセッションのコミット数。

CloudWatch メトリクス	単位と説明
AuroraForwardingReplicaDMLLatency	ミリ秒。レプリカ上で転送された DML の平均応答時間 (ミリ秒)。
AuroraForwardingReplicaDMLThroughput	1 秒あたりのカウント数。このレプリカで 1 秒あたりに処理された転送 DML ステートメントの数。
AuroraForwardingReplicaErrorSessionsLimit	カウント。最大接続または最大書き込み転送接続の制限に達したためにプライマリクラスターが拒否したセッションの数。
AuroraForwardingReplicaOpenSessions	カウント。リーダー DB インスタンスで書き込み転送を使用しているセッションの数。
AuroraForwardingReplicaReadWaitLatency	ミリ秒。レプリカがプライマリクラスターの LSN と一致するまで待機する平均待機時間 (ミリ秒単位)。リーダー DB インスタンスが待機する程度は、`apg_write_forward.consistency_mode` 設定によって異なります。この設定についての情報は、「the section called “Aurora PostgreSQL での書き込み転送の設定パラメータ”」を参照してください。

Aurora PostgreSQL での書き込み転送のイベントを待機する

Aurora PostgreSQL で書き込み転送を使用すると、Amazon Aurora は次の待機イベントを生成します。

トピック

- IPC:AuroraWriteForwardConnect
- IPC:AuroraWriteForwardConsistencyPoint
- IPC:AuroraWriteForwardExecute
- IPC:AuroraWriteForwardGetGlobalConsistencyPoint
- IPC:AuroraWriteForwardXactAbort

- IPC:AuroraWriteForwardXactCommit
- IPC:AuroraWriteForwardXactStart

IPC:AuroraWriteForwardConnect

`IPC:AuroraWriteForwardConnect` イベントは、セカンダリ DB クラスターのバックエンドプロセスが、プライマリ DB クラスターのライターノードへの接続が開かれるのを待っているときに発生します。

待機時間が増加する原因の可能性

このイベントは、セカンダリージョンのリーダーノードからプライマリ DB クラスターのライターノードへの接続試行回数が増えるにつれて増加します。

アクション

セカンダリノードからプライマリリージョンのライターノードへの同時接続数を減らします。

IPC:AuroraWriteForwardConsistencyPoint

`IPC:AuroraWriteForwardConsistencyPoint` イベントは、転送された書き込みオペレーションの結果が現在のリージョンにレプリケートされるまでセカンダリ DB クラスターのノードからのクエリが待機する時間を記述します。このイベントは、セッションレベルのパラメータ `apg_write_forward.consistency_mode` が次のいずれかに設定されている場合にのみ生成されます。

- `SESSION` — セカンダリノードのクエリは、そのセッションで行われたすべての変更の結果が終わるまで待機します。
- `GLOBAL` — セカンダリノード上のクエリは、そのセッションで行われた変更の結果に加えて、グローバルクラスター内のプライマリリージョンと他のセカンダリリージョンの両方でコミットされたすべての変更の結果を待ちます。

`apg_write_forward.consistency_mode` パラメータの設定の詳細については、「the section called “Aurora PostgreSQL での書き込み転送の設定パラメータ”」をご参照ください。

待機時間が増加する原因の可能性

待機時間が長くなる一般的な原因には以下のものがあります。

- Amazon CloudWatch `ReplicaLag` メトリクスで測定すると、レプリカの遅延が増加しました。このメトリクスの詳細については、「Aurora PostgreSQL レプリケーションのモニタリング」を参照してください。
- プライマリリージョンのライターノードまたはセカンダリノードの負荷が増加した。

アクション

アプリケーションの要件に応じて、整合性モードを変更します。

IPC:AuroraWriteForwardExecute

`IPC:AuroraWriteForwardExecute` イベントは、セカンダリ DB クラスターのバックエンドプロセスが、転送されたクエリを完了し、プライマリ DB クラスターのライターノードからの結果を取得するのを待っているときに発生します。

待機時間が増加する原因の可能性

待機時間の増加の一般的な原因としては、以下が挙げられます。

- プライマリリージョンのライターノードから多数の行をフェッチしている。
- セカンダリノードとプライマリリージョンのライターノード間のネットワークレイテンシーが長くなると、セカンダリノードがライターノードからデータを受信するまでにかかる時間が長くなります。
- セカンダリノードの負荷が増加すると、セカンダリノードからプライマリリージョンのライターノードへのクエリリクエストの送信が遅れる可能性があります。
- プライマリリージョンのライターノードの負荷が増加すると、ライターノードからセカンダリノードへのデータの送信が遅れる可能性があります。

アクション

待機イベントの原因に応じたさまざまなアクションをお勧めします。

- 必要なデータのみを取得するようにクエリを最適化します。
- データ操作言語 (DML) オペレーションを最適化し、必要なデータのみを変更します。
- セカンダリノードまたはプライマリリージョンのライターノードが CPU またはネットワーク帯域幅によって制約されている場合は、CPU キャパシティまたはネットワーク帯域幅の大きいインスタンスタイプに変更することを検討してください。

IPC:AuroraWriteForwardGetGlobalConsistencyPoint

`IPC:AuroraWriteForwardGetGlobalConsistencyPoint` イベントは、GLOBAL 整合性モードを使用しているセカンダリ DB クラスター上のバックエンドプロセスが、クエリを実行する前に、ライターノードからグローバル整合性ポイントを取得するのを待っているときに発生します。

待機時間が増加する原因の可能性

待機時間の増加の一般的な原因としては、以下が挙げられます。

- セカンダリノードとプライマリリージョンのライターノード間のネットワークレイテンシーが長くなると、セカンダリノードがライターノードからデータを受信するまでにかかる時間が長くなります。
- セカンダリノードの負荷が増加すると、セカンダリノードからプライマリリージョンのライターノードへのクエリリクエストの送信が遅れる可能性があります。
- プライマリリージョンのライターノードの負荷が増加すると、ライターノードからセカンダリノードへのデータの送信が遅れる可能性があります。

アクション

待機イベントの原因に応じたさまざまなアクションをお勧めします。

- アプリケーションの要件に応じて、整合性モードを変更します。
- セカンダリノードまたはプライマリリージョンのライターノードが CPU またはネットワーク帯域幅によって制約されている場合は、CPU キャパシティまたはネットワーク帯域幅の大きいインスタンスタイプに変更することを検討してください。

IPC:AuroraWriteForwardXactAbort

`IPC:AuroraWriteForwardXactAbort` イベントは、セカンダリ DB クラスターのバックエンドプロセスがリモートクリーンアップクエリの結果を待っているときに発生します。クリーンアップクエリは、書き込み転送されたトランザクションが中止された後にプロセスを適切な状態に戻すために発行されます。Amazon Aurora は、エラーが見つかったか、ユーザーが明示的な `ABORT` コマンドを発行したか、実行中のクエリをキャンセルしたためにそれらを実行します。

待機時間が増加する原因の可能性

待機時間の増加の一般的な原因としては、以下が挙げられます。

- セカンダリノードとプライマリリージョンのライターノード間のネットワークレイテンシーが長くなると、セカンダリノードがライターノードからデータを受信するまでにかかる時間が長くなります。
- セカンダリノードの負荷が増加すると、セカンダリノードからプライマリリージョンのライターノードへのクリーンナップクエリリクエストの送信が遅れる可能性があります。
- プライマリリージョンのライターノードの負荷が増加すると、ライターノードからセカンダリノードへのデータの送信が遅れる可能性があります。

アクション

待機イベントの原因に応じたさまざまなアクションをお勧めします。

- 中断されたトランザクションの原因を調査します。
- セカンダリノードまたはプライマリリージョンのライターノードが CPU またはネットワーク帯域幅によって制約されている場合は、CPU キャパシティまたはネットワーク帯域幅の大きいインスタンスタイプに変更することを検討してください。

IPC:AuroraWriteForwardXactCommit

`IPC:AuroraWriteForwardXactCommit` イベントは、セカンダリ DB クラスターのバックエンドプロセスが、転送されたコミットトランザクションコマンドの結果を待っているときに発生します。

待機時間が増加する原因の可能性

待機時間の増加の一般的な原因としては、以下が挙げられます。

- セカンダリノードとプライマリリージョンのライターノード間のネットワークレイテンシーが長くなると、セカンダリノードがライターノードからデータを受信するまでにかかる時間が長くなります。
- セカンダリノードの負荷が増加すると、セカンダリノードからプライマリリージョンのライターノードへのクエリリクエストの送信が遅れる可能性があります。
- プライマリリージョンのライターノードの負荷が増加すると、ライターノードからセカンダリノードへのデータの送信が遅れる可能性があります。

アクション

セカンダリノードまたはプライマリリージョンのライターノードが CPU またはネットワーク帯域幅によって制約されている場合は、CPU キャパシティまたはネットワーク帯域幅の大きいインスタンスタイプに変更することを検討してください。

IPC:AuroraWriteForwardXactStart

`IPC:AuroraWriteForwardXactStart` イベントは、セカンダリ DB クラスターのバックエンドプロセスが、転送された開始トランザクションコマンドの結果を待っているときに発生します。

待機時間が増加する原因の可能性

待機時間の増加の一般的な原因としては、以下が挙げられます。

- セカンダリノードとプライマリリージョンのライターノード間のネットワークレイテンシーが長くなると、セカンダリノードがライターノードからデータを受信するまでにかかる時間が長くなります。
- セカンダリノードの負荷が増加すると、セカンダリノードからプライマリリージョンのライターノードへのクエリリクエストの送信が遅れる可能性があります。
- プライマリリージョンのライターノードの負荷が増加すると、ライターノードからセカンダリノードへのデータの送信が遅れる可能性があります。

アクション

セカンダリノードまたはプライマリリージョンのライターノードが CPU またはネットワーク帯域幅によって制約されている場合は、CPU キャパシティまたはネットワーク帯域幅の大きいインスタンスタイプに変更することを検討してください。

Amazon Aurora Global Database でスイッチオーバーまたはフェイルオーバーを使用する

Aurora グローバルデータベースは、単一の AWS リージョン で Aurora DB クラスターによって提供される標準高可用性よりも、事業継続性とディザスタリカバリ (BCDR) 保護を強化します。Aurora グローバルデータベースを使用すれば、実際の地域災害に備えて計画を立てて復旧したり、サービスレベルの停止を迅速に完了したりできます。災害からの復旧は、通常、次の 2 つのビジネス目標によって推進されます。

- 目標復旧時間 (RTO) - 災害またはサービス障害後にシステムが稼働状態に戻るまでにかかる時間。つまり、RTO はダウンタイムを測定します。Aurora Global Database の場合、RTO は分単位で行えます。
- 目標復旧時点 (RPO) - 災害またはサービス障害後に損失する可能性があるデータの量 (時間単位)。このデータ損失は、通常、非同期レプリケーションの遅延が原因です。Aurora Global Database の場合、RPO (目標復旧時点) は通常、秒単位で測定されます。Aurora PostgreSQL - ベースのグローバルデータベースでは、`rds.global_db_rpo` パラメータを使用して RPO の上限を設定および追跡できますが、そうすると、プライマリクラスターの読み取りノードでのトランザクション処理に影響を与える可能性があります。詳細については、「Aurora PostgreSQL- ベースのグローバルデータベースの RPO (目標復旧時点) 管理」を参照してください。

Aurora グローバルデータベースの切り替えまたはフェイルオーバーには、グローバルデータベースのセカンダリリージョンのいずれかにある DB クラスターをプライマリ DB クラスターに昇格させる必要があります。「リージョン障害」という用語は、さまざまな障害シナリオを表すためによく使用されます。最悪のシナリオとしては、数百平方マイルに及ぶ壊滅的な事象による広範囲にわたる停電が考えられます。しかし、ほとんどの停電ははるかに局所的で、影響を受けるのはクラウドサービスや顧客システムのごく一部に限られます。障害の全容を検討して、クロスリージョンフェイルオーバーが適切な解決策であることを確認し、状況に適したフェイルオーバー方法を選択してください。フェイルオーバーとスイッチオーバーのどちらを使用するかは、具体的な停止シナリオによって異なります。

- フェイルオーバー — このアプローチを使用して、計画外のシステム停止から回復します。この方法では、Aurora グローバルデータベース内のセカンダリ DB クラスターの 1 つにクロスリージョンフェイルオーバーを実行します。このアプローチの RPO は、通常、秒単位で測定される 0 以外の値です。障害発生時のデータ損失の量は、AWS リージョン 全体の Aurora グローバルデータベースレプリケーションの遅延によって異なります。詳細については、「予期しない停止からの Amazon Aurora Global Database の復旧」を参照してください。
- スイッチオーバー— この操作は、以前は「マネージドプランニングフェイルオーバー」と呼ばれていました。このアプローチは、運用メンテナンス、その他の計画された運用手順など、管理されたシナリオに使用してください。この機能は、他の変更を行う前にセカンダリ DB クラスターとプライマリクラスターを同期するため、RPO は 0 (データの損失なし) になります。詳細については、「Amazon Aurora Global Database に対するスイッチオーバーの実行」を参照してください。

> **Note**
>
> ヘッドレスセカンダリ Aurora DB クラスターへのスイッチオーバーまたはフェイルオーバーを行う場合は、まず DB インスタンスをそのクラスターに追加する必要があります。ヘッドレス DB クラスターの詳細については、「セカンダリリージョンでのヘッドレス Aurora DB クラスターの作成」を参照してください。

トピック

- 予期しない停止からの Amazon Aurora Global Database の復旧
- Amazon Aurora Global Database に対するスイッチオーバーの実行
- Aurora PostgreSQL- ベースのグローバルデータベースの RPO (目標復旧時点) 管理

予期しない停止からの Amazon Aurora Global Database の復旧

ごくまれに、Aurora Global Database のプライマリ AWS リージョン で予期しない停止が発生することがあります。この場合、プライマリ Aurora DB クラスターとその読み取りノードを使用できなくなり、プライマリとセカンダリ DB クラスター間のレプリケーションが停止します。ダウンタイム (RTO) とデータ損失 (RPO) の両方を最小限に抑えるため、迅速に作業を行ってリージョン間のフェイルオーバーを実行できます。

ディザスタリカバリ時のフェイルオーバーには 2 つの方法があります。

- マネージドフェイルオーバー — ディザスタリカバリにはこの方法が推奨されます。この方法を使用すると、Aurora は古いプライマリリージョンが再び使用可能になったときに、セカンダリリージョンとしてグローバルデータベースに自動的に追加します。これにより、グローバルクラスターの元のトポロジーが維持されます。この方法を使用する手順については、「Aurora Global Database のマネージドフェイルオーバーを実行する」を参照してください。
- 手動フェイルオーバー — この代替方法は、プライマリリージョンとセカンダリリージョンで互換性のないエンジンバージョンが稼働している場合など、マネージドフェイルオーバーを使用できない場合に使用できます。この方法を使用する手順については、「Aurora Global Database のマニュアルフェイルオーバーを実行する」を参照してください。

> **Important**
>
> どちらのフェイルオーバー方法でも、フェイルオーバーイベントが発生する前に選択したセカンダリにレプリケートされていなかった書き込みトランザクションデータが失われる可能性があります。ただし、選択したセカンダリ DB クラスター上の DB インスタンスをプライマリライター DB インスタンスに昇格させるリカバリプロセスにより、データがトランザクション的に一貫した状態になることが保証されます。

Aurora Global Database のマネージドフェイルオーバーを実行する

このアプローチは、真の地域災害やサービスレベルの全面的な停止が発生した場合でも事業を継続できるようにするためのものです。

マネージドフェイルオーバー中、プライマリクラスターは選択したセカンダリリージョンにフェイルオーバーされ、Aurora Global Database の既存のレプリケーショントポロジが維持されます。選択したセカンダリクラスターは、読み取り専用ノードの 1 つを完全な読み取り状態に昇格します。このステップにより、クラスターがプライマリクラスターのロールを引き受けることができます。クラスターが新しいロールを引き受ける間、データベースは短時間使用できなくなります。古いプライマリクラスターから選択したセカンダリクラスターにレプリケートされなかったデータは、このセカンダリクラスターが新しいプライマリクラスターになると失われます。

> **Note**
>
> プライマリ DB クラスターとセカンダリ DB クラスターが同じメジャー、マイナー、パッチレベルのエンジンバージョンである場合にのみ、Aurora グローバルデータベースでマネージドクロスリージョンデータベースフェイルオーバーを実行できます。ただし、パッチレベルはマイナーエンジンバージョンによって異なる場合があります。詳細については、「マネージドクロスリージョンスイッチオーバーまたはフェイルオーバーに対するパッチレベルの互換性」を参照してください。ご使用のエンジンバージョンに互換性がない場合は、Aurora Global Database のマニュアルフェイルオーバーを実行する の手順に従ってフェイルオーバーを手動で実行できます。

データの損失を最小限に抑えるため、この機能を使用する前に次のことを行うことをお勧めします。

- アプリケーションをオフラインにして、Aurora Global Database のプライマリクラスターへの書き込みが送信されないようにします。

- Aurora Global Database 内のすべてのセカンダリ Aurora DB クラスターのラグタイムを確認します。レプリケーションの遅延が最も少ないセカンダリリージョンを選択すると、現在障害が発生しているプライマリリージョンでのデータ損失を最小限に抑えることができます。すべての Aurora PostgreSQL ベースのグローバルデータベース、およびエンジンバージョン 3.04.0 以降、または 2.12.0 以降の Aurora MySQL ベースのグローバルデータベースについては、Amazon CloudWatch を使用してすべてのセカンダリ DB クラスターの `AuroraGlobalDBRPOLag` メトリクスを確認します。Aurora MySQL ベースのグローバルデータベースの下位マイナーバージョンについては、代わりに `AuroraGlobalDBReplicationLag` メトリクスを確認します。これらのメトリクスは、セカンダリクラスターがプライマリ DB クラスターに対してどのくらい遅れているか (ミリ秒単位) を示します。

  Aurora 向け CloudWatch メトリクスの詳細については、Amazon Aurora のクラスターレベルのメトリクス を参照してください。

マネージドフェイルオーバー中、選択したセカンダリ DB クラスターは、プライマリとして新しいロールに昇格されます。ただし、プライマリ DB クラスターのさまざまな設定オプションは継承されません。構成の不一致は、パフォーマンスの問題、ワークロードの非互換性、およびその他の異常な動作につながる可能性があります。このような問題を回避するには、Aurora Global Database クラスター間の次の違いを解決することをお勧めします。

- 新しいプライマリの Aurora DB クラスターパラメータグループの構成 (必要な場合) - Aurora Global Database の Aurora クラスターごとに Aurora DB クラスターパラメータグループを個別に設定できます。そのため、セカンダリ DB クラスターを昇格してプライマリロールを引き継ぐ場合、セカンダリからのパラメータグループは、プライマリとは異なる設定になることがあります。その場合は、プロモートされたセカンダリ DB クラスターのパラメータグループを、プライマリクラスターの設定に適合するように変更します。この方法については、「Aurora Global Database のパラメータの修正」を参照してください。
- モニタリングツールとオプション (Amazon CloudWatch Events やアラームなど) - グローバルデータベースに必要なログ機能、アラームなどを使用して、昇格された DB クラスターの設定を行います。パラメータグループと同様に、フェイルオーバープロセス中にこれらの機能の設定がプライマリから継承されることはありません。レプリケーションラグなどの一部の CloudWatch メトリクスは、セカンダリリージョンのみで使用できます。そのため、フェイルオーバーによってメトリクスの表示方法やアラームの設定方法が変わり、定義済みのダッシュボードを変更する必要が生じる場合があります。Aurora DB クラスターとモニタリングの詳細については、Overview of monitoring Amazon Aurora を参照してください。

- 他の AWS サービスとの統合を設定する - Aurora Global Database を AWS サービス (AWS Secrets Manager、AWS Identity and Access Management、Amazon S3、AWS Lambda など) と統合する場合は、それぞれに応じた設定を行う必要があります。IAM、Amazon S3、Lambda との Aurora Global Database の統合の詳細については、「Amazon Aurora Global Database を他の AWS サービスと併用する」を参照してください。Secrets Manager の詳細については、「AWS リージョン 間で AWS Secrets Manager のシークレットのレプリケーションを自動化する方法」を参照してください。

通常、選択したセカンダリクラスターが数分以内に主要な役割を引き継ぎます。新しいプライマリリージョンのライターノードが使用可能になり次第、アプリケーションをそのライターノードに接続してワークロードを再開できます。Aurora が新しい主クラスターをプロモートすると、追加のセカンダリリージョンクラスターはすべて自動的に再構築されます。

Aurora グローバルデータベースは非同期レプリケーションを使用するため、各セカンダリリージョンのレプリケーションラグは異なる場合があります。Aurora はこれらのセカンダリリージョンを新しいプライマリリージョンクラスターとまったく同じポイントインタイムデータを持つように再構築します。ストレージボリュームのサイズとリージョン間の距離によっては、再構築タスクが完了するまでに数分から数時間かかることがあります。セカンダリージョンのクラスターが新しいプライマリリージョンからの再構築を完了すると、読み取りアクセスが可能になります。

新しいプライマリライターが昇格して使用可能になると、新しいプライマリリージョンのクラスターは Aurora グローバルデータベースの読み取りおよび書き込み操作を処理できるようになります。アプリケーションのエンドポイントを変更して、新しいエンドポイントを使用してください。Aurora Global Database の作成時に指定された名前を受け入れた場合は、アプリケーションの昇格されたクラスターのエンドポイント文字列から `-ro` を削除することで、エンドポイントを変更できます。

例えば、セカンダリクラスターのエンドポイント `my-global.cluster-**ro**-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` は、そのクラスターがプライマリに昇格したときに `my-global.cluster-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` になります。

RDS Proxy を使用する場合、アプリケーションの書き込みオペレーションを、新しいプライマリクラスターに関連付けられているプロキシの適切な読み取り/書き込みエンドポイントにリダイレクトします。このプロキシエンドポイントは、デフォルトのエンドポイントでも、カスタムの読み取り/書き込みエンドポイントでもかまいません。詳細については、「RDS Proxy エンドポイントとグローバルデータベースの連携について」を参照してください。

グローバルデータベースクラスターの元のトポロジを復元するために、Aurora は古いプライマリリージョンの可用性を監視します。そのリージョンが正常になり、再び使用可能になるとすぐ

に、Aurora はそのリージョンをセカンダリリージョンとしてグローバルクラスターに自動的に追加します。古いプライマリリージョンに新しいストレージボリュームを作成する前に、Aurora は障害発生時点で古いストレージボリュームのスナップショットを作成しようとします。これにより、欠落しているデータを回復することができます。この操作が成功すると、Aurora は「rds:unplanned-global-failover-*name-of-old-primary-DB-cluster-timestamp*」というこのスナップショットを AWS Management Console のスナップショットセクションに入れます。このスナップショットは、DescribeDBClusterSnapshots API オペレーションによって返される情報にも一覧表示されています。

> **Note**
>
> 古いストレージボリュームのスナップショットは、古い主クラスターに設定されたバックアップ保持期間の対象となるシステムスナップショットです。このスナップショットを保存期間外に保存するには、スナップショットをコピーして手動スナップショットとして保存できます。価格などのスナップショットのコピーの詳細については、「DB クラスターのスナップショットのコピー」を参照してください。

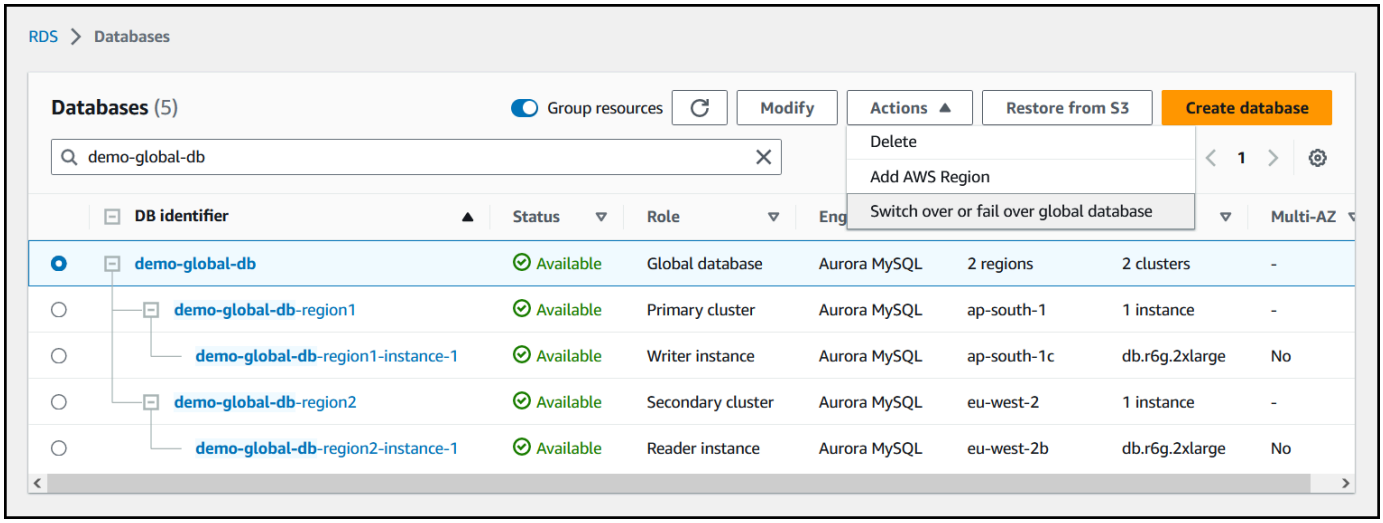
元のトポロジが復元されたら、ビジネスとワークロードにとって最も都合のよいときにスイッチオーバー操作を実行することで、グローバルデータベースを元のプライマリリージョンにフェイルバックできます。これを行うには、「Amazon Aurora Global Database に対するスイッチオーバーの実行」の手順を実行します。

Aurora Global Database をフェイルオーバーするには AWS Management Console、AWS CLI、または RDS API を使用します。

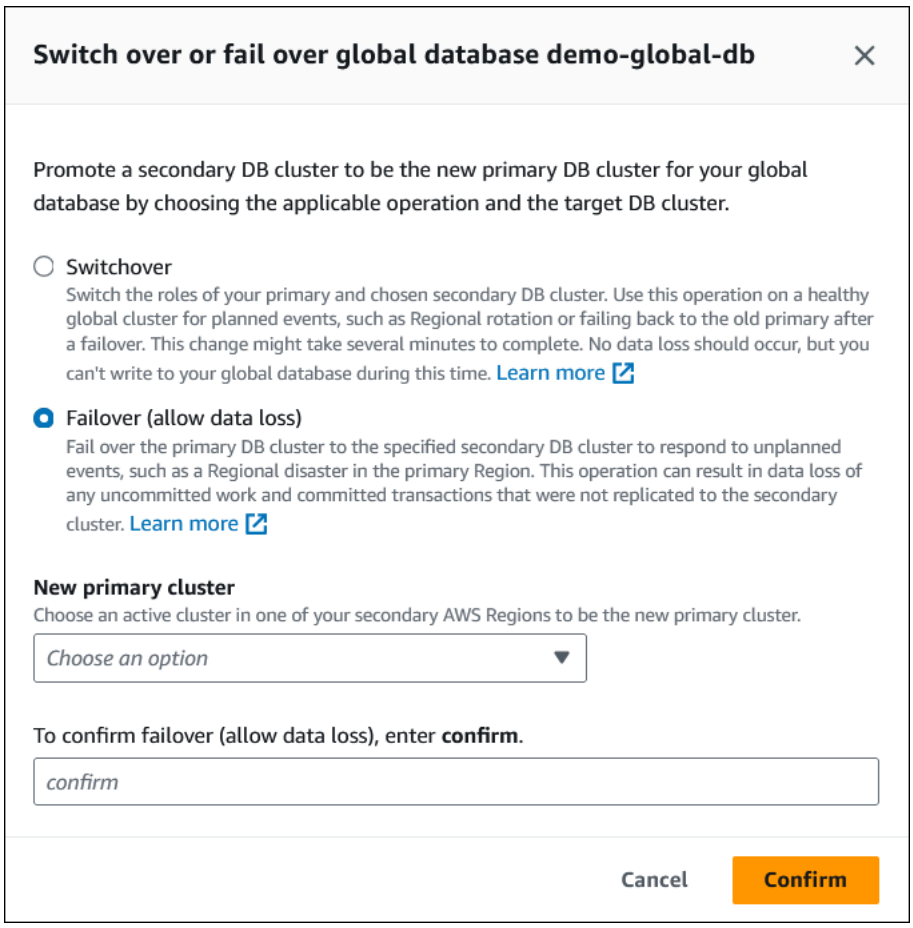
コンソール

Aurora グローバルデータベースでマネージドフェイルオーバーを実行するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. [Databases (データベース)] を選択し、フェイルオーバーしたい Aurora Global Database を見つけます。
3. [アクション] メニューから [グローバルデータベースのスイッチオーバーまたはフェイルオーバー] を選択します。



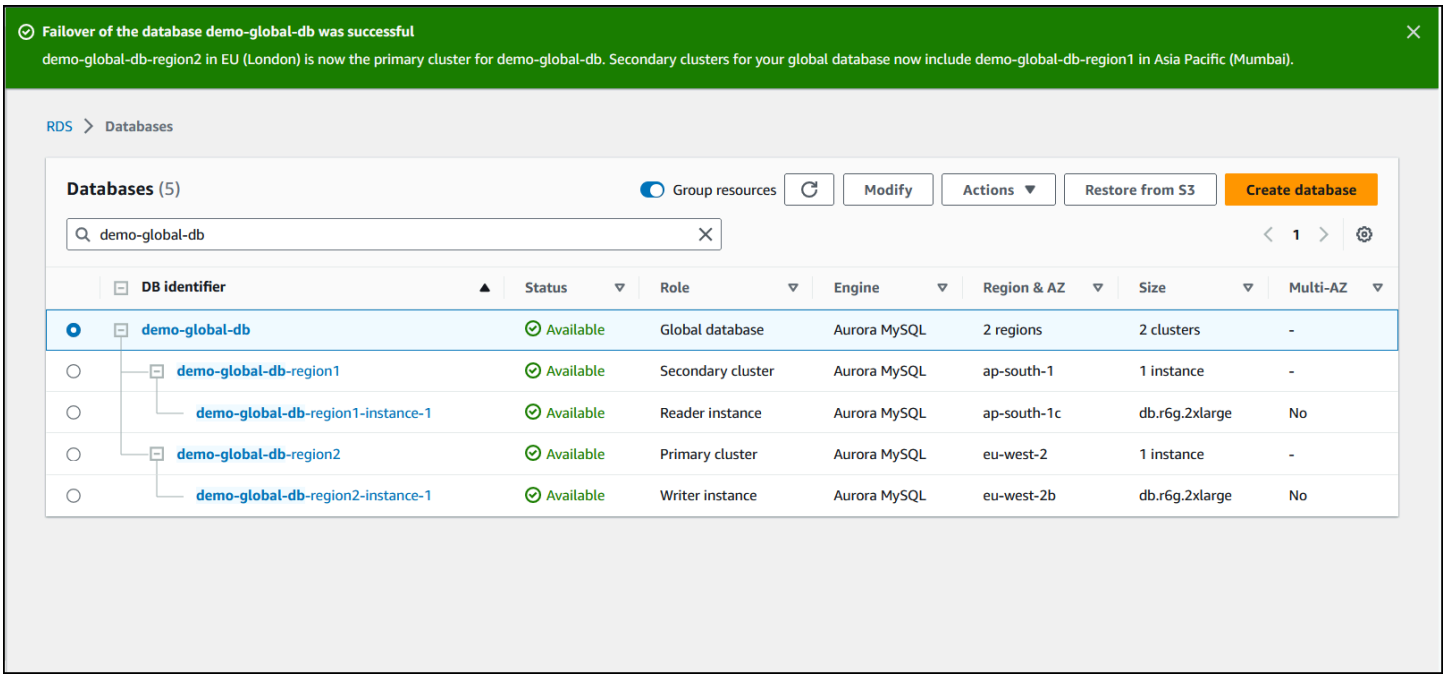
4. [フェイルオーバー (データ損失を許可)] を選択します。



5. 新しいプライマリクラスタ の場合、セカンダリ AWS リージョン の 1 つで新しいプライマリクラスターになるアクティブなクラスターを選択してください。

6. **`confirm`** を入力して [確認] を選択します。

フェイルオーバーが完了すると、次の画像に示すように、[データベース] リストから Aurora DB クラスターとその現在の状態を確認できます。



AWS CLI

Aurora グローバルデータベースでマネージドフェイルオーバーを実行するには

`failover-global-cluster` CLI コマンドを使用して、Aurora Global Database をフェイルオーバーします。コマンドを使用して、次のパラメータ値を渡します。

- `--region` — Aurora グローバルデータベースの新しいプライマリにしたいセカンダリ DB クラスターが DB クラスターで実行されている AWS リージョン を指定します。
- `--global-cluster-identifier` - Aurora Global Database の名前を指定します。
- `--target-db-cluster-identifier` - Aurora Global Database の新しいプライマリに昇格する Aurora DB クラスターの Amazon リソースネーム (ARN) を指定します。
- `--allow-data-loss` — これをスイッチオーバー操作ではなくフェイルオーバー操作に明示的に設定します。非同期レプリケーションコンポーネントがレプリケートされたすべてのデータをセカンダリリージョンに送信していない場合、フェイルオーバー操作によってデータの一部が失われる可能性があります。

Linux、macOS、Unix の場合:

```
aws rds --region region_of_selected_secondary \
```

```
    failover-global-cluster --global-cluster-identifier global_database_id \
    --target-db-cluster-identifier arn_of_secondary_to_promote \
    --allow-data-loss
```

Windows の場合:

```
aws rds --region region_of_selected_secondary ^
    failover-global-cluster --global-cluster-identifier global_database_id ^
    --target-db-cluster-identifier arn_of_secondary_to_promote ^
    --allow-data-loss
```

RDS API

Aurora Global Database をフェイルオーバーするには、FailoverGlobalCluster API オペレーションを実行します。

Aurora Global Database のマニュアルフェイルオーバーを実行する

シナリオによっては、マネージドフェイルオーバープロセスを使用できない場合があります。一例として、プライマリ DB クラスターとセカンダリ DB クラスターで互換性のあるエンジンバージョンが稼働していない場合が挙げられます。この場合、このマニュアルプロセスに従って、グローバルデータベースをターゲットセカンダリリージョンにフェイルオーバーできます。

> **Tip**
>
> このプロセスを理解してから使用することをお勧めします。リージョン全体にわたる問題の初期の兆候が見られたら、すぐに計画を進める準備をしてください。Amazon CloudWatch を定期的に使用してセカンダリクラスターのラグタイムを追跡することで、レプリケーションラグが最も少ないセカンダリリージョンを特定できます。実際に発生する前に計画をテストして、手順が完全かつ正確であることや、スタッフがディザスタリカバリフェイルオーバーの実行に習熟していることを確認します。

プライマリリージョンで予期しない停止が発生した後にセカンダリクラスターにマニュアルでフェイルオーバーするには

1. 停止している AWS リージョン のプライマリ Aurora DB クラスターに対する、DML ステートメントおよびその他の書き込みオペレーションの発行を停止します。
2. 新しいプライマリ DB クラスターとして使用するために、セカンダリ AWS リージョン の Aurora DB クラスターを指定します。Aurora Global Database に 2 つ以上のセカンダリ AWS

リージョン がある場合は、レプリケーション遅延が最も少ないセカンダリクラスターを選択します。

3. 選択したセカンダリ DB クラスターを Aurora Global Database からデタッチします。

   Aurora Global Database からセカンダリ DB クラスターを削除すると、プライマリからこのセカンダリへのレプリケーションが直ちに停止され、完全な読み取り/書き込み機能を備えたスタンドアロンのプロビジョニングされた Aurora DB クラスターに昇格します。停止しているリージョン内のプライマリクラスターに関連付けられたその他のセカンダリ Aurora DB クラスターは引き続き利用可能で、アプリケーションからの呼び出しを受け付けることができます。また、リソースを使用することになります。Aurora Global Database を再作成しているので、次の手順で新しい Aurora Global Database を作成する前に、他のセカンダリ DB クラスターを削除します。これにより、Aurora Global Database 内の DB クラスター間のデータの不整合 (スプリットブレイン問題) が回避されます。

   アタッチ解除の詳細なステップについては、Amazon Aurora Global Database からのクラスターの削除 を参照してください。

4. 新しいエンドポイントを使用して、このスタンドアロン Aurora DB クラスターにすべての書き込み操作を送信するように、アプリケーションを再構成します。Aurora Global Database の作成時に指定された名前を受け入れた場合は、アプリケーション内のクラスターのエンドポイント文字列から `-ro` を削除することで、エンドポイントを変更できます。

   例えば、セカンダリクラスターのエンドポイント `my-global.cluster-**ro**-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` は、そのクラスターが Aurora Global Database からデタッチされたときに `my-global.cluster-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` になります。

   この Aurora DB クラスターは、次のステップでリージョンを追加すると、新しい Aurora Global Database のプライマリクラスターになります。

   RDS Proxy を使用する場合、アプリケーションの書き込みオペレーションを、新しいプライマリクラスターに関連付けられているプロキシの適切な読み取り/書き込みエンドポイントにリダイレクトします。このプロキシエンドポイントは、デフォルトのエンドポイントでも、カスタムの読み取り/書き込みエンドポイントでもかまいません。詳細については、「RDS Proxy エンドポイントとグローバルデータベースの連携について」を参照してください。

5. DB クラスターに AWS リージョン を追加します。これを行うと、プライマリからセカンダリへのレプリケーションプロセスがスタートされます。リージョンを追加する詳細なステップについては、AWS リージョン の Amazon Aurora Global Database への追加 を参照してください。

6. 必要に応じて、AWS リージョン を追加して、アプリケーションのサポートに必要なトポロジを再作成します。

これらの変更を行う前、最中、および後に、アプリケーションの書き込みが正しい Aurora クラスターに送信されていることを確認してください。これにより、Aurora Global Database 内の DB クラスター間のデータの不整合 (スプリットブレイン問題) が回避されます。

AWS リージョン の停止に対して再設定を実行した場合、停止状態が解消された後に、AWS リージョン をプライマリに戻すことができます。このためには、古い AWS リージョン を新しいグローバルデータベースに追加し、スイッチオーバープロセスを使用してそのロールを切り替えます。Aurora Global Database では、スイッチオーバーをサポートする Aurora PostgreSQL または Aurora MySQL のバージョンを使用する必要があります。詳細については、「Amazon Aurora Global Database に対するスイッチオーバーの実行」を参照してください。

Amazon Aurora Global Database に対するスイッチオーバーの実行

> **Note**
>
> スイッチオーバーは、以前「管理型計画フェイルオーバー」と呼ばれていました。

スイッチオーバーを使用すると、プライマリクラスターのリージョンを定期的に変更できます。この機能は、運用メンテナンス、その他の計画された運用手順など、管理されたシナリオを対象としています。

スイッチオーバーを使用する一般的なユースケースは 3 つあります。

- 特定の業界に課せられる「リージョナルローテーション」要件向け。たとえば、金融サービス規制では、ディザスタリカバリ手順が定期的に実施されるように、Tier-0 システムを別の地域に数か月間切り替えることが求められる場合があります。
- マルチリージョンの「follow-the-sun」アプリケーション向け。たとえば、ある企業が、さまざまなタイムゾーンの営業時間に基づいて、さまざまなリージョンで低レイテンシーの書き込みを提供したいとします。
- データ損失ゼロの方法として、フェイルオーバー後に元のプライマリリージョンにフェイルバックします。

> **Note**
>
> スイッチオーバーは、正常な Aurora グローバルデータベースで使用するように設計されています。予期しないシステム停止から回復するには、予期しない停止からの Amazon Aurora Global Database の復旧 の該当する手順に従ってください。
> スイッチオーバーを実行するには、エンジンのバージョンに応じて、パッチレベルを含め、ターゲットのセカンダリ DB クラスターがプライマリとまったく同じバージョンを実行している必要があります。詳細については、「マネージドクロスリージョンスイッチオーバーまたはフェイルオーバーに対するパッチレベルの互換性」を参照してください。スイッチオーバーを開始する前に、グローバルクラスター内のエンジンバージョンをチェックして、マネージドクロスリージョンスイッチオーバーをサポートしていることを確認し、必要に応じてアップグレードしてください。

スイッチオーバー中、グローバルデータベースの既存のレプリケーショントポロジを維持しながら、Aurora がプライマリクラスターを選択したセカンダリリージョンにスイッチオーバーします。スイッチオーバープロセスを開始する前に、Aurora はすべてのセカンダリリージョンクラスターがプライマリリージョンクラスターと完全に同期されるまで待ちます。次に、プライマリリージョンの DB クラスターは読み取り専用になり、選択したセカンダリ DB クラスターは、読み取り専用ノードの 1 つを、フルライターステータスに昇格させます。このノードをライターに昇格させると、そのセカンダリクラスターがプライマリクラスターの役割を引き受けることができます。プロセスのスタート時にすべてのセカンダリクラスターがプライマリと同期されているため、新しいプライマリは、データを失うことなく、Aurora Global Database の操作を続行します。プライマリクラスターと選択したセカンダリクラスターが新しいロールを引き受ける間、短時間データベースが使用できなくなります。

アプリケーションの可用性を最適化するには、この機能を使用する前に、次の操作を行うことをお勧めします。

- この操作は、ピーク時以外に、またはプライマリ DB クラスターへの書き込みが最小限である別の時間に実行します。
- アプリケーションをオフラインにして、Aurora Global Database のプライマリクラスターへの書き込みが送信されないようにします。
- Aurora Global Database 内のすべてのセカンダリ Aurora DB クラスターのラグタイムを確認します。すべての Aurora PostgreSQL ベースのグローバルデータベース、およびエンジンバージョン 3.04.0 以降、または 2.12.0 以降の Aurora MySQL ベースのグローバルデータベースについては、Amazon CloudWatch を使用してすべてのセカンダリ DB クラスターの

`AuroraGlobalDBRPOLag` メトリクスを確認します。Aurora MySQL ベースのグローバルデータベースの下位マイナーバージョンについては、代わりに `AuroraGlobalDBReplicationLag` メトリクスを確認します。これらのメトリクスは、セカンダリクラスターがプライマリ DB クラスターに対してどのくらい遅れているか (ミリ秒単位) を示します。この値は、Aurora がスイッチオーバーを完了するまでにかかる時間に正比例します。したがって、遅延値が大きいほど、スイッチオーバーにかかる時間は長くなります。

Aurora 向け CloudWatch メトリクスの詳細については、Amazon Aurora のクラスターレベルのメトリクス を参照してください。

スイッチオーバー中、選択したセカンダリ DB クラスターは、プライマリとして新しいロールに昇格されます。ただし、プライマリ DB クラスターのさまざまな設定オプションは継承されません。構成の不一致は、パフォーマンスの問題、ワークロードの非互換性、およびその他の異常な動作につながる可能性があります。このような問題を回避するには、Aurora Global Database クラスター間の次の違いを解決することをお勧めします。

- 新しいプライマリの Aurora DB クラスターパラメータグループの構成 (必要な場合) - Aurora Global Database の Aurora クラスターごとに Aurora DB クラスターパラメータグループを個別に設定できます。つまり、セカンダリ DB クラスターを昇格してプライマリロールを引き継ぐ場合、セカンダリからのパラメータグループは、プライマリとは異なる設定になることがあります。その場合は、プロモートされたセカンダリ DB クラスターのパラメータグループを、プライマリクラスターの設定に適合するように変更します。この方法については、「Aurora Global Database のパラメータの修正」を参照してください。
- モニタリングツールとオプション (Amazon CloudWatch Events やアラームなど) - グローバルデータベースに必要なログ機能、アラームなどを使用して、昇格された DB クラスターの設定を行います。パラメータグループと同様に、スイッチオーバープロセス中にこれらの機能の設定がプライマリから継承されることはありません。レプリケーションラグなどの一部の CloudWatch メトリクスは、セカンダリリージョンのみで使用できます。そのため、スイッチオーバーによってメトリクスの表示方法やアラームの設定方法が変わり、定義済みのダッシュボードを変更する必要が生じる場合があります。Aurora DB クラスターとモニタリングの詳細については、Overview of monitoring Amazon Aurora を参照してください。
- 他の AWS サービスとの統合を設定する - Aurora Global Database を AWS サービス (AWS Secrets Manager、AWS Identity and Access Management、Amazon S3、AWS Lambda など) と統合する場合は、必要に応じてこれらのサービスとの統合を必ず設定してください。IAM、Amazon S3、Lambda との Aurora Global Database の統合の詳細については、「Amazon Aurora Global Database を他の AWS サービスと併用する」を参照してくださ

い。Secrets Manager の詳細については、「AWS リージョン 間で AWS Secrets Manager のシークレットのレプリケーションを自動化する方法」を参照してください。

> **Note**
>
> 通常、ロールスイッチオーバーには数分かかることがあります。ただし、データベースのサイズとリージョン間の物理的な距離によって、追加のセカンダリクラスターの構築に数分から数時間かかる場合もあります。

スイッチオーバープロセスが完了すると、昇格された Aurora DB クラスターは Aurora Global Database の書き込み操作を処理できます。アプリケーションのエンドポイントを変更して、新しいエンドポイントを使用してください。Aurora Global Database の作成時に指定された名前を受け入れた場合は、アプリケーションの昇格されたクラスターのエンドポイント文字列から `-ro` を削除することで、エンドポイントを変更できます。

例えば、セカンダリクラスターのエンドポイント `my-global.cluster-**ro**-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` は、そのクラスターがプライマリに昇格したときに `my-global.cluster-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` になります。

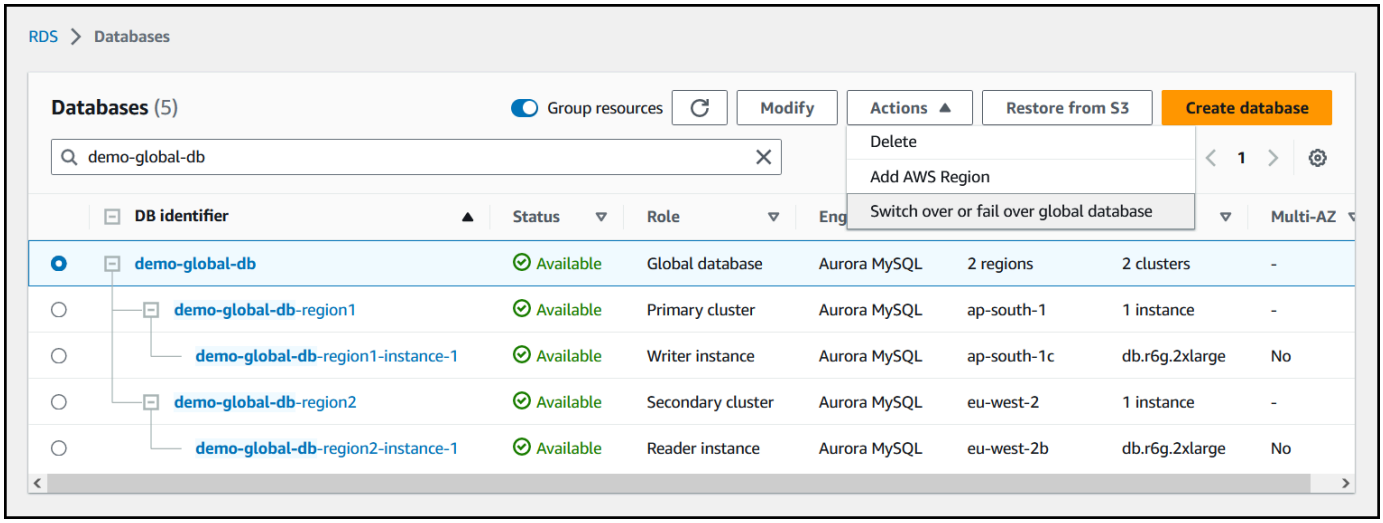
RDS Proxy を使用する場合、アプリケーションの書き込みオペレーションを、新しいプライマリクラスターに関連付けられているプロキシの適切な読み取り/書き込みエンドポイントにリダイレクトします。このプロキシエンドポイントは、デフォルトのエンドポイントでも、カスタムの読み取り/書き込みエンドポイントでもかまいません。詳細については、「RDS Proxy エンドポイントとグローバルデータベースの連携について」を参照してください。

Aurora Global Database をスイッチオーバーするには、AWS Management Console、AWS CLI、または RDS API を使用します。

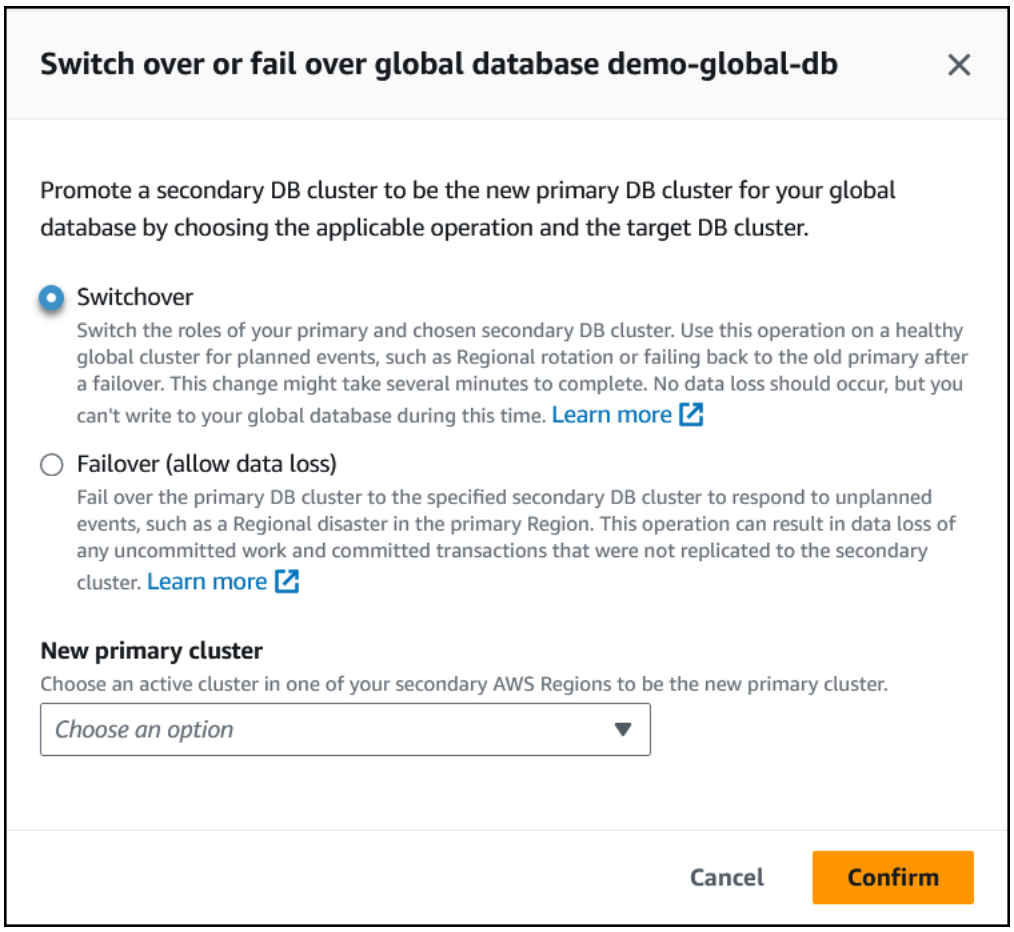
コンソール

Aurora グローバルデータベースでスイッチオーバーを実行するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. [データベース] を選択し、フェイルオーバーしたい Aurora Global Database を見つけます。
3. [アクション] メニューから [グローバルデータベースのスイッチオーバーまたはフェイルオーバー] を選択します。



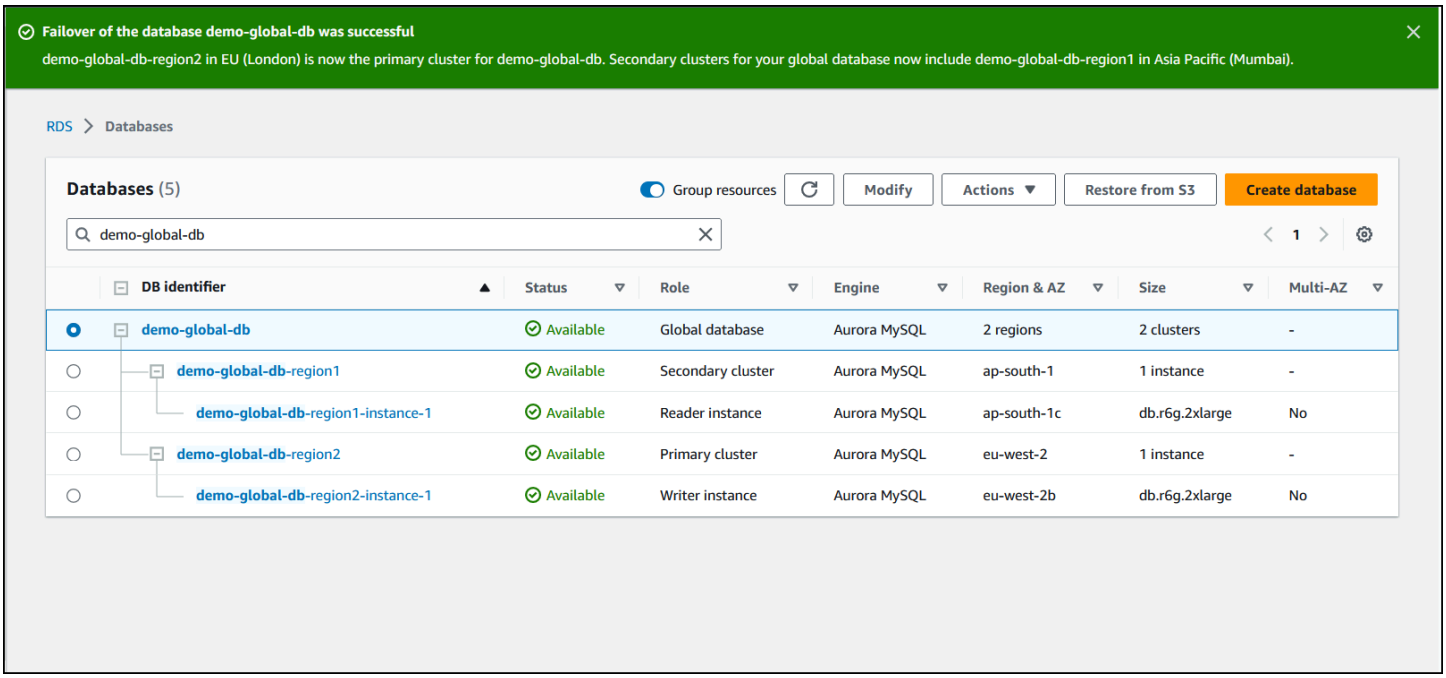
4. [スイッチオーバー] を選択します。



5. 新しいプライマリクラスタ の場合、セカンダリ AWS リージョン の 1 つで新しいプライマリクラスターになるアクティブなクラスターを選択してください。

6. [確認] を選択します。

スイッチオーバーが完了すると、次の画像に示すように、[データベース] リストから Aurora DB クラスターとその現在のロールを確認できます。



AWS CLI

Aurora グローバルデータベースでスイッチオーバーを実行するには

`switchover-global-cluster` CLI コマンドを使用して、Aurora Global Database をスイッチオーバーします。コマンドを使用して、次のパラメータ値を渡します。

- `--region` – Aurora Global Database のプライマリ DB クラスターが実行されている AWS リージョン を指定します。
- `--global-cluster-identifier` - Aurora Global Database の名前を指定します。
- `--target-db-cluster-identifier` - Aurora Global Database のプライマリに昇格する Aurora DB クラスターの Amazon リソースネーム (ARN) を指定します。

Linux、macOS、Unix の場合:

```
aws rds --region region_of_primary \
   switchover-global-cluster --global-cluster-identifier global_database_id \
  --target-db-cluster-identifier arn_of_secondary_to_promote
```

Windows の場合:

```
aws rds --region region_of_primary ^
   switchover-global-cluster --global-cluster-identifier global_database_id ^
  --target-db-cluster-identifier arn_of_secondary_to_promote
```

RDS API

Aurora グローバルデータベースをスイッチオーバーするには、SwitchoverGlobalCluster API オペレーションを実行します。

Aurora PostgreSQL- ベースのグローバルデータベースの RPO (目標復旧時点) 管理

Aurora PostgreSQL - ベースのグローバルデータベースでは、`rds.global_db_rpo` パラメータを使用して、目標復旧ポイント (RPO) を管理できます。RPO (目標復旧時点) は、停止時に失われる可能性があるデータの最大量を表します。

Aurora PostgreSQL - ベースのグローバルデータベースに RPO を設定する場合、Aurora はすべてのセカンダリクラスターの RPO ラグタイムをモニタリングし、少なくとも 1 つのセカンダリクラスターがターゲット RPO ウィンドウ内に留まることを確認します。RPO ラグタイムは、もう 1 つの時間ベースのメトリクスです。

RPO は、データベースが、フェイルオーバー後に新しい AWS リージョン でオペレーションを再開するときに使用されます。Aurora は、RPO と RPO ラグタイムを評価して、以下に示すように、プライマリでのトランザクションをコミット (またはブロック) します。

- 少なくとも 1 つのセカンダリ DB クラスターの RPO ラグタイムが RPO よりも短い場合に、トランザクションをコミットします。
- すべてのセカンダリ DB クラスターの RPO ラグタイムが RPO よりも大きい場合、トランザクションをブロックします。また、イベントを PostgreSQL ログファイルに記録し、ブロックされたセッションを示す「待機」イベントも出力します。

つまり、すべてのセカンダリクラスターがターゲット RPO より遅れている場合、Aurora は、セカンダリクラスターの少なくとも 1 つが追いつくまでプライマリクラスターのトランザクションを一時停止します。1 つ以上のセカンダリ DB クラスターのラグタイムが RPO を下回るとすぐに、一時停止中のトランザクションが再開され、コミットされます。その結果、RPO (目標復旧時点) に達するまで、トランザクションはコミットできません。

rds.global_db_rpo パラメータは動的です。遅延が十分に減少するまですべての書き込みトランザクションを停止させたくない場合は、すぐにリセットできます。この場合、Aurora は少し遅れて変更を認識して実装します。

> **Important**
>
> グローバルデータベースが 2 つのリージョンにしかない場合、rds.global_db_rpo パラメータのデフォルト値をセカンダリージョンのパラメータグループに保持しておくことをお勧めします。そうしないと、プライマリリージョンが失われてこのリージョンにフェイルオーバーしたときに、Aurora はトランザクションを一時停止する可能性があります。代わりに、障害が発生した古いリージョンで Aurora がクラスターの再構築を完了するまで待ってから、このパラメータを変更して最大 RPO を適用してください。

このパラメータを以下に示すように設定すると、生成されたメトリクスをモニタリングすることもできます。これを行うには、psql または別のツールを使用して、Aurora Global Database のプライマリ DB クラスターをクエリし、Aurora PostgreSQL - ベースのグローバルデータベースの操作に関する詳細情報を取得できます。この方法については、「Aurora PostgreSQL ベースのグローバルデータベースのモニタリング」を参照してください。

トピック

- 目標復旧時点の設定
- 目標復旧時点の表示
- 目標復旧時点の無効化

目標復旧時点の設定

rds.global_db_rpo パラメータは、PostgreSQL データベースの RPO 設定を制御します。このパラメータは、Aurora PostgreSQL でサポートされています。有効な値の rds.global_db_rpo 範囲は 20 秒から 2,147,483,647 秒 (68 年) です。ビジネスニーズとユースケースに合わせて、現実的な価値をお選びください。例えば、RPO (目標復旧時点) に最大 10 分かかる場合があります。この場合、値を 600 に設定します。

この値は、AWS Management Console、AWS CLI、または RDS API を使用して、Aurora PostgreSQL ベースのグローバルデータベースに設定します。

コンソール

RPO を設定するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. Aurora Global Database のプライマリクラスターを選択し、[Configuration (設定)] タブを開いて DB クラスターのパラメータグループを見つけます。例えば、Aurora PostgreSQL 11.7 を実行するプライマリ DB クラスターのデフォルトパラメータグループは `default.aurora-postgresql11` です。

   パラメータグループは直接編集できません。代わりに、以下を実行できます。

   - 適切なデフォルトパラメータグループをスタート点として使用して、カスタム DB クラスターのパラメータグループを作成します。例えば、`default.aurora-postgresql11` に基づいてカスタム DB クラスターのパラメータグループを作成します。
   - カスタム DB パラメータグループで、ユースケースに合わせて rds.global_db_rpo パラメータの値を設定します。有効な値の範囲は、20 秒から最大整数値 2,147,483,647 (68 年) までです。
   - 変更した DB クラスターパラメータグループを Aurora DB クラスターに適用します。

詳細については、「DB クラスターパラメータグループのパラメータの変更」を参照してください。

AWS CLI

`rds.global_db_rpo` パラメータを設定するには、modify-db-cluster-parameter-group CLI コマンドを使用します。コマンドで、プライマリクラスターのパラメータグループの名前と RPO パラメータの値を指定します。

次の例では、`my_custom_global_parameter_group` という名前のプライマリ DB クラスターのパラメータグループの RPO を 600 秒 (10分) に設定します。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name my_custom_global_parameter_group \
    --parameters
 "ParameterName=rds.global_db_rpo,ParameterValue=600,ApplyMethod=immediate"
```

Windows の場合:

```
aws rds modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name my_custom_global_parameter_group ^
    --parameters
 "ParameterName=rds.global_db_rpo,ParameterValue=600,ApplyMethod=immediate"
```

RDS API

rds.global_db_rpo パラメータを変更するには、Amazon RDS ModifyDBClusterParameterGroup API オペレーションを使用します。

目標復旧時点の表示

グローバルデータベースの目標復旧時点 (RPO) は、各 DB クラスターの rds.global_db_rpo パラメータに保存されます。表示するセカンダリクラスターのエンドポイントに接続し、この値についてインスタンスのクエリに psql を使用できます。

```
db-name=>show rds.global_db_rpo;
```

このパラメータが設定されていない場合、クエリは次の値を返します。

```
rds.global_db_rpo
-------------------
 -1
(1 row)
```

この次の応答は、1 分の RPO 設定を持つセカンダリ DB クラスターからのものです。

```
rds.global_db_rpo
-------------------
 60
(1 row)
```

CLI を使用して、クラスターのすべての rds.global_db_rpo パラメータ値を取得することで、Aurora DB クラスターのいずれかで user がアクティブかどうかを調べるための値を取得することもできます。

Linux、macOS、Unix の場合:

```
aws rds describe-db-cluster-parameters \
 --db-cluster-parameter-group-name lab-test-apg-global \
 --source user
```

Windows の場合:

```
aws rds describe-db-cluster-parameters ^
 --db-cluster-parameter-group-name lab-test-apg-global *
 --source user
```

このコマンドは、user または default-engine DB クラスターパラメータではないすべての system パラメータについて、次のような出力を返します。

```
{
    "Parameters": [
        {
            "ParameterName": "rds.global_db_rpo",
            "ParameterValue": "60",
            "Description": "(s) Recovery point objective threshold, in seconds, that
 blocks user commits when it is violated.",
            "Source": "user",
            "ApplyType": "dynamic",
            "DataType": "integer",
            "AllowedValues": "20-2147483647",
            "IsModifiable": true,
            "ApplyMethod": "immediate",
            "SupportedEngineModes": [
                "provisioned"
            ]
        }
    ]
}
```

クラスターパラメータグループのパラメータ表示の詳細については、DB クラスターパラメータグループのパラメータ値を表示する を参照してください。

目標復旧時点の無効化

RPO を無効にするには、rds.global_db_rpo パラメータをリセットします。AWS Management Console、AWS CLI、または RDS API を使用してパラメータをリセットできます。

コンソール

RPO を無効にするには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[パラメータグループ] を選択します。
3. リストで、プライマリ DB クラスターパラメータグループを選択します。
4. [Edit parameters] を選択します。
5. [rds.global_db_rpo] パラメータの横にあるボックスを選択します。
6. [リセット] を選択します。
7. 画面に [Reset parameters in DB parameter group (DB パラメータグループのパラメータのリセット)] が表示されたら、[Reset parameters (パラメータのリセット)] を選択します。

コンソールでパラメータをリセットする方法の詳細については、「DB クラスターパラメータグループのパラメータの変更」を参照してください。

AWS CLI

`rds.global_db_rpo` パラメータをリセットするには、reset-db-cluster-parameter-group コマンドを使用します。

Linux、macOS、Unix の場合:

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name global_db_cluster_parameter_group \
    --parameters "ParameterName=rds.global_db_rpo,ApplyMethod=immediate"
```

Windows の場合:

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name global_db_cluster_parameter_group ^
    --parameters "ParameterName=rds.global_db_rpo,ApplyMethod=immediate"
```

RDS API

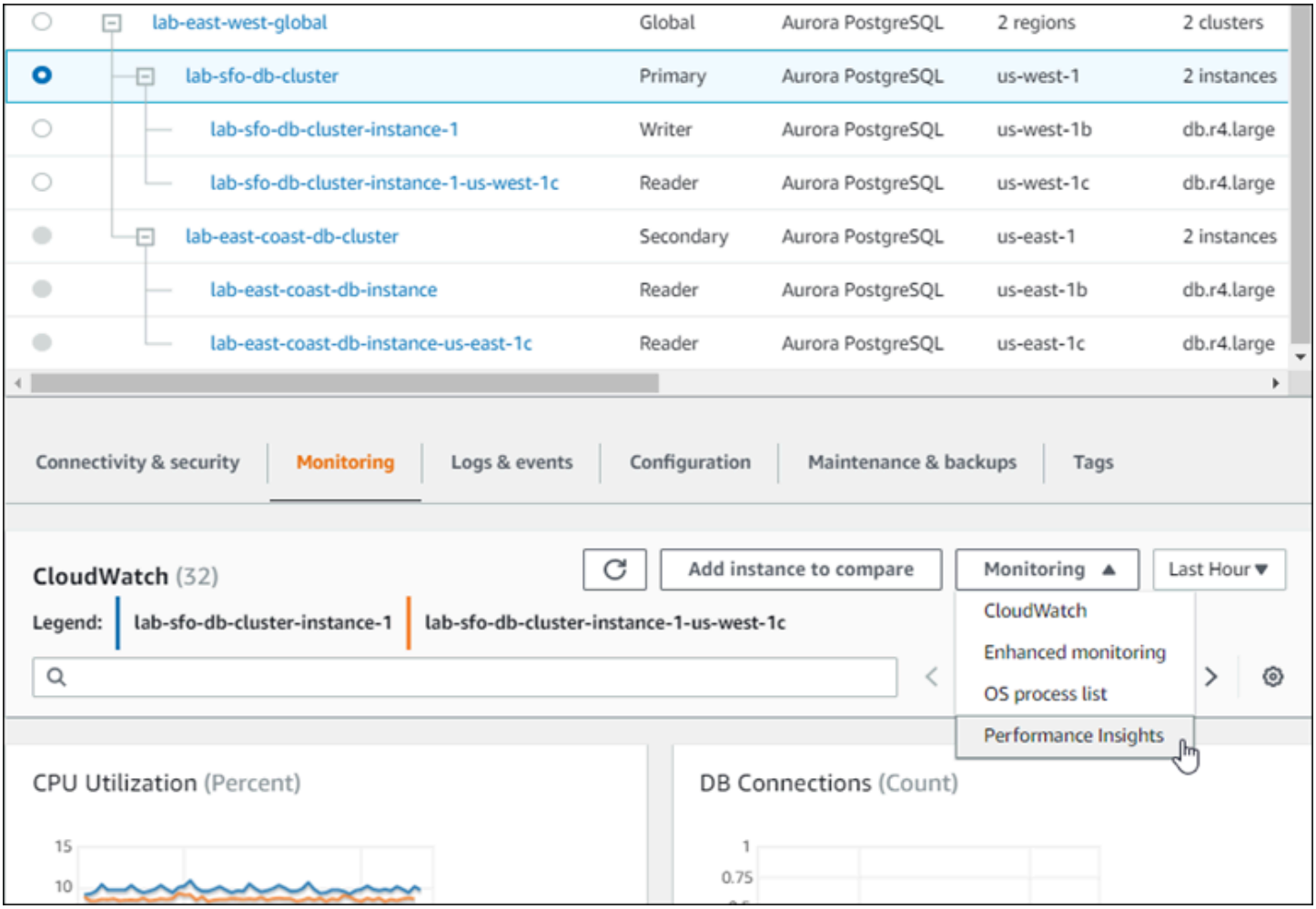
`rds.global_db_rpo` パラメータをリセットするには、Amazon RDS API ResetDBClusterParameterGroup オペレーションを使用します。

Amazon Aurora Global Database のモニタリング

Aurora Global Database を構成している Aurora DB クラスターを作成するときは、DB クラスターのパフォーマンスをモニタリングできる多くのオプションを選択できます。オプションは以下のとおりです。

- Amazon RDS Performance Insights -基盤となる Aurora データベースエンジンで Performance Schema を有効にします。Performance Insights と Aurora Global Database の詳細については、「Amazon RDS Performance Insights を使用した Amazon Aurora Global Database のモニタリング」を参照してください。
- 拡張モニタリング-CPU のプロセスまたはスレッド使用率のメトリクスを生成します。強化モニタリングの詳細については、「拡張モニタリングを使用した OS メトリクスのモニタリング」を参照してください。
- Amazon CloudWatch Logs- 指定されたログタイプを CloudWatch Logs にパブリッシュします。エラーログはデフォルトでパブリッシュされますが、Aurora データベースエンジンに固有の、その他のログを選択することが可能です。
  - Aurora MySQL- ベースの Aurora DB クラスターの場合、監査ログ、全般ログ、スロークエリログをエクスポートすることができます。
  - Aurora PostgreSQL- ベースの Aurora DB クラスターの場合、PostgreSQL ログをエクスポートすることができます。
- Aurora MySQL ベースのグローバルデータベースの場合、特定の `information_schema` テーブルを使用して、Aurora Global Database とそのインスタンスのステータスを確認することができます。この方法の詳細は、「Aurora MySQL ベースのグローバルデータベースのモニタリング」を参照してください。
- Aurora PostgreSQL ベースのグローバルデータベースの場合、特定の関数を使用して、Aurora Global Database とそのインスタンスのステータスを確認することができます。この方法の詳細は、「Aurora PostgreSQL ベースのグローバルデータベースのモニタリング」を参照してください。

次のスクリーンショットは、Aurora Global Database の、プライマリ Aurora DB クラスターの [モニタリング] タブで使用できるオプションの一部です。



詳細については、「Amazon Aurora クラスターでのメトリクスのモニタリング」を参照してください。

Amazon RDS Performance Insights を使用した Amazon Aurora Global Database のモニタリング

Amazon RDS Performance Insights は Aurora Global Database で使用できます。この機能は、Aurora Global Database の各 Aurora DB クラスターで個別に有効にします。有効にするには、[データベースの作成] ページの [追加設定] のセクションで、[Performance Insights を有効にする] を選択します。あるいは、起動後の実行中に、この機能を使用するように Aurora DB クラスターを変更することもできます。Aurora Global Database に含まれる各クラスターで、Performance Insights を有効または無効にすることができます。

Performance Insights が作成したレポートは、グローバルデータベースの各クラスターに適用されます。既に Performance Insights を使用している Aurora Global Database に新しいセカンダリ AWS

リージョン を追加する場合は、新たに追加したクラスターでも必ず Performance Insights を有効にします。既存のグローバルデータベースから Performance Insights 設定が継承されることはありません。

グローバルデータベースにアタッチされている DB インスタンスの Performance Insights ページを表示したまま、AWS リージョン を切り替えることができます。ただし、AWS リージョン を切り替えた直後は、パフォーマンス情報が表示されない場合があります。DB インスタンスの名前が各 AWS リージョン で同じになることがありますが、関連付けられた Performance Insights の URL は、各 DB インスタンスで異なります。AWS リージョン を切り替えたら、Performance Insights のナビゲーションペインで DB インスタンスの名前を再度選択します。

グローバルデータベースに関連付けられた DB インスタンスでは、パフォーマンスに影響を与える要因が各 AWS リージョン で異なる場合があります。例えば、DB インスタンスの容量は、各 AWS リージョン で異なることがあります。

Performance Insights の詳細については、「Amazon Aurora での Performance Insights を使用した DB 負荷のモニタリング」を参照してください 。

データベースアクティビティストリーミングを使用した Aurora グローバルデータベースのモニタリング

データベースアクティビティストリーム機能を使用して、グローバルデータベース内の DB クラスターの監査活動を監視し、アラームを設定することができます。DB クラスターでデータベースアクティビティストリーミングを個別に開始します。各クラスターは、独自の AWS リージョン 内の独自の Kinesis ストリーミングに監査データを配信します。詳細については、「データベースアクティビティストリームを使用した Amazon Aurora のモニタリング」を参照してください。

Aurora MySQL ベースのグローバルデータベースのモニタリング

Aurora MySQL ベースのグローバルデータベースのステータスを表示するには、information_schema.aurora_global_db_status および information_schema.aurora_global_db_instance_status テーブルをクエリします。

> **Note**
>
> `information_schema.aurora_global_db_status` および `information_schema.aurora_global_db_instance_status` テーブルは、Aurora MySQL バージョン 3.04.0 以降のグローバルデータベースでのみ使用できます。

Aurora MySQL ベースのグローバルデータベースをモニタリングするには

1. MySQL クライアントを使用して、グローバルデータベースのプライマリクラスターエンドポイントに接続します。接続方法の詳細については、「Amazon Aurora Global Database への接続」を参照してください。
2. mysql コマンドで information_schema.aurora_global_db_status テーブルをクエリして、プライマリボリュームとセカンダリボリュームを一覧表示します。このクエリは、次の例のように、グローバルデータベースのセカンダリ DB クラスターのラグタイムを返します。

```
mysql> select * from information_schema.aurora_global_db_status;
```

```
AWS_REGION | HIGHEST_LSN_WRITTEN | DURABILITY_LAG_IN_MILLISECONDS |
 RPO_LAG_IN_MILLISECONDS | LAST_LAG_CALCULATION_TIMESTAMP | OLDEST_READ_VIEW_TRX_ID
-----------+---------------------+--------------------------------
+-------------------------+---------------------------------
+------------------------
us-east-1  |           183537946 |                              0   |
      0 |  1970-01-01 00:00:00.000000     |               0
us-west-2  |           183537944 |                            428 |
      0 |  2023-02-18 01:26:41.925000     |               20806982
(2 rows)
```

出力には、次の列を含むグローバルデータベースの各 DB クラスターの行が含まれます。

- AWS_REGION – この DB クラスターがある AWS リージョン。AWS リージョン をエンジン別にリスト化した表については、「Regions and Availability Zones」(リージョンとアベイラビリティーゾーン) を参照してください。
- HIGHEST_LSN_WRITTEN - この DB クラスターに現在書き込まれているログシーケンス番号 (LSN) の最大値。

  ログシーケンス番号 (LSN) は、データベーストランザクションログ内のレコードを識別する一意の連続番号です。LSN は、より大きな LSN が後のトランザクションを表すように順序付けられます。
- DURABILITY_LAG_IN_MILLISECONDS – セカンダリ DB クラスター の HIGHEST_LSN_WRITTEN とプライマリ DB クラスターの HIGHEST_LSN_WRITTEN とのタイムスタンプ値の差。この値は、Aurora グローバルデータベースのプライマリ DB クラスターでは常に 0 です。

- RPO_LAG_IN_MILLISECONDS - 目標復旧時点 (RPO) のラグ。RPO 遅延とは、最近のユーザートランザクション COMMIT が、Aurora グローバルデータベースのプライマリ DB クラスターに保存された後、セカンダリ DB クラスターに保存されるまでにかかる時間です。この値は、Aurora グローバルデータベースのプライマリ DB クラスターでは常に 0 です。

  簡単に言うと、このメトリクスは、Aurora グローバルデータベース内の各 Aurora MySQL DB クラスターの目標復旧時点、つまり、障害が発生した場合に失われる可能性のあるデータの量を計算します。遅延と同様に、RPO は時間単位で測定されます。

- LAST_LAG_CALCULATION_TIMESTAMP - DURABILITY_LAG_IN_MILLISECONDS および RPO_LAG_IN_MILLISECONDS に対して値が最後に計算されたタイムスタンプ。1970-01-01 00:00:00+00 のような時間値は、これがプライマリ DB クラスターであることを意味します。

- OLDEST_READ_VIEW_TRX_ID - ライター DB インスタンスがパージできる最も古いトランザクションの ID。

3. information_schema.aurora_global_db_instance_status テーブルにクエリして、プライマリ DB クラスターとセカンダリ DB クラスターの両方のセカンダリ DB インスタンスをすべて一覧表示します。

```
mysql> select * from information_schema.aurora_global_db_instance_status;
```

```
SERVER_ID            |              SESSION_ID              | AWS_REGION
 | DURABLE_LSN | HIGHEST_LSN_RECEIVED | OLDEST_READ_VIEW_TRX_ID |
 OLDEST_READ_VIEW_LSN | VISIBILITY_LAG_IN_MSEC
---------------------+--------------------------------------+------------
+-------------+----------------------+-------------------------
+----------------------+------------------------
ams-gdb-primary-i2   | MASTER_SESSION_ID                    | us-east-1  |
 183537698   |                    0 |                      0 |
 0 |                      0
ams-gdb-secondary-i1 | cc43165b-bdc6-4651-abbf-4f74f08bf931 | us-west-2  |
 183537689   |           183537692 |               20806928 |
 183537682 |                     0
ams-gdb-secondary-i2 | 53303ff0-70b5-411f-bc86-28d7a53f8c19 | us-west-2  |
 183537689   |           183537692 |               20806928 |
 183537682 |                   677
ams-gdb-primary-i1   | 5af1e20f-43db-421f-9f0d-2b92774c7d02 | us-east-1  |
 183537697   |           183537698 |               20806930 |
 183537691 |                    21
```

```
(4 rows)
```

出力には、次の列を含むグローバルデータベースの各 DB インスタンスの行が含まれます。

- SERVER_ID - DB インスタンスのサーバー識別子。
- SESSION_ID - 現在のセッションの一意識別子。MASTER_SESSION_ID の値は、Writer (プライマリ) DB インスタンスを識別します。
- AWS_REGION – この DB インスタンスがある AWS リージョン。AWS リージョン をエンジン別にリスト化した表については、「Regions and Availability Zones」(リージョンとアベイラビリティーゾーン) を参照してください。
- DURABLE_LSN - ストレージで耐久性の高い LSN。
- HIGHEST_LSN_RECEIVED - ライター DB インスタンスから DB インスタンスが受信した最高の LSN。
- OLDEST_READ_VIEW_TRX_ID - ライター DB インスタンスがパージできる最も古いトランザクションの ID。
- OLDEST_READ_VIEW_LSN - ストレージから読み取るために DB インスタンスが使用する最も古い LSN。
- VISIBILITY_LAG_IN_MSEC — プライマリ DB クラスターのリーダーの場合、この DB インスタンスがライター DB インスタンスよりどれだけ遅れているか (ミリ秒単位)。セカンダリ DB クラスターのリーダーの場合、この DB インスタンスがセカンダリボリュームからどれだけ遅れているかをミリ秒単位で示します。

これらの値が時間の経過とともにどのように変化するかを確認するには、テーブルの挿入に 1 時間かかる次のトランザクションブロックを検討してください。

```
mysql> BEGIN;
mysql> INSERT INTO table1 SELECT Large_Data_That_Takes_1_Hr_To_Insert;
mysql> COMMIT;
```

場合によっては、BEGIN ステートメントの後にプライマリ DB クラスターとセカンダリ DB クラスターの間でネットワークが切断されることがあります。その場合、セカンダリ DB クラスターのDURABILITY_LAG_IN_MILLISECONDS の値は増加し始めます。INSERT ステートメントの終了時点で、DURABILITY_LAG_IN_MILLISECONDS の値は 1 時間です。ただし、プライマリ DB クラスターとセカンダリ DB クラスター間でコミットされたすべてのユーザーデータは同じであるため、 RPO_LAG_IN_MILLISECONDS の値は 0 です。COMMIT ステートメントが完了するとすぐに、RPO_LAG_IN_MILLISECONDS の値は増加します。

Aurora PostgreSQL ベースのグローバルデータベースのモニタリング

Aurora PostgreSQL ベースのグローバルデータベースのステータスを表示するには、aurora_global_db_status 関数 と aurora_global_db_instance_status 関数を使用します。

> **Note**
>
> Aurora PostgreSQL のみが、aurora_global_db_status 関数および aurora_global_db_instance_status 関数をサポートします。

Aurora PostgreSQL ベースのグローバルデータベースをモニタリングするには

1. psql などの PostgreSQL ユーティリティを使用して、グローバルデータベースのプライマリクラスターエンドポイントに接続します。接続方法の詳細については、「Amazon Aurora Global Database への接続」を参照してください。
2. psql コマンドで aurora_global_db_status 関数を使用して、プライマリボリュームとセカンダリボリュームを一覧表示します。これは、グローバルデータベースのセカンダリ DB クラスターの遅延時間を示します。

```
postgres=> select * from aurora_global_db_status();
```

```
aws_region | highest_lsn_written | durability_lag_in_msec | rpo_lag_in_msec |
 last_lag_calculation_time  | feedback_epoch | feedback_xmin
------------+---------------------+------------------------+-----------------
+----------------------------+----------------+---------------
us-east-1  |         93763984222 |                     -1 |              -1 |
 1970-01-01 00:00:00+00     |              0 |              0
us-west-2  |         93763984222 |                    900 |            1090 |
 2020-05-12 22:49:14.328+00 |              2 |    3315479243
(2 rows)
```

出力には、次の列を含むグローバルデータベースの各 DB クラスターの行が含まれます。

- aws_region – この DB クラスターがある AWS リージョン。AWS リージョン をエンジン別にリスト化した表については、「Regions and Availability Zones」(リージョンとアベイラビリティーゾーン) を参照してください。

- highest_lsn_written - この DB クラスターに現在書き込まれている口グシーケンス番号 (LSN) の最大値。

  口グシーケンス番号 (LSN) は、データベーストランザクションログ内のレコードを識別する一意の連続番号です。LSN は、より大きな LSN が後のトランザクションを表すように順序付けられます。

- durability_lag_in_msec - セカンダリ DB クラスター (highest_lsn_written) に書き込まれた最大口グシーケンス番号とプライマリ DB クラスターに書き込まれた highest_lsn_written とのタイムスタンプ差。
- rpo_lag_in_msec - 目標復旧時点 (RPO) の遅れ。この遅延は、セカンダリ DB クラスターに保存された最新のユーザートランザクションコミットと、プライマリ DB クラスターに保存された最新のユーザートランザクションコミットの間の時間差です。
- last_lag_calculation_time - durability_lag_in_msec および rpo_lag_in_msec に対して値が最後に計算されたタイムスタンプ。
- feedback_epoch - セカンダリ DB クラスターがホットスタンバイ情報を生成するときに使用するエポック。

  ホットスタンバイとは、サーバーが復旧モードまたはスタンバイモードのときに DB クラスターが接続してクエリを実行できる場合です。ホットスタンバイのフィードバックは、ホットスタンバイの場合の DB クラスターに関する情報です。詳細については、PostgreSQL ドキュメントの「ホットスタンバイ」を参照してください。

- feedback_xmin - セカンダリ DB クラスターで使用される最小 (最も古い) のアクティブトランザクション ID。

3. aurora_global_db_instance_status 関数を使用して、プライマリ DB クラスターとセカンダリ DB クラスターの両方のセカンダリ DB インスタンスをすべて一覧表示します。

```
postgres=> select * from aurora_global_db_instance_status();
```

```
server_id                                    |              session_id
  | aws_region | durable_lsn | highest_lsn_rcvd | feedback_epoch | feedback_xmin |
 oldest_read_view_lsn | visibility_lag_in_msec
----------------------------------------------+-------------------------------------
+------------+-------------+------------------+----------------+--------------
+----------------------+------------------------
apg-global-db-rpo-mammothrw-elephantro-1-n1 | MASTER_SESSION_ID
 | us-east-1  | 93763985102 |                  |                |              |
                    |
```

```
apg-global-db-rpo-mammothrw-elephantro-1-n2 | f38430cf-6576-479a-b296-dc06b1b1964a
 | us-east-1  | 93763985099 |      93763985102 |              2 |    3315479243 |
       93763985095 |                      10
apg-global-db-rpo-elephantro-mammothrw-n1   | 0d9f1d98-04ad-4aa4-8fdd-e08674cbbbfe
 | us-west-2  | 93763985095 |      93763985099 |              2 |    3315479243 |
       93763985089 |                    1017
(3 rows)
```

出力には、次の列を含むグローバルデータベースの各 DB インスタンスの行が含まれます。

- server_id - DB インスタンスのサーバー識別子。
- session_id - 現在のセッションの一意の識別子。
- aws_region – この DB インスタンスがある AWS リージョン。AWS リージョン をエンジン別にリスト化した表については、「Regions and Availability Zones」(リージョンとアベイラビリティーゾーン) を参照してください。
- durable_lsn - ストレージで耐久性の高い LSN。
- highest_lsn_rcvd - ライター DB インスタンスから DB インスタンスが受信した最高の LSN。
- feedback_epoch - DB インスタンスがホットスタンバイ情報を生成するときに使用するエポック。

  ホットスタンバイは、サーバーが復旧モードまたはスタンバイモードのときに DB インスタンスが接続してクエリを実行できる場合です。ホットスタンバイのフィードバックは、ホットスタンバイの場合の DB インスタンスに関する情報です。詳細については、ホットスタンバイに関する PostgreSQL ドキュメントを参照してください。
- feedback_xmin - DB インスタンスで使用される最小 (最も古い) のアクティブなトランザクション ID。
- oldest_read_view_lsn - ストレージから読み取るために DB インスタンスが使用する最も古い LSN。
- visibility_lag_in_msec - この DB インスタンスが書き込み DB インスタンスからどのくらい遅れているか。

これらの値が時間の経過とともにどのように変化するかを確認するには、テーブルの挿入に 1 時間かかる次のトランザクションブロックを検討してください。

```
psql> BEGIN;
psql> INSERT INTO table1 SELECT Large_Data_That_Takes_1_Hr_To_Insert;
```

```
psql> COMMIT;
```

場合によっては、BEGIN ステートメントの後にプライマリ DB クラスターとセカンダリ DB クラスターの間でネットワークが切断されることがあります。その場合は、セカンダリ DB クラスターの durability_lag_in_msec 値が増加し始めます。INSERT ステートメントの終了時は、durability_lag_in_msec 値は 1 時間です。ただし、プライマリ DB クラスターとセカンダリ DB クラスター間でコミットされたすべてのユーザーデータは同じであるため、rpo_lag_in_msec 値は 0 です。COMMIT ステートメントが完了すると、すぐに rpo_lag_in_msec 値が増加します。

Amazon Aurora Global Database を他の AWS サービスと併用する

Aurora Global Database は、Amazon S3 や AWS Lambda など他の AWS サービスと併用できます。これを行うためには、グローバルデータベースにあるすべての Aurora DB クラスターが、それぞれの AWS リージョン で同じ権限や外部関数などを持っている必要があります。Aurora Global Database の読み取り専用の Aurora セカンダリ DB クラスターは、プライマリのロールに昇格可能であるため、Aurora Global Database で使用予定のすべてのサービス用に、すべての Aurora DB クラスターに対して、事前に書き込み権限をセットアップしておくことを推奨します。

次の手順は、各 AWS のサービス で行うアクションをまとめたものです。

Aurora Global Database から AWS Lambda 関数を呼び出すには

1. Aurora Global Database を構成するすべての Aurora クラスターで、「Amazon Aurora MySQL DB クラスターからの Lambda 関数の呼び出し」の手順を実行します。
2. Aurora Global Database の各クラスターで、新しい IAM (IAM) ロールの (ARN) を設定します。
3. Aurora Global Database のデータベースユーザーに Lambda 関数を呼び出すことを許可するには、「Amazon Aurora が AWS のサービスにアクセスすることを許可する IAM ロールの作成」で作成したロールを Aurora Global Database の各クラスターに関連付けます。
4. Lambda へのアウトバウンド接続を許可するように Aurora Global Database の各クラスターを設定します。手順については、「Amazon Aurora MySQL から他の AWS のサービスへのネットワーク通信の有効化」を参照してください。

Amazon S3 からデータをロードするには

1. Aurora Global Database を構成するすべての Aurora クラスターで、「Amazon S3 バケットのテキストファイルから Amazon Aurora MySQL DB クラスターへのデータのロード」の手順を実行します。
2. グローバルデータベースの Aurora クラスターごとに、DB クラスターの `aurora_load_from_s3_role` パラメータまたは `aws_default_s3_role` パラメータを新しい IAM ロールの Amazon リソースネーム (ARN) に設定します。IAM ロールが `aurora_load_from_s3_role` に指定されていない場合、Aurora は `aws_default_s3_role` に指定されている IAM ロールを使用します。
3. Aurora Global Database のデータベースユーザーに S3 を呼び出すことを許可するには、「Amazon Aurora が AWS のサービスにアクセスすることを許可する IAM ロールの作成」で作成したロールを、グローバルデータベースの各 Aurora クラスターに関連付けます。
4. S3 へのアウトバウンド接続を許可するように Aurora Global Database の各クラスターを設定します。手順については、「Amazon Aurora MySQL から他の AWS のサービスへのネットワーク通信の有効化」を参照してください。

クエリしたデータを Amazon S3 に保存するには

1. Aurora Global Database を構成するすべての Aurora クラスターで、「Amazon Aurora MySQL DB クラスターから Amazon S3 バケット内のテキストファイルへのデータの保存」の手順を実行します。
2. グローバルデータベースの Aurora クラスターごとに、DB クラスターの `aurora_select_into_s3_role` パラメータまたは `aws_default_s3_role` パラメータを新しい IAM ロールの Amazon リソースネーム (ARN) に設定します。IAM ロールが `aurora_select_into_s3_role` に指定されていない場合、Aurora は `aws_default_s3_role` に指定されている IAM ロールを使用します。
3. Aurora Global Database のデータベースユーザーに S3 を呼び出すことを許可するには、「Amazon Aurora が AWS のサービスにアクセスすることを許可する IAM ロールの作成」で作成したロールを、グローバルデータベースの各 Aurora クラスターに関連付けます。
4. S3 へのアウトバウンド接続を許可するように Aurora Global Database の各クラスターを設定します。手順については、「Amazon Aurora MySQL から他の AWS のサービスへのネットワーク通信の有効化」を参照してください。

Amazon Aurora Global Database のアップグレード

Aurora Global Database のアップグレードは、Aurora DB クラスターのアップグレードと同じ手順に従います。ただし、プロセスを開始する前に注意すべき重要な相違点は次のとおりです。

プライマリとセカンダリの DB クラスターを同じバージョンにアップグレードすることをお勧めします。プライマリ DB クラスターとセカンダリ DB クラスターが同じメジャー、マイナー、パッチレベルのエンジンバージョンである場合にのみ、Aurora グローバルデータベースでマネージドクロスリージョンデータベースフェイルオーバーを実行できます。ただし、パッチレベルはマイナーエンジンバージョンによって異なる場合があります。詳細については、「マネージドクロスリージョンスイッチオーバーまたはフェイルオーバーに対するパッチレベルの互換性」を参照してください。

メジャーバージョンのアップグレード

Amazon Aurora Global Database のメジャーバージョンアップグレードを実行する場合は、それに含まれる個々のクラスターではなく、グローバルデータベースクラスターをアップグレードします。

Aurora PostgreSQL グローバルデータベースを上位のメジャーバージョンにアップグレードする方法については、「グローバルデータベースのメジャーアップグレード」を参照してください。

> **Note**
>
> Aurora PostgreSQL に基づく Aurora グローバルデータベースでは、目標復旧時点 (RPO) 機能がオンになっている場合、Aurora DB エンジンのメジャーバージョンアップグレードを実行できません。RPO 機能については、「Aurora PostgreSQL- ベースのグローバルデータベースの RPO (目標復旧時点) 管理」を参照してください。

Aurora MySQL グローバルデータベースを上位のメジャーバージョンにアップグレードする方法については、「グローバルデータベースのインプレースメジャーアップグレード」を参照してください。

> **Note**
>
> Aurora MySQL に基づく Aurora グローバルデータベースでは、`lower_case_table_names` パラメータがオンの場合、Aurora MySQL バージョン 2 からバージョン 3 へのインプレースアップグレードを実行できません。`lower_case_table_names` を使用しているときに、Aurora MySQL バージョン 3 へのメジャーバージョンのアップグレードを実行するには、次のプロセスを使用します。

1. グローバルクラスターからすべてのセカンダリリージョンを削除します。「Amazon Aurora Global Database からのクラスターの削除」の手順を実行します。
2. プライマリリージョンのエンジンバージョンを Aurora MySQL バージョン 3 にアップグレードします。「インプレースアップグレードの実行手順」の手順を実行します。
3. グローバルクラスターにセカンダリリージョンを追加します。「AWS リージョン の Amazon Aurora Global Database への追加」の手順を実行します。

代わりに、スナップショット復元方法を使用することもできます。詳細については、「DB クラスターのスナップショットからの復元」を参照してください。

マイナーバージョンのアップグレード

Aurora Global Database をマイナーアップグレードするときは、プライマリクラスターをアップグレードする前に、すべてのセカンダリクラスターをアップグレードします。

Aurora PostgreSQL グローバルデータベースを上位のマイナーバージョンにアップグレードする方法については、「マイナーバージョンのアップグレードとパッチの適用方法」を参照してください。Aurora MySQL グローバルデータベースを上位のマイナーバージョンにアップグレードする方法については、「エンジンのバージョンを変更して Aurora MySQL アップグレードする」を参照してください。

アップグレードを実行する前に、次の考慮事項を確認してください。

- セカンダリクラスターのマイナーバージョンをアップグレードしても、プライマリクラスターの可用性や使用にはまったく影響しません。
- マイナーアップグレードを実行するには、セカンダリクラスターは少なくとも 1 つの DB インスタンスを持っている必要があります。
- Aurora MySQL グローバルデータベースをバージョン 2.11.* にアップグレードする場合、プライマリ DB クラスターとセカンダリ DB クラスターを、パッチレベルを含めてまったく同じバージョンにアップグレードする必要があります。
- マネージドクロスリージョンスイッチオーバーまたはフェイルオーバーをサポートするには、プライマリ DB クラスターとセカンダリ DB クラスターを、エンジンのバージョンに応じて、パッチレベルを含めてまったく同じバージョンにアップグレードする必要があります。詳細については、「マネージドクロスリージョンスイッチオーバーまたはフェイルオーバーに対するパッチレベルの互換性」を参照してください。

マネージドクロスリージョンスイッチオーバーまたはフェイルオーバーに対するパッチレベルの互換性

Aurora グローバルデータベースを次のマイナーエンジンバージョンのいずれかにアップグレードすると、プライマリおよびセカンダリ DB クラスターのパッチレベルが一致しない場合でも、マネージドクロスリージョンスイッチオーバーまたはフェイルオーバーを実行できます。このリストにあるものよりもマイナーエンジンバージョンが低い場合、マネージドクロスリージョンスイッチオーバーまたはフェイルオーバーを実行するには、プライマリ DB クラスターとセカンダリ DB クラスターを同じメジャー、マイナー、パッチレベルにアップグレードする必要があります。次の表のバージョン情報および注意事項を必ず確認してください。

> **Note**
>
> 手動によるクロスリージョンフェイルオーバーの場合、ターゲットセカンダリ DB クラスターがプライマリ DB クラスターと同じメジャーおよびマイナーエンジンバージョンを実行している場合に限り、フェイルオーバープロセスを実行できます。この場合、パッチレベルが一致する必要はありません。

データベースエンジン	マイナーエンジンバージョン	注意
Aurora MySQL	マイナーバージョンはありません	すべてのマイナーバージョンで、プライマリ DB クラスターとセカンダリ DB クラスターのパッチレベルが一致する場合のみ、マネージドクロスリージョンスイッチオーバーまたはフェイルオーバーを実行できます。
Aurora PostgreSQL	• バージョン 14.5 以上 のマイナーバージョン • バージョン 13.8 以上 のマイナーバージョン • バージョン 12.12 以上 のマイナーバージョン	前の列に記載されているマイナーエンジンバージョンでは、あるパッチレベルのプライマリ DB クラスターから、異なるパッチレベルのセカンダリ DB クラスターへのマネージドクロスリージョンスイッチオーバー

データベースエンジン	マイナーエンジンバージョン	注意
	• バージョン 11.17 以上 のマイナーバージョン	またはフェイルオーバーを実行できます。 これらより前のマイナーバージョンで、プライマリ DB クラスターとセカンダリ DB クラスターのパッチレベルが一致する場合のみ、マネージドクロスリージョンスイッチオーバーまたはフェイルオーバーを実行できます。

Amazon RDS Proxy for Aurora の使用

Amazon RDS Proxy を使用すると、アプリケーションでデータベース接続をプールおよび共有して、アプリケーションのスケーリング能力を向上させることができます。RDS Proxy は、アプリケーション接続を維持しながらスタンバイ DB インスタンスに自動的に接続することで、データベースの障害に対するアプリケーションの耐障害性を高めます。RDS Proxy を使用することで、データベースに AWS Identity and Access Management (IAM) 認証を適用し、クレデンシャルを AWS Secrets Manager に安全に保存することもできます。

RDS Proxy を使用すると、データベーストラフィックの予測不可能なサージを処理できます。そうでない場合、このようなサージは、接続のオーバーサブスクリプションや新しい接続の急速な作成による問題の原因となることがあります。RDS Proxy は、データベース接続プールを確立し、このプール内の接続を再利用します。この方法により、毎回新しいデータベース接続を開くことによるメモリと CPU のオーバーヘッドを回避することができます。オーバーサブスクリプションからデータベースを保護するために、データベース接続の作成数を制御できます。

RDS Proxy は、接続プールからアプリケーション接続をすぐに提供できない場合に、これらの接続の処理順序を決めたり、スロットリングを行ったりします。レイテンシーは増加する場合がありますが、データベースの障害や過負荷が突然発生することはなく、アプリケーションは継続してスケーリングされます。接続リクエスト数が指定した制限を超えると、RDS Proxy はアプリケーション接続を拒否 (負荷を削除) します。同時に、使用可能な容量で RDS が対応できる負荷に対して、予測可能なパフォーマンスを維持します。

認証情報を処理するオーバーヘッドを減らし、新しい接続ごとに安全な接続を確立できます。RDS Proxy は、この作業の一部をデータベースに代わって処理できます。

RDS Proxy には、RDS Proxy でサポートされているエンジンバージョンとの完全な互換性があります。ほとんどのアプリケーションでは、コードを変更せずに RDS Proxy を有効にすることができます。サポートされているエンジンバージョンのリストについては、「Amazon RDS Proxy でサポートされているリージョンと Aurora DB エンジン」を参照してください。

トピック

- リージョンとバージョンの可用性
- RDS Proxy のクォータと制限事項
- RDS Proxy の使用場所の計画
- RDS Proxy の概念と用語

- RDS Proxy のスタート方法
- RDS Proxy の管理
- Amazon RDS Proxy エンドポイントの操作
- Amazon CloudWatch を使用した RDS Proxy メトリクスのモニタリング
- RDS Proxy イベントの使用
- RDS Proxy コマンドラインの例
- RDS Proxy のトラブルシューティング
- RDS Proxy の AWS CloudFormation での使用
- Aurora グローバルデータベースで RDS Proxy を使用する

リージョンとバージョンの可用性

特定の AWS リージョン での RDS Proxy のデータベースエンジンのバージョンサポートおよび可用性については、「Amazon RDS Proxy でサポートされているリージョンと Aurora DB エンジン」を参照してください。

RDS Proxy のクォータと制限事項

RDS Proxy には以下のクォータと制限が適用されます。

- AWS アカウント ID ごとに設定できるプロキシは最大 20 個です。アプリケーションでさらなるプロキシが必要な場合は、AWS サポート組織でチケットを開いて、追加のプロキシをリクエストできます。
- 各プロキシには、最大 200 個の Secrets Manager シークレットを関連付けることができます。したがって、各プロキシは、任意の時点で最大 200 の異なるユーザーアカウントに接続できます。
- 各プロキシにはデフォルトのエンドポイントがあります。プロキシごとに最大 20 のプロキシエンドポイントを追加することもできます。これらのエンドポイントを作成、表示、変更、および削除できます。
- Aurora クラスターでは、デフォルトのプロキシエンドポイントを使用するすべての接続が Aurora ライターインスタンスによって処理されます。読み取り負荷の高いワークロードの負荷分散を実行するには、プロキシの読み取り専用エンドポイントを作成できます。そのエンドポイントは、クラスターのリーダーエンドポイントに接続を渡します。そうすれば、プロキシ接続で Aurora 読み取りのスケーラビリティを利用できます。詳細については、「プロキシエンドポイントの概要」を参照してください。

- RDS Proxy は Aurora Serverless v2 クラスターでは使用できますが、Aurora Serverless v1 クラスターでは使用できません。
- RDS Proxy は、データベースと同じ仮想プライベートクラウド (VPC) 内に存在する必要があります。プロキシにはパブリックにアクセスできませんが、データベースにはパブリックにアクセスできます。例えば、ローカルホストでデータベースをプロトタイプ化する場合、プロキシへの接続を許可するために必要なネットワーク要件を設定しない限り、プロキシに接続することはできません。これは、ローカルホストがプロキシの VPC の外側にあるためです。

  > **Note**
  >
  > Aurora DB クラスターでは、クロス VPC アクセスを有効にできます。これを行うには、プロキシ用に追加のエンドポイントを作成し、そのエンドポイントで別の VPC、サブネット、およびセキュリティグループを指定します。詳細については、「VPC 間の Aurora データベースへのアクセス」を参照してください。

- `dedicated` に設定されたテナンシーを含む VPC では、RDS Proxy を使用できません。
- IAM 認証が有効になっている Aurora DB クラスターで RDS プロキシを使用する場合、ユーザー認証を確認してください。プロキシ経由で接続するユーザーは、サインイン認証情報で認証する必要があります。RDS Proxy の Secrets Manager および IAM サポートの詳細については、「AWS Secrets Manager でのデータベース認証情報の設定」と「AWS Identity and Access Management (IAM) ポリシーの設定」を参照してください 。
- SSL ホスト名の検証を使用するときは、カスタム DNS で RDS Proxy を使用することができません。
- 各プロキシは、1 つのターゲット DB クラスターに関連付けることができます。ただし、同じ DB クラスターに複数のプロキシを関連付けることができます。
- ステートメントのテキストサイズが 16 KB を超える場合、プロキシはセッションを現在の接続に固定します。
- 特定のリージョンには、プロキシを作成する際に考慮すべきアベイラビリティーゾーン (AZ) の制限があります。米国東部 (バージニア北部) リージョンは、`use1-az3` アベイラビリティーゾーンで RDS Proxy サポートしていません。米国西部 (北カリフォルニア) リージョンは、`usw1-az2` アベイラビリティーゾーンで RDS プロキシをサポートしていません。プロキシの作成時にサブネットを選択するときは、上記のアベイラビリティーゾーンのサブネットを選択しないようにしてください。
- 現在、RDS プロキシはグローバル条件コンテキストキーをサポートしていません。

グローバル条件コンテキストキーの詳細については、「IAM ユーザーガイド」の「AWS グローバル条件コンテキストキー」を参照してください。

以下のセクションに各 DB エンジンに関するその他の制限事項については、以下のセクションを参照してください。

- Aurora MySQL のその他の制限事項
- Aurora PostgreSQL の追加の制限事項

Aurora MySQL のその他の制限事項

Aurora MySQL データベースを使用した RDS Proxy には、以下の追加の制限事項が適用されます。

- RDS Proxy は MySQL `sha256_password`および`caching_sha2_password`認証プラグインをサポートしていません。これらのプラグインは、ユーザーアカウントパスワードに SHA-256 ハッシュを実装しています。
- 現在、すべてのプロキシはポート 3306 で MySQL をリッスンします。プロキシは引き続き、データベース設定で指定したポートを使用してデータベースに接続します。
- RDS Proxy は、EC2 インスタンスのセルフマネージド MySQL データベースでは使用できません。
- DB パラメータグループが 1 に設定された `read_only` パラメータを含む RDS for MySQL DB インスタンスでは、RDS Proxy を使用できません。
- RDS Proxy は MySQL の圧縮モードをサポートしていません。例えば、`mysql` コマンドの `--compress` オプションや `-C` オプションで使用される圧縮はサポートされていません。
- RDS プロキシが同じデータベース接続を再利用して別のクエリを実行すると、`GET DIAGNOSTIC` コマンドを処理するデータベース接続が不正確な情報を返すことがあります。これは、RDS プロキシがデータベース接続を多重化する場合に発生する可能性があります。
- `SET LOCAL` など、一部の SQL ステートメントと関数は、ピニングを起こさずに接続状態を変更できます。固定の最新の動作については、「固定を回避する」を参照してください。
- マルチステートメントクエリでの `ROW_COUNT()` 関数の使用はサポートされていません。
- RDS Proxy は、1 つの TLS レコードで複数のレスポンスメッセージを処理できないクライアントアプリケーションをサポートしていません。

> **Important**
>
> MySQL データベースに関連付けられているプロキシの場合、初期化クエリで設定パラメータ `sql_auto_is_null` を `true` または 0 以外の値に設定しないでください。その場合、アプリケーションの動作が正常でなくなる場合があります。

Aurora PostgreSQL の追加の制限事項

Aurora PostgreSQL データベースを使用した RDS Proxy には、以下の追加の制限事項が適用されます。

- RDS Proxy は PostgreSQL のセッション固定フィルターをサポートしていません。
- 現在、すべてのプロキシはポート 5432 で PostgreSQL をリッスンします。
- PostgreSQL の場合、RDS Proxy は現在、`CancelRequest` を発行してクライアントからのクエリのキャンセルをサポートしていません。これは例えば、インタラクティブな psql セッションで長時間実行されるクエリを、Ctrl+C を使用してキャンセルする場合に相当します。
- PostgreSQL 関数 lastval の結果は必ずしも正確ではありません。回避策としては、INSERT ステートメントを `RETURNING` 句と共に使用します。
- RDS Proxy は現在、ストリーミングレプリケーションモードをサポートしていません。

> **Important**
>
> PostgreSQL データベースを使用する既存のプロキシでは、`SCRAM` のみを使用するようにデータベース認証を変更すると、プロキシが最大 60 秒間使用できなくなります。この問題を回避するには、以下のいずれかの方法で対応します。
>
> - データベースが `SCRAM` と `MD5` 認証の両方を許可していることを確認します。
> - `SCRAM` 認証のみを使用するには、新しいプロキシを作成し、アプリケーショントラフィックを新しいプロキシに移行してから、以前にデータベースに関連付けられていたプロキシを削除します。

RDS Proxy の使用場所の計画

RDS Proxy を使用することで最大の利点を得られる DB インスタンス、クラスター、およびアプリケーションを判別できます。そのためには、以下の要因を考慮します。

- 「接続が多すぎます」エラーが頻繁に発生する DB クラスターは、プロキシと関連付けることをお勧めします。これは多くの場合、`ConnectionAttempts` CloudWatch メトリクスの値が高いことが特徴です。プロキシを使用すると、アプリケーションは多数のクライアント接続を開くことができます。一方、プロキシは DB クラスターへの長続きする接続の数をより少なく管理します。
- より小さい AWS インスタンスクラス (T2 や T3 など) を使用する DB クラスターの場合、プロキシを使用すると、メモリ不足状態を回避できます。また、接続を確立するための CPU オーバーヘッドを削減することもできます。メモリ不足状態は、多数の接続を処理するときに発生する場合があります。
- Amazon CloudWatch の特定のメトリクスをモニタリングして、DB クラスターが特定のタイプの制限に近づいているかどうかを判断できます。これらの制限は、接続数や接続管理関連のメモリに適用されます。また、特定の CloudWatch メトリクスをモニタリングすることで、DB クラスターが、多数の存続期間の短い接続を処理しているかどうかも判断できます。このような接続を開いたり閉じたりすると、データベースにパフォーマンスのオーバーヘッドが生じる可能性があります。モニタリングするメトリクスの詳細については、「Amazon CloudWatch を使用した RDS Proxy メトリクスのモニタリング」を参照してください。
- AWS Lambda 関数も、プロキシの使用に適しています。これらの関数で頻繁に行う短いデータベース接続は、RDS Proxy の接続プールを使用することで利点を得られます。Lambda アプリケーションコードでデータベース認証情報を管理する代わりに、Lambda 機能に設定済みの IAM 認証を利用できます。
- 通常、多数のデータベース接続を開いたり閉じたりし、また、接続プーリング機構が組み込まれていないアプリケーションは、プロキシの使用に適しています。
- 長期にわたって多数の接続を開いたままにするアプリケーションは、通常、プロキシの使用に適しています。SaaS (サービスとしてのソフトウェア) や e コマースなどの業界のアプリケーションは、接続を開いたままにしておくことで、データベースリクエストのレイテンシーを最小化できる場合があります。RDS Proxy を使用すると、アプリケーションは、DB クラスターに直接接続する場合よりも多くの接続を開いたままにできます。
- IAM 認証や Secrets Manager は設定が複雑であるという理由で、すべての DB クラスターには導入されていない場合があります。その場合は、既存の認証方法をそのままにして、認証をプロキシに委任できます。プロキシは、特定のアプリケーションのクライアント接続に対して認証ポリシーを適用できます。Lambda アプリケーションコードでデータベース認証情報を管理する代わりに、Lambda 機能に設定済みの IAM 認証を利用できます。

- RDS Proxy は、データベース障害に対してアプリケーションの回復力と透過性を高めるのに役立ちます。RDS Proxy はドメインネームシステム (DNS) キャッシュをバイパスすることで、 Aurora マルチ AZ データベースのフェイルオーバー時間を最大 66% 短縮できます。また、RDS Proxy は、アプリケーション接続を維持したまま、新しいデータベースインスタンスにトラフィックを自動的にルーティングします。これにより、アプリケーションに対して透明性の高いフェイルオーバーを行うことができます。

RDS Proxy の概念と用語

RDS Proxy を使用すると、 Amazon Aurora DB クラスターの接続管理をシンプルにできます。

RDS Proxy は、クライアントアプリケーションとデータベースとの間のネットワークトラフィックを処理します。これを行うアクティブな方法として、まず、データベースプロトコルを確認します。次に、その動作をアプリケーションからの SQL オペレーションとデータベースからの結果セットに基づいて調整します。

RDS Proxy により、データベースの接続管理から発生するメモリと CPU のオーバーヘッドが減ります。アプリケーション接続を同時に開く数が多いほど、データベースに必要なメモリと CPU リソースが少なくなります。また、アプリケーションの接続を閉じてから長いアイドル状態の後でアプリケーションを再度開くためのロジックは不要です。同様に、データベース問題の発生時に接続を再確立するために必要なアプリケーションロジックも減ります。

RDS Proxy のインフラストラクチャは可用性が高く、複数のアベイラビリティーゾーン (AZ) にデプロイできます。RDS Proxy の計算、メモリ、およびストレージは、Aurora DB クラスターから独立しています。この分離によって、データベースサーバーのオーバーヘッドが減り、そのリソースをデータベースワークロードの処理に集中させることができます。RDS Proxy コンピューティングリソースはサーバーレスであり、データベースのワークロードに基づいて自動的にスケーリングされます。

トピック

RDS Proxy の概念

RDS Proxy は、接続プールや以降のセクションで説明するその他の機能を実行するインフラストラクチャを処理します。プロキシは、RDS コンソールの [プロキシ] ページに表示されます。

各プロキシは、単一の Aurora DB クラスターへの接続を処理します。プロキシは、 Aurora のプロビジョンドクラスターで、現在のライターインスタンスを自動的に判別します。

プロキシで開いたままにしてデータベースアプリケーションで使用できるようにした接続は、接続プールを形成します。

デフォルトでは、RDS Proxy はセッション内の各トランザクションの後で接続を再利用できます。このトランザクションレベルの再利用は、多重化と呼ばれます。RDS Proxy が接続を接続プールから一時的に削除して再利用する場合、そのオペレーションは、接続の借用と呼ばれます。この再利用が安全であれば、RDS Proxy はその接続を接続プールに返します。

場合によっては、RDS Proxy は、現在のセッションの外でデータベース接続を再利用しても安全であると判断できません。このような場合、セッションが終了するまで、セッションは同じ接続で維持されます。このフォールバック動作は、固定と呼ばれます。

プロキシにはデフォルトのエンドポイントがあります。Amazon Aurora DB クラスターを使用する場合は、このエンドポイントに接続します。 クラスターに直接接続する読み取り/書き込みエンドポイントに接続する代わりに、この操作を行います。特定用途のエンドポイントは、Aurora クラスターで引き続き使用可能です。Aurora DB クラスターの場合、追加の読み取り/書き込みエンドポイントと読み取り専用エンドポイントを作成することもできます。詳細については、「プロキシエンドポイントの概要」を参照してください。

例えば、接続プールを使用しなくても、読み取り/書き込み接続に、引き続きクラスターエンドポイントを使用できます。負荷分散された読み取り専用接続に、引き続きリーダーエンドポイントを使用できます。クラスター内の特定の DB インスタンスの診断やトラブルシューティングに、引き続きインスタンスエンドポイントを使用できます。他の AWS のサービス (AWS Lambda など) を使用して RDS データベースに接続している場合、プロキシエンドポイントを使用するには、接続設定を変更します。例えば、RDS Proxy 機能を利用してプロキシエンドポイントを通じてデータベースにアクセスすることを Lambda 関数に許可するように指定します。

プロキシごとにターゲットグループがあります。このターゲットグループは、プロキシが接続できる Aurora DB クラスターを指します。Aurora クラスターの場合、ターゲットグループはデフォルトでクラスター内のすべての DB インスタンスに関連付けられます。これにより、プロキシは、クラスター内のライターインスタンスとして昇格された Aurora DB インスタンスに接続できます。プロキ

シに関連付けられた Aurora DB クラスターは、そのプロキシのターゲットと呼ばれます。コンソールでプロキシを作成すると、RDS Proxy によって自動的に、対応するターゲットグループも作成され、関連するターゲットが登録されます。

エンジンファミリーは、同じ DB プロトコルを使用する関連データベースエンジンのセットです。作成するプロキシごとにエンジンファミリーを選択します。

接続プーリング

各プロキシは、関連付けられた Aurora DB のライターインスタンスに対して接続プーリングを実行します。接続プーリングは、接続の開閉や、多数の接続を同時に開いたままにすることに伴うオーバーヘッドを削減するための最適化方法です。このオーバーヘッドには、新しい各接続を処理するために必要なメモリも含まれます。また、各接続を閉じて新しい接続を開くため、CPU のオーバーヘッドを伴います。例えば、TLS/SSL (Transport Layer Security/Secure Sockets Layer) のハンドシェイク、認証、ネゴシエーション機能などがあります。接続プーリングは、アプリケーションロジックを簡素化します。同時に開いている接続の数を最小限に抑えるためのアプリケーションコードを記述する必要はありません。

各プロキシは、接続の多重化も実行します。これは接続の再利用とも呼ばれます。多重化により、RDS Proxy は 1 つの基となるデータベース接続を使用して 1 つのトランザクションのすべてのオペレーションを実行します。RDS は、次のトランザクションに別の接続を使用できます。プロキシへの接続は同時に多数を開くことができます。プロキシから DB インスタンスやクラスターへの接続の数はより少なく保持されます。これにより、データベースサーバーでの接続に伴うメモリオーバーヘッドがさらに最小化されます。「接続が多すぎます」エラーが発生する可能性も減ります。

RDS Proxy のセキュリティ

RDS Proxy は、TLS/SSL や AWS Identity and Access Management (IAM) などの既存の RDS セキュリティメカニズムを使用します。これらのセキュリティ機能の概要については、「Amazon Aurora でのセキュリティ」を参照してください。また、 Aurora で使用される認証、承認、その他のセキュリティ領域でどのように機能するかを理解する必要があります。

RDS Proxy は、クライアントアプリケーションおよび基となるデータベースの間に別のセキュリティ層を追加します。例えば、基になる DB インスタンスが古いバージョンの TLS をサポートしている場合でも、TLS 1.3 を使用してプロキシに接続できます。プロキシへの接続には IAM ロールを使用できます。これは、ネイティブユーザーとパスワードによる認証方法を使用してプロキシからデータベースへの接続する場合でも同じです。この方法により、DB インスタンス自体の高額な移行作業を必要とせずに、データベースアプリケーションに強力な認証要件を適用できます。

RDS Proxy で使用するデータベース認証情報は、AWS Secrets Manager に保存します。プロキシからアクセスされる Aurora DB クラスターの各データベースユーザーは、Secrets Manager に対応するシークレットを保持している必要があります。RDS Proxy のユーザーに対して IAM 認証を設定することもできます。これにより、データベースでパスワードによるネイティブ認証方法が使用されている場合でも、データベースへのアクセスに IAM 認証を適用できます。アプリケーションコードにデータベース認証情報を埋め込む代わりに、これらのセキュリティ機能を使用することをお勧めします。

RDS Proxy での TLS/SSL の使用

TLS/SSL プロトコルを使用して RDS Proxy に接続できます。

> **Note**
>
> RDS Proxy は AWS Certificate Manager (ACM) の証明書を使用します。RDS Proxy を使用している場合は、Amazon RDS 証明書をダウンロードしたり、RDS Proxy 接続を使用するアプリケーションを更新したりする必要はありません。

プロキシとデータベース間のすべての接続に TLS を適用するには、AWS Management Consoleでプロキシを作成または変更するときに [Transport Layer Security が必要] 設定を指定できます。

RDS Proxy により、セッションでクライアントと RDS Proxy エンドポイント間でも TLS/SSL が必ず使用されるようにもできます。そのためには RDS Proxy で、クライアント側の要件を指定します。SSL セッション可変は、RDS Proxy を使用したデータベースへの SSL 接続には設定されません。

- Aurora MySQL の場合、`mysql` コマンドを実行するときに、`--ssl-mode` パラメータを使用してクライアント側の要件を指定します。
- および Aurora PostgreSQL の場合、`psql` コマンドを実行するときに、`conninfo` 文字列の一部として `sslmode=require` を指定します。

RDS Proxy は、TLS プロトコルバージョン 1.0、1.1、1.2、および 1.3 をサポートしています。プロキシに接続するには、基になるデータベースで使用しているものよりも高いバージョンの TLS を使用します。

デフォルトでは、クライアントプログラムは RDS Proxy との暗号化された接続を確立します。さらに制御する場合は、`--ssl-mode` オプションを使用できます。RDS Proxy は、クライアント側のすべての SSL モードをサポートします。

クライアントの SSL モードは次のとおりです。

PREFERRED

SSL は初期の選択肢ですが、必須ではありません。

DISABLED

SSL は許可されていません。

REQUIRED

SSL を強制します。

VERIFY_CA

SSL を義務化し、認証機関 (CA) を検証します。

VERIFY_IDENTITY

SSL を義務化し、CA と CA ホスト名を検証します。

`--ssl-mode`、`VERIFY_CA`、または `VERIFY_IDENTITY` でクライアントを使用する場合は、CA を指す `--ssl-ca` を `.pem` 形式で指定します。`.pem` ファイルを使用するには、Amazon Trust Services からすべてのルート CA PEM をダウンロードし、1 つの `.pem` ファイルに配置します。

RDS Proxy は、ドメインとそのサブドメインの両方に適用されるワイルドカード証明書を使用します。`mysql` クライアントを使用して SSL モード `VERIFY_IDENTITY` で接続する場合、現時点では、MySQL 8.0 互換の `mysql` コマンドを使用する必要があります。

フェイルオーバー

フェイルオーバーは、元のデータベースインスタンスが使用できなくなったときに、別のインスタンスに置き換える高可用性機能です。フェイルオーバーは、データベースインスタンスの問題が原因で発生することがあります。また、通常のメンテナンス手順の一環として、データベースのアップグレード時などに発生することもあります。フェイルオーバーは、ライターインスタンスに加えて 1 つ以上のリーダーインスタンスが含まれている Aurora DB クラスターにも適用されます。

プロキシを介して接続すると、データベースのフェイルオーバーに対してアプリケーションの耐障害性が高くなります。元の DB インスタンスが使用できなくなると、RDS Proxy はアイドル状態のアプリケーション接続を切断せずにスタンバイデータベースに接続します。これにより、フェイルオーバープロセスが高速化および簡素化されます。これは、通常の再起動やデータベース問題に伴う停止よりも、アプリケーションの停止が短くなります。

RDS Proxy を使用していない場合は、フェイルオーバーによって短時間の停止が発生します。停止中は、フェイルオーバー中のデータベースに対して書き込み操作を実行できません。既存のデータベース接続はすべて中断されるため、アプリケーションで接続を再度開く必要があります。利用できなくなった DB インスタンスの代わりに読み取り専用 DB インスタンスが昇格すると、新しい接続および書き込みオペレーションでデータベースが使用可能になります。

DB フェイルオーバー中、RDS Proxy は引き続き同じ IP アドレスで接続を受け入れ、接続を自動的に新しいプライマリ DB インスタンスに転送します。RDS Proxy を介して接続しているクライアントは、以下の影響を受けません。

- フェイルオーバー時のドメインネームシステム (DNS) 伝播の遅延。
- ローカル DNS キャッシュ。
- 接続タイムアウト。
- どの DB インスタンスが現在のライターであるかの不確実性。
- 接続を閉じずに使用不能になった以前のライターからのクエリ応答の待機。

アプリケーション独自の接続プールがある場合は、RDS Proxy を経由することで、ほとんどの接続がフェイルオーバーやその他の中断時にも存続します。トランザクションや SQL ステートメントの途中の接続のみがキャンセルされます。RDS Proxy は、すぐに新しい接続を受け入れます。データベースライターが使用できない場合、RDS Proxy は着信リクエストをキューに入れます。

アプリケーション独自の接続プールがない場合は、RDS Proxy を使用することで、接続速度が高速になり、開いたままの接続の数を増やすことができます。データベースからの頻繁な再接続に伴う高額なオーバーヘッドがオフロードされます。そのために、RDS Proxy 接続プールに保持されているデータベース接続を再利用します。このアプローチは、設定コストが大きい TLS 接続で特に重要です。

トランザクション

1 つのトランザクション内のすべてのステートメントは、常に同じ基となるデータベース接続を使用します。このトランザクションが終了すると、この接続は別のセッションで使用可能になります。トランザクションを粒度の単位として使用すると、次のような結果になります。

- Aurora MySQL `autocommit` 設定がオンのときは、各ステートメントが終わるたびに接続の再利用が発生することがあります。
- 逆に、`autocommit` 設定が無効になっていると、セッションで最初に発行したステートメントによって新しいトランザクションが開始されます。例えば、`SELECT`、`INSERT`、`UPDATE` などの

データ操作言語 (DML) ステートメントのシーケンスを入力したとします。この場合、COMMIT または ROLLBACK を発行するか、またはトランザクションが終了するまで、接続の再利用は起こりません。

- データ定義言語 (DDL) ステートメントを入力すると、そのステートメントの完了後にトランザクションが終了します。

RDS Proxy は、データベースクライアントアプリケーションで使用されているネットワークプロトコルを介してトランザクションが終了したことを検出します。トランザクションの検出は、SQL ステートメントのテキスト内にある COMMIT や ROLLBACK などのキーワードに依存しません。

場合によっては、セッションを別の接続に移動することが実用的でないようなデータベースリクエストが RDS Proxy で検出されることがあります。このような場合、セッションの残りの部分では、その接続の多重化がオフになります。セッションに対して多重化が実用的であることを RDS Proxy で確信できない場合は、同じルールが適用されます。このオペレーションは固定と呼ばれます。固定を検出して最小化する方法については、「固定を回避する」を参照してください。

RDS Proxy のスタート方法

以下のセクションでは、RDS Proxy のセットアップおよび管理方法について説明します。また、関連するセキュリティオプションの設定方法が記載されています。これらのオプションは、各プロキシにアクセスできるユーザーと、各プロキシから DB インスタンスに接続する方法を制御します。

トピック

- ネットワーク前提条件の設定
- AWS Secrets Manager でのデータベース認証情報の設定
- AWS Identity and Access Management (IAM) ポリシーの設定
- RDS Proxy の作成
- RDS Proxy の表示
- RDS Proxy を介したデータベースへの接続

ネットワーク前提条件の設定

RDS Proxy を使用するには、Aurora DB クラスターと RDS Proxy の間に、共通の仮想プライベートクラウド (VPC) が必要です。この VPC には、異なるアベイラビリティーゾーンにあるサブネットが

2 つ以上必要です。アカウントは、これらのサブネットを所有することも、他のアカウントと共有することもできます。VPC 共有の詳細については、共有 VPC の操作を参照してください。

Amazon EC2、Lambda、Amazon ECS などのクライアントアプリケーションリソースは、プロキシと同じ VPC に置くことができます。または、プロキシとは別の VPC に置くこともできます。Aurora DB クラスターに正常に接続できた場合は、既に必要なネットワークリソースが存在します。

トピック

- サブネット情報の取得
- IP アドレス容量の計画

サブネット情報の取得

Aurora の使用を開始したばかりの場合は、「Amazon Aurora の環境をセットアップする」の手順に従って、データベースに接続するための基本を学習できます。「Amazon Aurora の開始方法」のチュートリアルに従うこともできます。

プロキシを作成するには、プロキシが動作するサブネットと VPC を指定する必要があります。次の Linux の例は、AWS アカウント が所有する VPC とサブネットを調べる AWS CLI コマンドを示しています。特に、CLI を使用してプロキシを作成するときは、サブネット ID をパラメータとして渡します。

```
aws ec2 describe-vpcs
aws ec2 describe-internet-gateways
aws ec2 describe-subnets --query '*[].[VpcId,SubnetId]' --output text | sort
```

次の Linux の例は、特定の Aurora DB クラスターに対応するサブネット ID を決定する AWS CLI コマンドを示しています。

Aurora クラスターの場合は、まず、関連付けられた DB インスタンスの 1 つの ID を見つけます。その DB インスタンスで使用されているサブネット ID を抽出できます。そのためには、DB インスタンスの describe 出力で、DBSubnetGroup とSubnets 属性内のネストされたフィールドを調べます。データベースサーバーのプロキシを設定するときに、それらのサブネット ID の一部またはすべてを指定します。

```
$ # Find the ID of any DB instance in the cluster.
$  aws rds describe-db-clusters --db-cluster-identifier my_cluster_id --query '*[].
[DBClusterMembers]||[0]||[0][*].DBInstanceIdentifier' --output text
```

```
my_instance_id
instance_id_2
instance_id_3
```

DB インスタンス識別子を見つけたら、関連する VPC を調べて、そのサブネットを見つけます。次の Linux の例は、その方法を示しています。

```
$ #From the DB instance, trace through the DBSubnetGroup and Subnets to find the subnet
 IDs.
$ aws rds describe-db-instances --db-instance-identifier my_instance_id --query '*[].
[DBSubnetGroup]|[0]|[0]|[Subnets]|[0]|[*].SubnetIdentifier' --output text
```

```
subnet_id_1
subnet_id_2
subnet_id_3
...
```

```
$ #From the DB instance, find the VPC.
$ aws rds describe-db-instances --db-instance-identifier my_instance_id --query '*[].
[DBSubnetGroup]|[0]|[0].VpcId' --output text
```

```
my_vpc_id
```

```
$ aws ec2 describe-subnets --filters Name=vpc-id,Values=my_vpc_id --query '*[].
[SubnetId]' --output text
```

```
subnet_id_1
subnet_id_2
subnet_id_3
subnet_id_4
subnet_id_5
subnet_id_6
```

IP アドレス容量の計画

RDS Proxy は、登録されている DB インスタンスのサイズと数に基づき、必要に応じて自動的に容量を調整します。特定の操作には、登録されたデータベースのサイズの増加や内部 RDS Proxy メンテナンス操作など、プロキシ容量の追加が必要になる場合もあります。これらのオペレーション中、

プロキシは追加の容量をプロビジョニングするため、より多くの IP アドレスを必要とする場合があります。これらの追加アドレスによって、ワークロードに影響を与えずにプロキシを拡張できます。サブネットに空き IP アドレスがない場合、プロキシのスケールアップはできません。そのため、クエリの待ち時間が長くなったり、クライアント接続が失敗したりする可能性があります。サブネットに十分な空きの IP アドレスがないと、RDS はイベント RDS-EVENT-0243 を通じて通知します。このイベントのフィールドの詳細については、「RDS Proxy イベントの使用」を参照してください。

以下は、DB インスタンスのクラスサイズに基づいて、プロキシ用としてサブネットで未使用のままにすべき最小の IP アドレス数の推奨値です。

DB インスタンスクラス	IP アドレスの最小値
db.*.xlarge 以下	10
db.*.2xlarge	15
db.*.4xlarge	25
db.*.8xlarge	45
db.*.12xlarge	60
db.*.16xlarge	75
db.*.24xlarge	110

これらの推奨 IP アドレス数は、デフォルトのエンドポイントのみを使用するプロキシの推定値です。エンドポイントまたはリードレプリカを追加したプロキシには、さらに多くの空き IP アドレスが必要になる場合があります。エンドポイントを追加するたびに、追加で 3 つの IP アドレスを予約することをお勧めします。各リードレプリカに、そのリードレプリカのサイズに基づき、表に指定された IP アドレスを追加で予約することをお勧めします。

> **Note**
>
> RDS Proxy は、VPC 内で 215 を超える IP アドレスをサポートしていません。

例えば、Aurora DB クラスターに関連付けられるプロキシに必要な IP アドレスを見積もるとします。

この場合、次のように仮定します。

- Aurora DB クラスターには、サイズが db.r5.8xlarge のライターインスタンスが 1 つと、サイズが db.r5.2xlarge のリーダーインスタンスが 1 つあります。
- この DB クラスターに接続されているプロキシには、デフォルトのエンドポイントと、読み取り専用ロールのカスタムエンドポイントが 1 つずつあります。

この場合、プロキシには約 63 個の空き IP アドレスが必要です (ライターインスタンス用に 45 個、リーダーインスタンス用に 15 個、追加のカスタムエンドポイント用に 3 個)。

AWS Secrets Manager でのデータベース認証情報の設定

作成するプロキシごとに、まず Secrets Manager サービスを使用してユーザー名およびパスワード認証情報のセットを保存します。Aurora DB クラスターで、プロキシの接続先のデータベースユーザーアカウントごとに個別の Secrets Manager シークレットを作成します。

Secrets Manager コンソールでは、`username` および `password` フィールドの値を使用してこれらのシークレットを作成します。これにより、プロキシに関連付けた Aurora DB クラスターの対応するデータベースユーザーにプロキシから接続できます。これを行うには、[他のデータベース認証情報]、[RDS データベース認証情報]、[他の種類のシークレット] のいずれかの設定を使用します。[ユーザー名] フィールドと [パスワード] フィールドに適切な値を入力し、その他の必須フィールドに値を入力します。プロキシは、ホスト や ポート などの他のフィールド (シークレット内に存在する場合) を無視します。これらの詳細は、プロキシによって自動的に提供されます。

[その他のタイプのシークレット] を選択することもできます。この場合、`username` と `password` という名前のキーを使用してシークレットを作成します。

プロキシが使用するシークレットは特定のデータベースサーバーには関連しないため、複数のプロキシでシークレットを再利用できます。そのためには、複数のデータベースサーバーで同じ認証情報を使用します。例えば、開発サーバーとテストサーバーで同じ認証情報を使用できます。

特定のデータベースユーザーとしてプロキシ経由で接続するには、シークレットに関連付けられているパスワードが、そのユーザーのデータベースパスワードと一致していることを確認してください。不一致がある場合は、Secrets Manager で該当するシークレットを更新できます。この場合でも、シークレットの認証情報とデータベースパスワードが一致する他のアカウントには接続できます。

AWS CLI または RDS API を通じてプロキシを作成する場合、対応するシークレットの Amazon リソースネーム (ARN) を指定します。プロキシがアクセスできるすべての DB ユーザーアカウントに

対し、同じように操作します。AWS Management Console では、シークレットをそのわかりやすい名前を使用して選択します。

Secrets Manager でシークレットを作成する手順については、Secrets Manager ドキュメントのシークレットの作成ページを参照してください。次のいずれかの方法を使用します。

- コンソールで Secrets Manager を使用します。
- CLI を使用して RDS Proxy 用の Secrets Manager シークレットを作成するには、次のようなコマンドを使用します。

```
aws secretsmanager create-secret
  --name "secret_name"
  --description "secret_description"
  --region region_name
  --secret-string '{"username":"db_user","password":"db_user_password"}'
```

- Secrets Manager シークレットを暗号化するカスタムキーを作成することもできます。次のコマンドはキーの例を作成します。

```
PREFIX=my_identifier
aws kms create-key --description "$PREFIX-test-key" --policy '{
  "Id":"$PREFIX-kms-policy",
  "Version":"2012-10-17",
  "Statement":
    [
      {
        "Sid":"Enable IAM User Permissions",
        "Effect":"Allow",
        "Principal":{"AWS":"arn:aws:iam::account_id:root"},
        "Action":"kms:*","Resource":"*"
      },
      {
        "Sid":"Allow access for Key Administrators",
        "Effect":"Allow",
        "Principal":
          {
            "AWS":
              ["$USER_ARN","arn:aws:iam:account_id::role/Admin"]
          },
        "Action":
          [
            "kms:Create*",
```

```
            "kms:Describe*",
            "kms:Enable*",
            "kms:List*",
            "kms:Put*",
            "kms:Update*",
            "kms:Revoke*",
            "kms:Disable*",
            "kms:Get*",
            "kms:Delete*",
            "kms:TagResource",
            "kms:UntagResource",
            "kms:ScheduleKeyDeletion",
            "kms:CancelKeyDeletion"
          ],
        "Resource":"*"
      },
      {
        "Sid":"Allow use of the key",
        "Effect":"Allow",
        "Principal":{"AWS":"$ROLE_ARN"},
        "Action":["kms:Decrypt","kms:DescribeKey"],
        "Resource":"*"
      }
    ]
}'
```

例えば、以下のコマンドは、2 つのデータベースユーザーの Secrets Manager シークレットを作成します。

```
aws secretsmanager create-secret \
  --name secret_name_1 --description "db admin user" \
  --secret-string '{"username":"admin","password":"choose_your_own_password"}'

aws secretsmanager create-secret \
  --name secret_name_2 --description "application user" \
  --secret-string '{"username":"app-user","password":"choose_your_own_password"}'
```

カスタム AWS KMS キーで暗号化されたシークレットを作成するには、以下のコマンドを使用します。

```
aws secretsmanager create-secret \
  --name secret_name_1 --description "db admin user" \
```

```
  --secret-string '{"username":"admin","password":"choose_your_own_password"}'
  --kms-key-id arn:aws:kms:us-east-2:account_id:key/key_id

aws secretsmanager create-secret \
  --name secret_name_2 --description "application user" \
  --secret-string '{"username":"app-user","password":"choose_your_own_password"}'
  --kms-key-id arn:aws:kms:us-east-2:account_id:key/key_id
```

AWS アカウントが所有する秘密を確認するには、次のようなコマンドを使用します。

```
aws secretsmanager list-secrets
```

CLI を使用してプロキシを作成する場合、1 つ以上のシークレットの Amazon リソースネーム (ARN) を `--auth` パラメータに渡します。次の Linux の例は、AWS アカウントが所有する各シークレットの名前と ARN のみを含むレポートを準備する方法を示しています。この例では、`--output table` バージョン 2 で使用可能な AWS CLI パラメータを使用します。AWS CLI バージョン 1 を使用している場合は、代わりに `--output text` を使用します。

```
aws secretsmanager list-secrets --query '*[].[Name,ARN]' --output table
```

シークレットに正しい資格情報を正しい形式で格納したことを確認するには、次のようなコマンドを使用します。*your_secret_name* は、短縮名またはシークレットの ARN に置き換えます。

```
aws secretsmanager get-secret-value --secret-id your_secret_name
```

出力には、次のような JSON でエンコードした値を表示する行を含める必要があります。

```
"SecretString": "{\"username\":\"your_username\",\"password\":\"your_password\"}",
```

AWS Identity and Access Management (IAM) ポリシーの設定

Secrets Manager でシークレットを作成したら、これらのシークレットにアクセスできる IAM ポリシーを作成します。IAM の使用に関する一般情報については、「Amazon Aurora での Identity and Access Management」を参照してください。

> **Tip**
>
> 以下の手順は、IAM コンソールを使用する場合に適用されます。RDS に AWS Management Console を使用する場合は、RDS によって自動的に IAM ポリシーが作成されます。その場合は、以下の手順を省略できます。

プロキシで使用する Secrets Manager シークレットにアクセスするための IAM ポリシーを作成するには

1. IAM コンソールにサインインします。「IAM ロールの作成」で説明されているロールの作成プロセスに従い、[AWS のサービスにアクセス許可を委任するロールの作成] を選択します。

   [信頼されたエンティティタイプ] で、[AWS サービス] を選択します。[ユースケース] で、[他の AWS サービスのユースケース] ドロップダウンから [RDS] を選択します。[RDS – ロールをデータベースに追加する] を選択します。
2. 新しいロールの場合は、インラインポリシーを追加するステップを実行します。「IAM ポリシーの編集」と同じ一般的な手順を使用します。以下の JSON を [JSON] テキストボックスに貼り付けます。自分のアカウント ID に置き換えます。AWS リージョンを `us-east-2` に置き換えてください。作成したシークレットの Amazon リソースネーム (ARN) を置き換えます。「IAM ポリシーステートメントで KMS キーを指定する」を参照してください。`kms:Decrypt` アクションには、デフォルトの AWS KMS key の ARN または独自の KMS キーに置き換えてください。どちらを使用するかは、Secrets Manager のシークレットの暗号化に使用されたものによって決まります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": [
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_1",
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_2"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
```

```
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:us-east-2:account_id:key/key_id",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "secretsmanager.us-east-2.amazonaws.com"
                }
            }
        }
    ]
}
```

3. この IAM ロールの信頼ポリシーを編集します。以下の JSON を [JSON] テキストボックスに貼り付けます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

次のコマンドは、AWS CLI で同じ操作を実行します。

```
PREFIX=my_identifier
USER_ARN=$(aws sts get-caller-identity --query "Arn" --output text)

aws iam create-role --role-name my_role_name \
  --assume-role-policy-document '{"Version":"2012-10-17","Statement":
[{"Effect":"Allow","Principal":{"Service":
["rds.amazonaws.com"]},"Action":"sts:AssumeRole"}]}'

ROLE_ARN=arn:aws:iam::account_id:role/my_role_name

aws iam put-role-policy --role-name my_role_name \
  --policy-name $PREFIX-secret-reader-policy --policy-document '{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": [
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_1",
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_2"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:us-east-2:account_id:key/key_id",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "secretsmanager.us-east-2.amazonaws.com"
                }
            }
        }
    ]
}
```

RDS Proxy の作成

DB クラスターに対する接続を管理するには、プロキシを作成します。プロキシは、Aurora MySQL または Aurora PostgreSQL DB クラスターに関連付けることができます。

AWS Management Console

プロキシを作成するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[プロキシ] を選択します。
3. [Create proxy (プロキシの作成)] を選択します。
4. プロキシのすべての設定を選択します。

   [プロキシの設定] で、以下の情報を提供します。

- エンジンファミリー。この設定は、データベースとの間で送受信されるネットワークトラフィックを解釈するときに、プロキシが認識するデータベースネットワークプロトコルを決定します。Aurora MySQL の場合は、[MariaDB and MySQL] (MariaDB と MySQL) を選択します。Aurora PostgreSQL の場合は、[PostgreSQL] を選択します。
- プロキシ識別子。AWS アカウント ID と現在の AWS リージョン内で一意の名前を指定します。
- アイドル状態のクライアント接続のタイムアウト。プロキシがクライアント接続を閉じるまでに接続がアイドル状態を継続できる時間を選択します。デフォルトは 1,800 秒 (30 分) です。クライアント接続がアイドル状態と見なされるのは、前のリクエストの完了後、新しいリクエストが指定時間内にアプリケーションから送信されない場合です。基となるデータベース接続は開いたままで、接続プールに返されるため、新しいクライアント接続で再利用できます。

  プロキシで古い接続を事前に削除するには、クライアント接続でのアイドル状態のタイムアウトを短くします。ワークロードがスパイクしているときは、接続を確立するコストを節約するために、アイドル状態のクライアント接続のタイムアウトを長くします。

[ターゲットグループの設定] で、以下の情報を提供します。

- データベース。このプロキシを介してアクセスする Aurora DB クラスターを 1 つ選択します。このリストには、互換性のあるデータベースエンジン、エンジンバージョン、および他の設定を持つ DB インスタンスとクラスターのみが含まれます。リストが空の場合は、RDS Proxy と互換性のある新しい DB インスタンスまたはクラスターを作成します。これを行うには、「Amazon Aurora DB クラスターの作成」の手順に従います。次に、プロキシの作成をもう一度試します。
- 接続プールの最大接続数。1 ~ 100 の値を指定します。この設定は、RDS Proxy が接続に使用できる `max_connections` 値の割合 (%) を表します。この DB インスタンスまたはクラスターで 1 つのプロキシのみを使用する場合は、この値を 100 に設定できます。この設定を RDS Proxy で使用する方法の詳細については、「MaxConnectionsPercent」を参照してください。
- セッションの固定フィルター。(オプション) このオプションを使用すると、検出された特定のタイプのセッション状態に対して RDS プロキシが PIN されないようにすることができます。これにより、クライアント接続間でデータベース接続を多重化するというデフォルトの安全対策が回避されます。現在、設定は PostgreSQL についてはサポートされていません。唯一の選択肢は `EXCLUDE_VARIABLE_SETS` です。

この設定を有効にすると、ある接続のセッション変数が他の接続に影響を与える可能性があります。これにより、クエリが現在のトランザクション以外に設定されたセッション変数値に依存している場合、エラーや正確性の問題が発生する可能性があります。アプリケーションがクライアント接続間でデータベース接続を共有しても安全であることを確認した後に、このオプションの使用を検討してください。

以下のパターンは安全だと考えられます。

- 有効なセッション変数値に変更がない、つまりセッション変数に変更がない SET ステートメント。
- セッション変数の値を変更し、同じトランザクションでステートメントを実行します。

詳細については、「固定を回避する」を参照してください。

- 接続の借用タイムアウト。場合によっては、利用可能なすべてのデータベース接続をプロキシが使い切ることがあります。このような場合、プロキシがタイムアウトエラーを返す前に、データベース接続が使用可能になるまで待つ時間を指定できます。最大 5 分の期間を指定できます。この設定は、プロキシで最大数の接続が開いていて、すべての接続が既に使用されている場合にのみ適用されます。
- 初期化クエリ。(オプション) 新しい各データベース接続を開くときに実行するプロキシ用の 1 つ以上の SQL ステートメントを指定できます。設定は通常、各接続のタイムゾーンや文字セットなどの設定が同一であることを確認するために、SET ステートメントとともに使用されます。複数のステートメントの場合は、セミコロンをセパレータとして使用します。例えば、1 つの SET ステートメントに `SET x=1, y=2` など複数の可変を含めることもできます。

[Authentication] (認証) には、次の情報を入力します。

- IAM ロール。前に選択した Secrets Manager シークレットに対するアクセス許可のある IAM ロールを選択します。または、AWS Management Console から新しい IAM ロールを作成することもできます。
- Secrets Manager シークレット。プロキシが Aurora DB クラスターにアクセスできるようにするデータベースユーザー認証情報を含む Secrets Manager シークレットを少なくとも 1 つ選択します。
- クライアント承認タイプ。プロキシがクライアントからの接続に使用する認証タイプを選択します。選択した内容は、このプロキシに関連付けるすべての Secrets Manager シークレットに適用されます。シークレットごとに異なるクライアント認証タイプを指定する必要がある場合は、代わりに AWS CLI または API を使用してプロキシを作成します。

- IAM 認証。プロキシへの接続に対し、IAM 認証の要求、拒否のいずれかを選択します。選択した内容は、このプロキシに関連付けるすべての Secrets Manager シークレットに適用されます。シークレットごとに異なる IAM 認証を指定する必要がある場合は、代わりに AWS CLI または API を使用してプロキシを作成します。

[接続] で、以下の情報を提供します。

- Transport Layer Security が必要。プロキシですべてのクライアント接続に対して TLS/SSL を適用する場合は、この設定を選択します。プロキシへの暗号化された接続または暗号化されていない接続を使用すると、プロキシは基となるデータベースへの接続時に同じ暗号化設定を使用します。
- サブネット。このフィールドには、VPC に関連付けられたすべてのサブネットがあらかじめ入力されています。このプロキシに不要なサブネットは削除できます。少なくとも 2 つのサブネットを残す必要があります。

追加の接続設定を定義します。

- VPC セキュリティグループ。既存の VPC セキュリティグループを選択します。または、AWS Management Console から新しいセキュリティグループを作成することもできます。アプリケーションがプロキシにアクセスできるようにインバウンドルールを設定する必要があります。また、DB ターゲットからのトラフィックを許可するようにアウトバウンドルールを設定する必要があります。

> **Note**
>
> このセキュリティグループは、プロキシからデータベースへの接続を許可する必要があります。同じセキュリティグループが、アプリケーションからプロキシへのイングレスと、プロキシからデータベースへのエグレスに使用されます。例えば、データベースとプロキシに同じセキュリティグループを使用するとします。この場合は必ず、そのセキュリティグループ内のリソースが同じセキュリティグループ内の他のリソースと通信できるように指定してください。
> 共有 VPC を使用する場合、VPC のデフォルトのセキュリティグループや、別のアカウントに属しているセキュリティグループを使用することはできません。自分のアカウントに属しているセキュリティグループを選択します。存在しない場合は、作成します。この制限事項の詳細については、共有 VPC の操作を参照してください。

RDS は、高可用性を確保するために複数のアベイラビリティーゾーンにプロキシをデプロイします。このようなプロキシでクロス AZ 通信を有効にするには、プロキシサブネットのネットワークアクセスコントロールリスト (ACL) で、エンジンポート固有のエグレスとすべてのポートのイングレスを許可する必要があります。ネットワーク ACL の詳細については、「ネットワーク ACL を使用してサブネットへのトラフィックを制御する」を参照してください。プロキシとターゲットのネットワーク ACL が同じである場合は、ソースを VPC CIDR に設定した、TCP プロトコルイングレスルールを追加する必要があります。また、[送信先] を VPC CIDR に設定して、エンジンポート固有の TCP プロトコル Egress ルールも追加する必要があります。

(オプション) 詳細設定を定義します。

- 拡張ログ記録の有効化。この設定を有効にして、プロキシの互換性やパフォーマンスの問題のトラブルシューティングを行うことができます。

  この設定を有効にすると、RDS Proxy はプロキシのパフォーマンスに関する詳細情報をログに含めます。この情報に基づいて、SQL の動作やプロキシ接続のパフォーマンスとスケーラビリティに関する問題をデバッグできます。したがって、この設定を有効にするのは、デバッグのため、およびログ内の機密情報を保護するセキュリティ対策がある場合に限ります。

  プロキシに関連するオーバーヘッドを最小限に抑えるために、この設定は有効にしてから 24 時間後に RDS Proxy によって自動的にオフにされます。特定の問題のトラブルシューティングを行うには、その設定を一時的に有効にします。

5. [Create proxy (プロキシの作成)] を選択します。

AWS CLI

AWS CLI を使用してプロキシを作成するには、以下の必須パラメータを指定して create-db-proxy コマンドを呼び出します。

- `--db-proxy-name`
- `--engine-family`
- `--role-arn`
- `--auth`
- `--vpc-subnet-ids`

--engine-family 値では、大文字と小文字が区別されます。

Example

Linux、macOS、Unix の場合:

```
aws rds create-db-proxy \
    --db-proxy-name proxy_name \
    --engine-family { MYSQL | POSTGRESQL | SQLSERVER } \
    --auth ProxyAuthenticationConfig_JSON_string \
    --role-arn iam_role \
    --vpc-subnet-ids space_separated_list \
    [--vpc-security-group-ids space_separated_list] \
    [--require-tls | --no-require-tls] \
    [--idle-client-timeout value] \
    [--debug-logging | --no-debug-logging] \
    [--tags comma_separated_list]
```

Windows の場合:

```
aws rds create-db-proxy ^
    --db-proxy-name proxy_name ^
    --engine-family { MYSQL | POSTGRESQL | SQLSERVER } ^
    --auth ProxyAuthenticationConfig_JSON_string ^
    --role-arn iam_role ^
    --vpc-subnet-ids space_separated_list ^
    [--vpc-security-group-ids space_separated_list] ^
    [--require-tls | --no-require-tls] ^
    [--idle-client-timeout value] ^
    [--debug-logging | --no-debug-logging] ^
    [--tags comma_separated_list]
```

以下は、--auth オプションの JSON 値の例です。この例では、各シークレットに異なるクライアント認証タイプを適用します。

```
[
  {
    "Description": "proxy description 1",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:123456789123:secret/1234abcd-12ab-34cd-56ef-1234567890ab",
    "IAMAuth": "DISABLED",
```

```
    "ClientPasswordAuthType": "POSTGRES_SCRAM_SHA_256"
  },

  {
    "Description": "proxy description 2",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:111122223333:seret/1234abcd-12ab-34cd-56ef-1234567890cd",
    "IAMAuth": "DISABLED",
    "ClientPasswordAuthType": "POSTGRES_MD5"

  },

  {
    "Description": "proxy description 3",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:111122221111:secret/1234abcd-12ab-34cd-56ef-1234567890ef",
    "IAMAuth": "REQUIRED"
  }

]
```

Tip

`--vpc-subnet-ids` パラメータに使用するサブネット ID がまだわからない場合は、「ネットワーク前提条件の設定」を使用して、ID を検索する方法の例を参照してください。

Note

セキュリティグループは、プロキシの接続先のデータベースへのアクセスを許可する必要があります。同じセキュリティグループが、アプリケーションからプロキシへのイングレスと、プロキシからデータベースへのエグレスに使用されます。例えば、データベースとプロキシに同じセキュリティグループを使用するとします。この場合は必ず、そのセキュリティグループ内のリソースが同じセキュリティグループ内の他のリソースと通信できるように指定してください。

共有 VPC を使用する場合、VPC のデフォルトのセキュリティグループや、別のアカウントに属しているセキュリティグループを使用することはできません。自分のアカウントに属し

ているセキュリティグループを選択します。存在しない場合は、作成します。この制限事項の詳細については、共有 VPC の操作を参照してください。

プロキシに適切な関連付けを作成するには、register-db-proxy-targets コマンドも使用します。ターゲットグループ名 `default` を指定します。RDS Proxy は、各プロキシを作成するときに、この名前のターゲットグループを自動的に作成します。

```
aws rds register-db-proxy-targets
    --db-proxy-name value
    [--target-group-name target_group_name]
    [--db-instance-identifiers space_separated_list]  # rds db instances, or
    [--db-cluster-identifiers cluster_id]        # rds db cluster (all instances)
```

RDS API

RDS Proxy を作成するには、Amazon RDS API オペレーション CreateDBProxy を呼び出します。AuthConfig データ構造でパラメータを渡します。

RDS Proxy は、各プロキシを作成するときに、`default` という名前のターゲットグループを自動的に作成します。RegisterDBProxyTargets 関数を呼び出して、このターゲットグループに Aurora DB クラスターを関連付けます。

RDS Proxy の表示

1 つまたは複数の RDS プロキシを作成すると、すべてのプロキシを表示できます。これにより、設定の詳細を確認して、変更や削除などを行う設定を選択できます。

データベースアプリケーションがプロキシを使用するためには、接続文字列でプロキシエンドポイントを指定する必要があります。

AWS Management Console

プロキシを表示するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. AWS Management Console の右上隅で、RDS Proxy を作成した AWS リージョンを選択します。

3. ナビゲーションペインで、[プロキシ] を選択します。
4. 詳細を表示する RDS Proxy の名前を選択します。
5. 詳細ページの [ターゲットグループ] セクションに、プロキシと特定の Aurora DB クラスターとの関連付けが表示されます。[デフォルト] ターゲットグループページへのリンクに従って、プロキシとデータベースの関連付けの詳細を表示できます。このページでは、プロキシの作成時に指定した設定を確認できます。これには、最大接続数の割合 (%)、接続借用タイムアウト、エンジンファミリー、セッション固定フィルターが含まれます。

CLI

CLI を使用してプロキシを表示するには、describe-db-proxies コマンドを使用します。デフォルトでは、AWS アカウントが所有するすべてのプロキシが表示されます。単一のプロキシの詳細を表示するには、 `--db-proxy-name` パラメータで名前を指定します。

```
aws rds describe-db-proxies [--db-proxy-name proxy_name]
```

プロキシに関連付けられた他の情報を表示するには、以下のコマンドを使用します。

```
aws rds describe-db-proxy-target-groups  --db-proxy-name proxy_name

aws rds describe-db-proxy-targets --db-proxy-name proxy_name
```

プロキシに関連付けられている情報の詳細を表示するには、次のコマンドのシーケンスを使用します。

1. プロキシのリストを取得するには、describe-db-proxies を実行します。
2. プロキシが使用できる接続の最大割合などの接続パラメータを表示するには、describe-db-proxy-target-groups `--db-proxy-name` を実行します。プロキシの名前をパラメータ値として使用します。
3. 返されたターゲットグループに関連付けられている Aurora DB クラスターの詳細を表示するには、describe-db-proxy-targets を実行します。

RDS API

RDS API を使用してプロキシを表示するには、DescribeDBProxies オペレーションを使用します。DBProxy データ型の値が返されます。

プロキシの接続設定の詳細を表示するには、この戻り値のプロキシ識別子を DescribeDBProxyTargetGroups オペレーションで使用します。DBProxyTargetGroup データ型の値が返されます。

プロキシに関連付けられている RDS インスタンスや Aurora DB クラスターを表示するには、DescribeDBProxyTargets オペレーションを使用します。DBProxyTarget データ型の値が返されます。

RDS Proxy を介したデータベースへの接続

Aurora DB クラスター、またはプロキシを介してAurora Serverless v2 を使用するクラスターを接続する方法は、データベースに直接接続する方法と同じです。主な違いは、クラスターのエンドポイントの代わりに、プロキシのエンドポイントを指定することです。すべてのプロキシ接続にはデフォルトで読み取り/書き込み機能があり、ライターインスタンスを使用します。通常、読み取り専用の接続にリーダーエンドポイントを使用する場合は、プロキシ用の追加の読み取り専用エンドポイントを作成できます。そして、そのエンドポイントを同じ方法で使用できます。詳細については、「プロキシエンドポイントの概要」を参照してください。

トピック

- ネイティブ認証を使用したプロキシへの接続
- IAM 認証を使用したプロキシへの接続
- PostgreSQL でプロキシに接続する際の考慮事項

ネイティブ認証を使用したプロキシへの接続

以下のステップを使用して、ネイティブ認証を使用してプロキシに接続します。

1. プロキシエンドポイントを見つけます。AWS Management Console では、プロキシの詳細ページで対応するエンドポイントを見つけることができます。AWS CLI では、describe-db-proxies コマンドを使用できます。以下の例のように指定します。

```
# Add --output text to get output as a simple tab-separated list.
$ aws rds describe-db-proxies --query '*[*].
{DBProxyName:DBProxyName,Endpoint:Endpoint}'
[
    [
        {
            "Endpoint": "the-proxy.proxy-demo.us-east-1.rds.amazonaws.com",
```

```
            "DBProxyName": "the-proxy"
        },
        {
            "Endpoint": "the-proxy-other-secret.proxy-demo.us-
east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-other-secret"
        },
        {
            "Endpoint": "the-proxy-rds-secret.proxy-demo.us-
east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-rds-secret"
        },
        {
            "Endpoint": "the-proxy-t3.proxy-demo.us-east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-t3"
        }
    ]
]
```

2. クライアントアプリケーションの接続文字列で、そのエンドポイントをホストパラメータとして指定します。例えば、`mysql -h` オプションまたは `psql -h` オプションの値としてプロキシエンドポイントを指定します。
3. 通常と同じようにデータベースのユーザー名とパスワードを指定します。

IAM 認証を使用したプロキシへの接続

RDS Proxy で IAM 認証を使用する場合は、通常のユーザー名とパスワードで認証するようにデータベースユーザーを設定します。IAM 認証は、Secrets Manager からユーザー名とパスワードの認証情報 RDS Proxy を取得する場合に適用されます。RDS Proxy から基礎となるデータベースへの接続は IAM を経由しません。

IAM 認証を使用して RDS Proxy に接続するには、Aurora DB クラスターとの IAM 認証と同じ一般的な接続手順で行います。IAM の使用に関する一般情報については、「Amazon Aurora でのセキュリティ」を参照してください。

RDS Proxy での IAM の使用方法の主な違いは以下のとおりです。

- 認可プラグインで個々のデータベースユーザーを設定することはありません。データベースユーザーが通常使用するユーザー名とパスワードはデータベース内に保管されています。これらのユーザー名とパスワードを含む Secrets Manager シークレットを設定し、RDS Proxy に Secrets Manager からの認証情報の取得を認可します。

IAM 認証がクライアントプログラムとプロキシ間の接続に適用されます。その後、プロキシは、Secrets Manager から取得したユーザー名とパスワードの認証情報を使用して、データベースに対して認証します。

- インスタンス、クラスター、またはリーダーの各エンドポイントの代わりに、プロキシエンドポイントを指定します。プロキシエンドポイントの詳細については、「IAM 認証を使用した DB クラスターへの接続」を参照してください。
- 直接 DB IAM 認証では、データベースユーザーを選択し、特別な認証プラグインで識別されるように設定します。その後、IAM 認証を使用してそれらのユーザーに接続できます。

  プロキシのユースケースでは、ユーザーのユーザー名とパスワード (ネイティブ認証) を含むシークレットをプロキシに提供します。次に、IAM 認証を使用してプロキシに接続します。ここでは、データベースエンドポイントではなく、プロキシエンドポイントで認証トークンを生成して実行します。また、指定したシークレットのユーザー名の 1 つと一致するユーザー名を使用します。
- IAM 認証を使用してプロキシに接続する場合は、Transport Layer Security (TLS) / Secure Sockets Layer (SSL) を使用していることを確認してください。

IAM ポリシーを変更することで、特定のユーザーにプロキシへのアクセスを許可できます。以下に例を示します。

```
"Resource": "arn:aws:rds-db:us-east-2:1234567890:dbuser:prx-ABCDEFGHIJKL01234/db_user"
```

PostgreSQL でプロキシに接続する際の考慮事項

PostgreSQL では、クライアントが PostgreSQL データベースへの接続をスタートする際は起動メッセージを送信します。このメッセージには、パラメータ名と値の文字列のペアが含まれています。詳細については、PostgreSQL ドキュメントの「PostgreSQL Message Formats」で `StartupMessage` を参照してください。

RDS Proxy を介して接続する場合、起動メッセージには、現在認識されている以下のパラメータを含めることができます。

- `user`
- `database`

起動メッセージには、以下の追加のランタイムパラメータを含めることもできます。

- application_name
- client_encoding
- DateStyle
- TimeZone
- extra_float_digits
- search_path

PostgreSQL メッセージの詳細については、PostgreSQL ドキュメントの「Frontend/Backend Protocol」を参照してください。

PostgreSQL では、JDBC を使用する場合、ピニングを回避するために以下の操作をお勧めします。

- 固定を回避するために、JDBC 接続パラメータ assumeMinServerVersion を 9.0 以上に設定します。これにより、JDBC ドライバーが SET extra_float_digits = 3 を実行するとき、接続の始動時に余分なラウンドトリップを実行しなくなります。
- 固定を回避するために、JDBC 接続パラメータ ApplicationName を *any/your-application-name* に設定します。これを行うと、JDBC ドライバーが SET application_name = "PostgreSQL JDBC Driver" を実行するとき、接続のスタートアップ中に余分なラウンドトリップを実行しなくなります。JDBC パラメータは ApplicationName ですが、PostgreSQL StartupMessage パラメータは application_name です。

詳細については、「固定を回避する」を参照してください。JDBC を使用した接続の詳細については、PostgreSQL ドキュメントの「Connecting to the Database」を参照してください。

RDS Proxy の管理

このセクションでは、RDS Proxy の操作と設定を管理する方法について説明します。以下の手順は、アプリケーションがデータベース接続を最も効率的に使用し、接続を最大限に再利用するのに役立ちます。接続の再利用率を高めるほど、CPU とメモリのオーバーヘッドを減らすことができます。これにより、アプリケーションのレイテンシーを減らし、データベースのより多くのリソースをアプリケーションからのリクエストの処理に集中させることができます。

トピック

- RDS Proxy の変更
- 新しいデータベースユーザーの追加

- データベースユーザーのパスワードの変更
- クライアント接続とデータベース接続
- 接続設定の構成
- 固定を回避する
- RDS Proxy の削除

RDS Proxy の変更

プロキシの作成後に、プロキシに関連付けられた固有の設定を変更できます。これを行うには、プロキシ自体、プロキシに関連付けられているターゲットグループ、またはその両方を変更します。各プロキシには、ターゲットグループが関連付けられています。

AWS Management Console

> **Important**
>
> クライアント認証タイプと IAM 認証フィールドの値は、このプロキシに関連付けられているすべての Secrets Manager シークレットに適用されます。シークレットごとに異なる値を指定するには、代わりに AWS CLI または API を使用してプロキシを変更します。

プロキシの設定を変更するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[プロキシ] を選択します。
3. プロキシのリストで、設定を変更するプロキシを選択するか、その詳細ページに移動します。
4. [アクション]、[変更] の順に選択します。
5. 変更するプロパティを入力または選択します。次を変更できます。
   - プロキシ識別子 - プロキシの名前を変更するには、新しい識別子を入力します。
   - アイドル状態のクライアント接続のタイムアウト - アイドル状態のクライアント接続のタイムアウトの時間を入力します。
   - IAM ロール - Secrets Manager からシークレットを取得するために使用する IAM ロールを変更します。

- Secrets Manager シークレット - Secrets Manager シークレットを追加または削除します。これらのシークレットは、データベースのユーザー名とパスワードに対応します。
- クライアント認証タイプ – (PostgreSQL のみ) プロキシへのクライアント接続の認証タイプを変更します。
- IAM 認証 - プロキシへの接続に IAM 認証をすることを要求または禁止します。
- Transport Layer Security が必要 - Transport Layer Security (TLS) の要件を有効または無効にします。
- VPC セキュリティグループ - プロキシで使用する VPC セキュリティグループを追加または削除します。
- 拡張されたログ記録を有効にする - 拡張ログ記録を有効または無効にします。

6. [変更] を選択します。

変更する設定がリストにない場合は、以下の手順を使用して、プロキシのターゲットグループを更新します。プロキシに関連付けられているターゲットグループは、物理データベース接続に関連する設定を制御します。プロキシごとに `default` という名前の 1 つのターゲットグループが関連付けられています。このターゲットグループはプロキシと共に自動的に作成されます。

ターゲットグループは、プロキシの詳細ページからのみ変更できます。[プロキシ] ページのリストから変更することはできません。

プロキシのターゲットグループの設定を変更するには

1. [プロキシ] ページから、プロキシの詳細ページに移動します。
2. [ターゲットグループ] で、`default` リンクを選択します。現在、すべてのプロキシには `default` という名前のターゲットグループが 1 つ あります。
3. [デフォルト] ターゲットグループの詳細ページで、[変更] を選択します。
4. 変更できるプロパティに対して新しい設定を選択します。

   - データベース - 別の Aurora クラスターを選択します。
   - 接続プールの最大接続数 - プロキシで使用できる最大接続数の割合 (%) を調整できます。
   - セッション固定フィルター - (オプション) セッション固定フィルタを選択します。これにより、クライアント接続間でデータベース接続を多重化するというデフォルトの安全対策が回避されます。現在、設定は PostgreSQL についてはサポートされていません。唯一の選択肢は `EXCLUDE_VARIABLE_SETS` です。

この設定を有効にすると、ある接続のセッション変数が他の接続に影響を与える可能性があります。これにより、クエリが現在のトランザクション以外に設定されたセッション変数値に依存している場合、エラーや正確性の問題が発生する可能性があります。アプリケーションがクライアント接続間でデータベース接続を共有しても安全であることを確認した後に、このオプションの使用を検討してください。

以下のパターンは安全だと考えられます。

- 有効なセッション変数値に変更がない、つまりセッション変数に変更がない SET ステートメント。
- セッション変数の値を変更し、同じトランザクションでステートメントを実行します。

詳細については、「固定を回避する」を参照してください。

- 接続借用タイムアウト - 接続借用タイムアウト間隔を調整します。この設定は、プロキシで最大数の接続が既に使用されている場合に適用されます。この設定により、プロキシがタイムアウトエラーを返す前に、接続が使用可能になるまで待つ時間が決まります。
- 初期化クエリ - (オプション) 初期化クエリを追加するか、現在のクエリを変更します。新しい各データベース接続を開くときに実行するプロキシ用の 1 つ以上の SQL ステートメントを指定できます。設定は通常、各接続のタイムゾーンや文字セットなどの設定が同一であることを確認するために、SET ステートメントとともに使用されます。複数のステートメントの場合は、セミコロンをセパレータとして使用します。例えば、1 つの SET ステートメントに `SET x=1, y=2` など複数の可変を含めることもできます。

ターゲットグループ識別子やデータベースエンジンなどの特定のプロパティは変更できません。

5. [Modify target group (ターゲットグループの変更)] を選択します。

AWS CLI

AWS CLI を使用してプロキシを変更するには、modify-db-proxy、modify-db-proxy-target-group、deregister-db-proxy-targets、register-db-proxy-targets の各コマンドを使用します。

`modify-db-proxy` コマンドを使用すると、次のようなプロパティを変更できます。

- プロキシで使用する一連の Secrets Manager シークレット。
- TLS が必要かどうか。
- アイドルクライアントのタイムアウト。

- デバッグ用に SQL ステートメントからの追加情報をログに記録するかどうか。
- Secrets Manager シークレットの取得に使用する IAM ロール。
- プロキシで使用するセキュリティグループ。

次の例は、既存のプロキシの名前を変更する方法を示しています。

```
aws rds modify-db-proxy --db-proxy-name the-proxy --new-db-proxy-name the_new_name
```

接続関連の設定を変更したり、ターゲットグループの名前を変更したりするには、modify-db-proxy-target-group コマンドを使用します。現在、すべてのプロキシには default という名前のターゲットグループが 1 つ あります。このターゲットグループを使用する場合、プロキシの名前とターゲットグループ名 (default) を指定します。

次の例は、初期にプロキシの MaxIdleConnectionsPercent 設定をチェックし、次にターゲットグループを使用して設定を変更する方法を示しています。

```
aws rds describe-db-proxy-target-groups --db-proxy-name the-proxy

{
    "TargetGroups": [
        {
            "Status": "available",
            "UpdatedDate": "2019-11-30T16:49:30.342Z",
            "ConnectionPoolConfig": {
                "MaxIdleConnectionsPercent": 50,
                "ConnectionBorrowTimeout": 120,
                "MaxConnectionsPercent": 100,
                "SessionPinningFilters": []
            },
            "TargetGroupName": "default",
            "CreatedDate": "2019-11-30T16:49:27.940Z",
            "DBProxyName": "the-proxy",
            "IsDefault": true
        }
    ]
}

aws rds modify-db-proxy-target-group --db-proxy-name the-proxy --target-group-name
 default --connection-pool-config '
{ "MaxIdleConnectionsPercent": 75 }'
```

```
{
    "DBProxyTargetGroup": {
        "Status": "available",
        "UpdatedDate": "2019-12-02T04:09:50.420Z",
        "ConnectionPoolConfig": {
            "MaxIdleConnectionsPercent": 75,
            "ConnectionBorrowTimeout": 120,
            "MaxConnectionsPercent": 100,
            "SessionPinningFilters": []
        },
        "TargetGroupName": "default",
        "CreatedDate": "2019-11-30T16:49:27.940Z",
        "DBProxyName": "the-proxy",
        "IsDefault": true
    }
}
```

deregister-db-proxy-targets コマンドと register-db-proxy-targets コマンドでは、ターゲットグループを通じてプロキシが関連付けられている Aurora DB クラスターを変更します。現在、各プロキシが接続できる Aurora DB クラスターは 1 つです。ターゲットグループは、Aurora クラスター内のすべての DB インスタンスの接続の詳細を追跡します。

次の例では、cluster-56-2020-02-25-1399 という名前の Aurora MySQL クラスターに関連付けられているプロキシを初期に使用します。次に、このプロキシを変更して provisioned-cluster という名前の別のクラスターに接続できるようにします。

Aurora DB クラスターを使用する場合は、--db-cluster-identifier オプションを指定します。

次の例では、Aurora MySQL プロキシを変更します。Aurora PostgreSQL プロキシにはポート 5432 があります。

```
aws rds describe-db-proxy-targets --db-proxy-name the-proxy

{
    "Targets": [
        {
            "Endpoint": "instance-9814.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-9814"
        },
```

```
        {
            "Endpoint": "instance-8898.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-8898"
        },
        {
            "Endpoint": "instance-1018.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-1018"
        },
        {
            "Type": "TRACKED_CLUSTER",
            "Port": 0,
            "RdsResourceId": "cluster-56-2020-02-25-1399"
        },
        {
            "Endpoint": "instance-4330.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-4330"
        }
    ]
}

aws rds deregister-db-proxy-targets --db-proxy-name the-proxy --db-cluster-identifier
 cluster-56-2020-02-25-1399

aws rds describe-db-proxy-targets --db-proxy-name the-proxy

{
    "Targets": []
}

aws rds register-db-proxy-targets --db-proxy-name the-proxy --db-cluster-identifier
 provisioned-cluster

{
    "DBProxyTargets": [
        {
            "Type": "TRACKED_CLUSTER",
            "Port": 0,
            "RdsResourceId": "provisioned-cluster"
```

```
        },
        {
            "Endpoint": "gkldje.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "gkldje"
        },
        {
            "Endpoint": "provisioned-1.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "provisioned-1"
        }
    ]
}
```

RDS API

RDS API を使用してプロキシを変更するには、ModifyDBProxy、ModifyDBProxyTargetGroup、DeregisterDBProxyTargets、RegisterDBProxyTargets の各オペレーションを使用します。

`ModifyDBProxy` では、次のようなプロパティを変更できます。

- プロキシで使用する一連の Secrets Manager シークレット。
- TLS が必要かどうか。
- アイドルクライアントのタイムアウト。
- デバッグ用に SQL ステートメントからの追加情報をログに記録するかどうか。
- Secrets Manager シークレットの取得に使用する IAM ロール。
- プロキシで使用するセキュリティグループ。

`ModifyDBProxyTargetGroup` では、接続関連の設定や、ターゲットグループの名前を変更できます。現在、すべてのプロキシには `default` という名前のターゲットグループが 1 つ あります。このターゲットグループを使用する場合、プロキシの名前とターゲットグループ名 (`default`) を指定します。

`DeregisterDBProxyTargets` および `RegisterDBProxyTargets` では、ターゲットグループを通じてプロキシが関連付けられる Aurora クラスターを変更します。現在、各プロキシが接続でき

る Aurora DB クラスターは 1 つです。ターゲットグループは、 Aurora クラスターの DB インスタンスの接続の詳細を追跡します。

新しいデータベースユーザーの追加

状況に応じて、プロキシに関連付けられている Aurora クラスターに新しいデータベースユーザーを追加できます。その場合は、Secrets Manager シークレットを追加または転用して、そのユーザーの認証情報を保存します。これを行うには、次のいずれかのオプションを選択します。

1. 「AWS Secrets Manager でのデータベース認証情報の設定」で説明している手順を使用して、新しい Secrets Manager シークレットを作成します。
2. IAM ロールを更新して、RDS Proxy に新しい Secrets Manager シークレットへのアクセスを許可します。そのためには、IAM ロールポリシーのリソースセクションを更新します。
3. RDS Proxy を変更して、[Secrets Manager のシークレット] に新しい Secrets Manager のシークレットを追加します。
4. 既存のユーザーを新しいユーザーに置き換える場合は、プロキシで既存のユーザーの Secrets Manager シークレットに保存されている認証情報を更新します。

PostgreSQL データベースに新しいデータベースユーザーを追加する

PostgreSQL データベースに新しいユーザーを追加するとき、次のコマンドを実行する必要がある場合は、次のコマンドを実行します。

```
REVOKE CONNECT ON DATABASE postgres FROM PUBLIC;
```

ユーザーがターゲットデータベース上の接続をモニタリングできるように、`rdsproxyadmin` ユーザーに `CONNECT` 権限を付与します。

```
GRANT CONNECT ON DATABASE postgres TO rdsproxyadmin;
```

上記のコマンドで `rdsproxyadmin` をデータベースユーザーに変更することで、他のターゲットデータベースユーザーにヘルスチェックの実行を許可することもできます。

データベースユーザーのパスワードの変更

状況に応じて、プロキシに関連付けられている Aurora クラスターのデータベースユーザーのパスワードを変更できます。その場合は、対応する Secrets Manager シークレットを新しいパスワードで更新します。

クライアント接続とデータベース接続

アプリケーションから RDS Proxy への接続は、クライアント接続と呼ばれます。プロキシからデータベースへの接続はデータベース接続です。RDS Proxy を使用する場合、クライアント接続はプロキシで終了し、データベース接続は RDS Proxy 内で管理されます。

アプリケーション側の接続プーリングは、アプリケーションと RDS Proxy 間で繰り返し接続を確立する回数が減るというメリットを提供します。

アプリケーション側の接続プールを実装する前に、以下の設定について考慮してください。

- クライアント接続の最大有効期間: RDS Proxy では、クライアント接続の最大有効期間は 24 時間です。この値は設定できません。クライアント接続が予期せず切断されないように、プールは最大接続時間を 24 時間未満に設定してください。
- クライアント接続アイドルタイムアウト: RDS Proxy は、クライアント接続の最大アイドル時間を適用します。予期せぬ接続の切断を避けるため、RDS Proxy のクライアント接続アイドルタイムアウト設定よりも低い値のアイドル接続タイムアウトをプールに設定します。

アプリケーション側の接続プールに設定されるクライアント接続の最大数は、RDS Proxy の max_connections 設定に制限される必要はありません。

クライアント接続プールにより、クライアント接続の時間が長くなります。接続にピニングが発生した場合、クライアント接続をプールすると多重化の効率が低下する可能性があります。ピニングされているものの、アプリケーション側の接続プールでアイドル状態のクライアント接続は、引き続きデータベース接続を保持し、そのデータベース接続が他のクライアント接続で再利用されるのを防ぎます。プロキシログを確認して、接続にピニングが発生していないかチェックしてください。

> **Note**
>
> RDS Proxy は、データベース接続が使用されなくなった 24 時間後にその接続を閉じます。プロキシは、アイドル状態の最大接続数の設定値に関係なく、このアクションを実行します。

接続設定の構成

RDS Proxy の接続プーリングを調整するには、以下の設定を変更します。

- IdleClientTimeout

- MaxConnectionsPercent
- MaxIdleConnectionsPercent
- ConnectionBorrowTimeout

IdleClientTimeout

プロキシがクライアント接続を閉じるまでの間、接続がアイドル状態を継続できる時間を指定できます。デフォルトは 1,800 秒 (30 分) です。

クライアント接続がアイドル状態と見なされるのは、前のリクエストの完了後、新しいリクエストが指定時間内にアプリケーションから送信されない場合です。基となるデータベース接続は開いたままで、接続プールに返されるため、新しいクライアント接続で再利用できます。プロキシで古い接続を事前に削除する場合は、クライアント接続でのアイドル状態のタイムアウトを短くすることを検討してください。ワークロードが頻繁にプロキシとの接続を確立する場合には、接続コストを節約するために、クライアント接続でのアイドル状態のタイムアウトを長くすることを検討してください。

この設定は、RDS コンソール内の [Idle client connection timeout] (アイドルクライアントの接続タイムアウト) フィールドと、AWS CLI および API の `IdleClientTimeout` 設定で行います。RDS コンソールで [Idle client connection timeout] (アイドルクライアントの接続タイムアウト) フィールドの値を変更する方法については、「AWS Management Console」を参照してください。`IdleClientTimeout` 設定の値を変更するには、CLI コマンドの modify-db-proxy または API の ModifyDBProxy オペレーションを参照してください。

MaxConnectionsPercent

RDS Proxy がターゲットデータベースに対して確立できる接続の数を制限できます。上限は、データベースで使用可能な最大接続数に対する割合 (%) で指定します。この設定には、RDS Proxy コンソールの [接続プールの最大接続数] フィールド、または AWS CLI と API の `MaxConnectionsPercent` パラメータを使用します。

`MaxConnectionsPercent` 値はターゲットグループが使用する Aurora DB クラスターの `max_connections` 設定に対するパーセンテージで表されます。プロキシは、これらの接続のすべてを事前に作成するわけではありません。この設定では、ワークロードが必要とするときに、プロキシがこれらの接続を確立できます。

例えば、登録済みのデータベースターゲットの `max_connections` が 1000 に設定され、`MaxConnectionsPercent` が 95 に設定されている場合、RDS Proxy はそのデータベースターゲットへの同時接続の上限として 950 接続を設定します。

ワークロードが許容されるデータベース接続の最大数に達したときによく見られる副作用として、全体的なクエリ待ち時間が増加し、DatabaseConnectionsBorrowLatency メトリクスも増加します。DatabaseConnections メトリクスと MaxDatabaseConnectionsAllowed メトリクスを比較することで、現在使用中のデータベース接続数と許可されているデータベース接続の合計数を監視できます。

このパラメータを設定する場合は、次のベストプラクティスに注意してください。

- ワークロードパターンの変化に備えて、接続に十分な余裕を持たせてください。このパラメータは、最近監視した最大使用量より少なくとも 30% 高く設定することをお勧めします。RDS Proxy はデータベース接続クォータを複数のノードに再配分するため、内部容量の変更に伴い、借用レイテンシーの増加を避けるために、少なくとも 30% の余裕を持たせて接続を追加することが必要になる場合があります。
- RDS Proxy は、高速フェイルオーバー、トラフィックルーティング、内部オペレーションをサポートするために、アクティブモニタリング用に一定数の接続を予約します。MaxDatabaseConnectionsAllowed メトリクスには、これらの予約済み接続は含まれません。これはワークロードの処理に使用できる接続の数を表し、MaxConnectionsPercent 設定から算出された値よりも小さい場合があります。

  推奨される MaxConnectionsPercent の最小値は次のとおりです。

  - db.t3.small: 100
  - db.t3.medium: 55
  - db.t3.large: 35
  - db.r3.large またはそれ以上: 20

  読み取りノードを持つ Aurora クラスターのように RDS Proxy に複数のターゲットインスタンスが登録されている場合は、登録されている最小のインスタンスに基づいて最小値を設定します。

RDS コンソールの [Connection pool maximum connections] (接続プールの最大接続数) フィールドの値を変更する方法については、「AWS Management Console」を参照してください。MaxConnectionsPercent 設定での値の変更については、CLI コマンドの「modify-db-proxy-target-group」、または API オペレーションの「ModifyDBProxyTargetGroup」を参照してください。

> **Important**
>
> DB クラスターが、書き込み転送が有効になっているグローバルデータベースの一部である場合は、プロキシの MaxConnectionsPercent 値を書き込み転送に割り当てられたクォータだけ減らします。書き込み転送クォータは DB クラスターパラメータ aurora_fwd_writer_max_connections_pct で設定されます。書き込み転送の詳細については、「Amazon Aurora Global Database の書き込み転送を使用する」を参照してください。

データベース接続での制限の詳細については、「Aurora MySQL DB インスタンスへの最大接続数」、ならびに「Aurora PostgreSQL DB インスタンスへの最大接続数」を参照してください。

MaxIdleConnectionsPercent

RDS Proxy が接続プール内にアイドル状態で保持できる、データベース接続の数を制御できます。デフォルトでは、RDS Proxy は、接続に対するアクティビティが 5 分間なかった場合に、プール内のデータベース接続をアイドル状態とみなします。

上限は、データベースで使用可能な最大接続数に対する割合 (%) で指定します。そのデフォルト値は MaxConnectionsPercent の 50% で、上限は MaxConnectionsPercent の値で指定します。値を大きくすると、プロキシではアイドル状態のデータベース接続の大部分を開いたままにします。値を小さくすると、プロキシではアイドル状態のデータベース接続の大部分を閉じます。ワークロードが予測できない場合は、MaxIdleConnectionsPercent には大きな値を設定するように検討してください。これにより、RDS Proxy では多数の新しいデータベース接続を開くことなく、アクティビティの急増に対応できるようになります。

この設定には、AWS CLI と API における DBProxyTargetGroup の MaxIdleConnectionsPercent 設定を使用します。MaxIdleConnectionsPercent 設定での値の変更については、CLI コマンドの「modify-db-proxy-target-group」、または API オペレーションの「ModifyDBProxyTargetGroup」を参照してください。

データベース接続での制限の詳細については、「Aurora MySQL DB インスタンスへの最大接続数」、ならびに「Aurora PostgreSQL DB インスタンスへの最大接続数」を参照してください。

ConnectionBorrowTimeout

RDS Proxy がタイムアウトエラーを返す前に、接続プール内のデータベース接続が使用可能になるまで待つ時間を指定できます。デフォルト値は 120 秒です。この設定値は、接続数が最大値に達

し、接続プールで利用可能な接続がなくなった場合に適用されます。また、例えば、フェイルオーバー操作が進行中であるなどの理由で、リクエストを処理するために使用できる適切なデータベースインスタンスがない場合にも適用されます。この設定を使用すると、アプリケーションコードでクエリタイムアウトを変更しなくても、アプリケーションに最適な待機期間を設定できます。

この設定には、RDS Proxy コンソールの [接続借用タイムアウト] フィールド、または AWS CLI と API における `DBProxyTargetGroup` の `ConnectionBorrowTimeout` 設定を使用します。RDS コンソールの [Connection borrow timeout] (接続借用タイムアウト) フィールドの値を変更する方法については、「AWS Management Console」を参照してください。`ConnectionBorrowTimeout` 設定での値の変更については、CLI コマンドの「modify-db-proxy-target-group」、または API オペレーションの「ModifyDBProxyTargetGroup」を参照してください。

固定を回避する

データベースリクエストが以前のリクエストの状態情報に依存しない場合、多重化の効率が高まります。その場合、RDS Proxy は、各トランザクションの終了時に接続を再利用できます。このような状態情報の例には、`SET` ステートメントや `SELECT` ステートメントで変更できるほとんどの可変や設定パラメータが含まれます。クライアント接続の SQL トランザクションでは、デフォルトで、基となるデータベース接続を多重化できます。

プロキシへの接続は、固定と呼ばれる状態に入る場合があります。接続が固定されると、以降の各トランザクションは、セッションが終了するまで、同じ基になるデータベース接続を使用します。また、他のクライアント接続は、セッションが終了するまでそのデータベース接続を再利用できません。クライアント接続がドロップされると、セッションは終了します。

RDS Proxy は、他のセッションに不適切なセッション状態の変化を検出すると、クライアント接続を特定の DB 接続に自動的に固定します。固定により、接続の再利用の有効性が低下します。すべての接続やほぼすべての接続で固定が発生する場合は、固定が発生する状態を減らすようにアプリケーションコードやワークロードを変更します。

例えば、アプリケーションによってセッションの変数や設定パラメータが変更されたとします。この場合、後続のステートメントは変更後の変数やパラメータが有効かどうかによって変わります。したがって、RDS Proxy はセッションの可変や構成設定の変更リクエストを処理する場合、そのセッションを DB 接続に固定します。これにより、セッション状態は、同じセッション内の後続のすべてのトランザクションで有効になります。

一部のデータベースエンジンでは、設定可能なすべてのパラメータにこのルールが適用されるわけではありません。RDS Proxy は、特定のステートメントと変数を追跡します。したがって、これらの変更時に RDS Proxy はセッションを固定しません。この場合、RDS Proxy は、これらの設定の値が

同じである他のセッションでのみ、接続を再利用します。Aurora MySQL により追跡するステートメントと変数のリストについては、「RDS Proxy が Aurora MySQL データベースに対して追跡する内容」を参照してください。

RDS Proxy が Aurora MySQL データベースに対して追跡する内容

RDS Proxy が追跡する MySQL ステートメントを次に示します。

- DROP DATABASE
- DROP SCHEMA
- 使用

RDS Proxy が追跡する MySQL 変数を次に示します。

- AUTOCOMMIT
- AUTO_INCREMENT_INCREMENT
- CHARACTER SET (or CHAR SET)
- CHARACTER_SET_CLIENT
- CHARACTER_SET_DATABASE
- CHARACTER_SET_FILESYSTEM
- CHARACTER_SET_CONNECTION
- CHARACTER_SET_RESULTS
- CHARACTER_SET_SERVER
- COLLATION_CONNECTION
- COLLATION_DATABASE
- COLLATION_SERVER
- INTERACTIVE_TIMEOUT
- NAMES
- NET_WRITE_TIMEOUT
- QUERY_CACHE_TYPE
- SESSION_TRACK_SCHEMA
- SQL_MODE

- TIME_ZONE
- TRANSACTION_ISOLATION (or TX_ISOLATION)
- TRANSACTION_READ_ONLY (or TX_READ_ONLY)
- WAIT_TIMEOUT

固定を最小化する

RDS Proxy のパフォーマンスチューニングでは、固定を最小化してトランザクションレベルの接続の再利用 (多重化) を最大化します。

以下の方法で、固定を最小化できます。

- 固定の原因となる可能性のある不要なデータベースリクエストを避けます。
- すべての接続間で可変と構成設定を一貫して設定します。これにより、これらの特定の設定を持つ接続が後続のセッションで再利用される可能性が高くなります。

  ただし、PostgreSQL では、可変の設定によりセッションの固定が発生します。
- MySQL エンジンファミリデータベースの場合、セッション固定フィルターをプロキシに適用します。特定の種類のオペレーションがアプリケーションの正常な動作に影響しないことがわかっている場合、これらのオペレーションを除外してセッションの固定を起こさないように指定できます。
- Amazon CloudWatch メトリクス DatabaseConnectionsCurrentlySessionPinned をモニタリングして、固定が発生する頻度を確認します。このメトリクスおよび他の CloudWatch メトリクスの詳細については、「Amazon CloudWatch を使用した RDS Proxy メトリクスのモニタリング」を参照してください。
- SET ステートメントを使用して各クライアント接続を同等に初期化する場合、トランザクションレベルの多重化を維持したまま、この初期化を実行できます。この場合、初期セッション状態を設定するステートメントを、プロキシが使用する初期化クエリに移動します。このプロパティは、セミコロンで区切られた 1 つ以上の SQL ステートメントを含む文字列です。

  例えば、特定の設定パラメータを設定する初期化クエリをプロキシに定義できます。これにより、RDS Proxy でプロキシの新しい接続を設定するたびに、これらの設定が適用されます。トランザクションレベルの多重化を妨げないように、アプリケーションコードから対応する SET ステートメントを削除できます。

  プロキシでの固定の発生回数のメトリクスについては、「Amazon CloudWatch を使用した RDS Proxy メトリクスのモニタリング」を参照してください。

すべてのエンジンファミリーで固定が発生する条件

以下の場合は、多重化が予期しない動作をもたらす可能性があるため、プロキシはセッションを現在の接続に固定します。

- ステートメントのテキストサイズが 16 KB を超える場合、プロキシはセッションを固定します。

Aurora MySQL で固定が発生する条件

MySQL の場合、次の操作に伴って固定も発生します。

- 明示的なテーブルロックステートメントである `LOCK TABLE`、`LOCK TABLES`、または `FLUSH TABLES WITH READ LOCK` が原因でプロキシによるセッションの固定が発生します。
- `GET_LOCK` を使用して名前付きロックを作成するプロキシはセッションを固定します。
- ユーザー可変またはシステム可変 (例外あり) を設定した場合、プロキシはセッションを固定します。この状況によって接続の再利用が制限されすぎる場合は、`SET` 操作でピニングを発生させないように選択できます。セッションのピン留めフィルタープロパティを設定する方法については、「RDS Proxy の作成」または「RDS Proxy の変更」を参照してください。
- 一時テーブルを作成した場合、プロキシはセッションを固定します。これにより、トランザクションの境界に関係なく、一時テーブルの内容がセッション全体で保持されます。
- 関数 `ROW_COUNT`、`FOUND_ROWS`、および `LAST_INSERT_ID` を呼び出すと、固定が発生する場合があります。

  これらの関数によりピン留めが発生する正確な状況は、MySQL 5.7 と互換性のある Aurora MySQL のバージョン間で異なる場合があります。
- プリペアドステートメントの場合、プロキシはセッションを固定します。このルールは、プリペアドステートメントで SQL テキストを使用するか、バイナリプロトコルを使用するかに関係なく、適用されます。
- SET LOCAL を使用する場合、RDS Proxy は接続を固定しません。
- ストアドプロシージャやストアド関数を呼び出しても、固定は発生しません。RDS Proxy は、このような呼び出しに伴うセッション状態の変更を検出しません。トランザクション間でのセッション状態の維持を活用している場合は、アプリケーションがストアド ルーチン内でセッション状態を変更しないようにする必要があります。例えば、RDS プロキシは現在、すべてのトランザクションにわたって永続化される一時テーブルを作成するストアドプロシージャと互換性がありません。

アプリケーションの動作に関する専門知識がある場合は、特定のアプリケーションステートメントについて固定動作をスキップできます。これを行うには、プロキシの作成時に [セッションの固定フィルタ] オプションを選択します。現在、セッションの可変や構成設定の定義により発生するセッションの固定を回避できます。

Aurora PostgreSQL で固定が発生する条件

PostgreSQL の場合、次の操作に伴って固定も発生します。

- SET コマンドの使用
- PREPARE コマンド、DISCARD コマンド、DEALLOCATE コマンド、または EXECUTE コマンドを使用したプリペアドステートメントの管理
- 一時シーケンス、テーブル、またはビューの作成
- カーソルの宣言
- セッション状態の破棄
- 通知チャネルでのリッスン
- auto_explain などのライブラリモジュールのロード
- nextval や setval などの関数を使用したシーケンスの操作
- pg_advisory_lock や pg_try_advisory_lock などの関数を使用したロックの操作

> **Note**
>
> RDS プロキシは、トランザクションレベルのアドバイザリロック (特に pg_advisory_xact_lock、pg_advisory_xact_lock_shared、pg_try_advisory_xact_l をピン留めしません。

- パラメータの設定、またはパラメータのデフォルトへのリセット。具体的には、SET コマンドと set_config コマンドを使用して、セッション変数にデフォルト値を割り当てます。
- ストアドプロシージャやストアド関数を呼び出しても、固定は発生しません。RDS Proxy は、このような呼び出しに伴うセッション状態の変更を検出しません。トランザクション間でのセッション状態の維持を活用している場合は、アプリケーションがストアド ルーチン内でセッション状態を変更しないようにする必要があります。例えば、RDS プロキシは現在、すべてのトランザクションにわたって永続化される一時テーブルを作成するストアドプロシージャと互換性がありません。

RDS Proxy の削除

不要になったプロキシは削除できます。または、プロキシに関連付けられている DB インスタンスやクラスターがサービスから外された場合に、プロキシを削除できます。

AWS Management Console

プロキシを削除するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[プロキシ] を選択します。
3. リストから削除するプロキシを選択します。
4. [Delete Proxy (プロキシの削除)] を選択します。

AWS CLI

DB プロキシを削除するには、AWS CLI コマンド delete-db-proxy を使用します。該当する関連付けを削除するには、deregister-db-proxy-targets コマンドも使用します。

```
aws rds delete-db-proxy --name proxy_name
```

```
aws rds deregister-db-proxy-targets
    --db-proxy-name proxy_name
    [--target-group-name target_group_name]
    [--target-ids comma_separated_list]       # or
    [--db-instance-identifiers instance_id]       # or
    [--db-cluster-identifiers cluster_id]
```

RDS API

DB プロキシを削除するには、Amazon RDS API 関数 DeleteDBProxy を呼び出します。関連する項目と関連付けを削除するには、関数 DeleteDBProxyTargetGroup と DeregisterDBProxyTargets も呼び出します。

Amazon RDS Proxy エンドポイントの操作

RDS Proxy のエンドポイントとその使用方法について説明します。プロキシエンドポイントを使用すると、次の機能を活用できます。

- プロキシで複数のエンドポイントを使用して、異なるアプリケーションからの接続を個別にモニタリングおよびトラブルシューティングできます。
- Aurora DB クラスターでリーダーエンドポイントを使用して、クエリを多用するアプリケーションの読み取りスケーラビリティと可用性を向上させることができます。
- クロス VPC エンドポイントを使用して、ある VPC のデータベースに別の VPC の Amazon EC2 インスタンスなどのリソースからアクセスできるようにすることができます。

トピック

プロキシエンドポイントの概要

RDS Proxy エンドポイントを使用する際は、Aurora DB クラスターとリーダーエンドポイント と同じ種類の手順に従います。Aurora エンドポイントに詳しくない場合は、Amazon Aurora 接続管理 で詳細を確認してください。

デフォルトでは、RDS Proxy を Aurora クラスターで使用するときに接続するエンドポイントには読み取り/書き込み機能があります。そのため、このエンドポイントではすべてのリクエストをクラスターのライターインスタンスに送信します。これらの接続はすべて、ライターインスタンスの `max_connections` 値にカウントされます。プロキシが Aurora DB クラスターに関連付けられている場合は、そのプロキシ用に追加の読み取り/書き込みエンドポイントまたは読み取り専用エンドポイントを作成できます。

プロキシで読み取り専用エンドポイントを使用すると、読み取り専用クエリを実行できます。これは、Aurora のプロビジョニング済みクラスターにリーダーエンドポイントを使用するのと同じ方法です。そうすることで、1 つ以上のリーダー DB インスタンスを持つ Aurora クラスターの読み取りスケーラビリティを活用するのに役立ちます。読み取り専用エンドポイントを使用し、必要に応じ

て Aurora クラスターにリーダー DB インスタンスを追加することで、より多くの同時クエリを実行し、より多くの同時接続を確立できます。

> **Tip**
>
> AWS Management Console を使用して Aurora クラスターのプロキシを作成する場合、RDS Proxy にリーダーエンドポイントを自動で作成できます。リーダーエンドポイントの利点については、「Aurora クラスターでのリーダーエンドポイントの使用」を参照してください。

作成したプロキシエンドポイントについては、プロキシ自体が使用するものとは異なる Virtual Private Cloud (VPC) にエンドポイントを関連付けることもできます。これにより、組織内の別のアプリケーションで使用される VPC など、別の VPC からプロキシに接続できます。

プロキシエンドポイントに関連付けられた制限の詳細については、「プロキシエンドポイントの制限」を参照してください。

RDS Proxy ログでは、各エントリの前に、関連付けられたプロキシエンドポイントの名前が付けられます。この名前には、ユーザー定義のエンドポイントに指定した名前を使用できます。または、読み取り/書き込みリクエストを実行するプロキシのデフォルトエンドポイントの特別な名前 `default` にすることができます。

各プロキシのエンドポイントには、独自の CloudWatch メトリクスのセットがあります。プロキシのすべてのエンドポイントのメトリクスをモニタリングできます。また、プロキシの特定のエンドポイント、またはそのすべての読み取り/書き込みまたは読み取り専用エンドポイントのメトリクスをモニタリングすることもできます。詳細については、「Amazon CloudWatch を使用した RDS Proxy メトリクスのモニタリング」を参照してください。

プロキシエンドポイントは、関連付けられたプロキシと同じ認証メカニズムを使用します。RDS Proxy は、関連付けられたプロキシのプロパティと整合させて、ユーザー定義のエンドポイントのアクセス許可と認可を自動的に設定します。

プロキシエンドポイントが Aurora グローバルデータベースの DB クラスターでどのように機能するかについては、「RDS Proxy エンドポイントとグローバルデータベースの連携について」を参照してください。

Aurora クラスターでのリーダーエンドポイントの使用

RDS Proxy を Aurora クラスターで使用する場合は、リーダーエンドポイントと呼ばれる読み取り専用エンドポイントを作成して接続できます。このリーダーエンドポイントは、クエリを多用するアプ

リケーションの読み取りスケーラビリティを向上させるのに役立ちます。また、リーダーエンドポイントは、クラスター内のリーダー DB インスタンスが使用できなくなった場合に接続の可用性を向上させるのに役立ちます。

> **Note**
>
> 新しいエンドポイントが読み取り専用であることを指定する場合、RDS Proxy は Aurora クラスターに 1 つ以上のリーダー DB インスタンスがあることを必要とします。場合によっては、プロキシのターゲットを、シングルライターのみを含む Aurora クラスターまたはマルチライター Aurora クラスターに変更することがあります。この場合、リーダーエンドポイントへのリクエストはすべてエラーになり、失敗します。プロキシのターゲットが Aurora クラスターではなく RDS インスタンスである場合も、リクエストは失敗します。
> Aurora クラスターにリーダーインスタンスがあるが、それらのインスタンスが利用できない場合、RDS Proxy はすぐにエラーを返すのではなく、リクエストの送信を待ちます。接続借用タイムアウト期間内にリーダーインスタンスが使用可能にならなかった場合、リクエストはエラーで失敗します。

リーダーエンドポイントがアプリケーションの可用性をどのように支援するか

場合によっては、クラスター内の 1 つ以上のリーダーインスタンスが使用できなくなることがあります。その場合、DB プロキシのリーダーエンドポイントを使用する接続は、Aurora リーダーエンドポイントを使用する接続よりも迅速に回復できます。RDS Proxy は、クラスター内の利用可能なリーダーインスタンスのみに接続をルーティングします。インスタンスが使用できなくなると、DNS キャッシュによる遅延はありません。

接続が多重化されている場合、RDS Proxy はアプリケーションを中断せずに、後続のクエリを別のリーダー DB インスタンスに転送します。新しいリーダーインスタンスへの自動切り替え中に、RDS Proxy は古いリーダーインスタンスと新しいリーダーインスタンスのレプリケーションラグをチェックします。RDS Proxy は、以前のリーダーインスタンスと同じ変更を加えて、新しいリーダーインスタンスが最新の状態になるようにします。そうすれば、RDS Proxy があるリーダー DB インスタンスから別のリーダー DB インスタンスに切り替えたときに、アプリケーションで古いデータが表示されることはありません。

接続が固定されている場合、接続の次のクエリはエラーを返します。ただし、アプリケーションはすぐに同じエンドポイントに再接続できます。RDS Proxy は、`available` 状態の別のリーダー DB インスタンスに接続をルーティングします。手動で再接続する場合、RDS Proxy は古いリーダーインスタンスと新しいリーダーインスタンス間のレプリケーションラグをチェックしません。

Aurora クラスターに使用可能なリーダーインスタンスがない場合は、RDS Proxy はこの状態が一時的であるか永続的であるかを確認します。それぞれの場合の動作は次のとおりです。

- クラスターに 1 つ以上のリーダー DB インスタンスがありますが、いずれも `Available` 状態になっていないとします。例えば、すべてのリーダーインスタンスが再起動しているか、問題が発生している可能性があります。この場合、リーダーエンドポイントへの接続を試行し、リーダーインスタンスが利用可能になるまで待ちます。接続借用タイムアウト期間内にリーダーインスタンスが使用可能いならなかった場合、接続試行は失敗します。リーダーインスタンスが利用可能になると、接続試行は成功します。
- クラスターにリーダー DB インスタンスがないとします。その場合は、リーダーエンドポイントに接続しようとすると、RDS Proxy はすぐにエラーを返します。この問題を解決するには、リーダーエンドポイントに接続する前に、クラスターに 1 つ以上のリーダーインスタンスを追加します。

リーダーエンドポイントがクエリスケーラビリティにどのように役立つか

プロキシのリーダーエンドポイントは、次の方法で Aurora クエリのスケーラビリティを支援します。

- リーダーインスタンスを Aurora クラスターに追加すると、RDS Proxy は任意のリーダーエンドポイントへの新しい接続を別のリーダーインスタンスにルーティングできます。この方法では、1 つのリーダーエンドポイント接続を使用して実行されたクエリは、別のリーダーエンドポイント接続を使用して実行されるクエリの速度を低下させません。クエリは個別の DB インスタンスで実行されます。各 DB インスタンスには、独自のコンピューティングリソースやバッファキャッシュなどがあります。
- 実用的な場合は、RDS Proxy は特定のリーダーエンドポイント接続を使用するすべてのクエリの問題に同じリーダー DB インスタンスを使用します。これにより、同じテーブルに対する一連の関連クエリが、特定の DB インスタンスでキャッシング、プランの最適化などを活用できます。
- リーダー DB インスタンスが使用できなくなった場合、アプリケーションへの影響は、セッションが多重化されているか固定されているかによって異なります。セッションが多重化されている場合、RDS Proxy は、後続のクエリを、ユーザー側でアクションを実行せずに別のリーダー DB インスタンスにルーティングします。セッションが固定されている場合、アプリケーションはエラーを受け取り、再接続する必要があります。リーダーエンドポイントにすぐに再接続でき、RDS Proxy は、接続を利用可能なリーダー DB インスタンスにルーティングします。プロキシセッションの多重化および固定の詳細については、「RDS Proxy の概念」を参照してください。

- クラスター内のリーダー DB インスタンスの数が多いほど、リーダーエンドポイントを使用して作成できる同時接続が増えます。例えば、クラスターに 4 つのリーダー DB インスタンスがあり、それぞれが 200 の同時接続をサポートするように設定されているとします。また、プロキシが最大接続の 50% を使用するように設定されているとします。ここでは、プロキシのリーダーエンドポイントを介して確立できる接続の最大数は、リーダー 1 に対して 100 (200 の 50%) です。それはまた、リーダー 2 に対して 100 などと続き、合計で 400 になります。クラスターリーダー DB インスタンスの数を 2 倍の 8 とした場合、リーダーエンドポイントを経由する最大接続数は 2 倍の 800 になります。

リーダーエンドポイントの使用例

次の Linux の例は、リーダーエンドポイントを介して Aurora MySQL クラスターに接続していることを確認する方法を示しています。`innodb_read_only` 構成設定が有効になっています。`CREATE DATABASE` ステートメントなどの書き込み操作を実行しようとすると、エラーが発生して失敗します。また、`aurora_server_id` 可変を使用して DB インスタンス名を確認することで、リーダー DB インスタンスに接続していることを確認できます。

> **Tip**
>
> 接続が読み取り/書き込み可能か読み取り専用かを判断するために、DB インスタンス名の確認だけに頼らないでください。フェイルオーバーが発生すると、Aurora クラスター内の DB インスタンスがライターとリーダーの間でロールを変更する可能性があることに注意してください。

```
$ mysql -h endpoint-demo-reader.endpoint.proxy-demo.us-east-1.rds.amazonaws.com -u
 admin -p
...
mysql> select @@innodb_read_only;
+--------------------+
| @@innodb_read_only |
+--------------------+
|                  1 |
+--------------------+
mysql> create database shouldnt_work;
ERROR 1290 (HY000): The MySQL server is running with the --read-only option so it
 cannot execute this statement

mysql> select @@aurora_server_id;
```

```
+---------------------------------------+
| @@aurora_server_id                    |
+---------------------------------------+
| proxy-reader-endpoint-demo-instance-3 |
+---------------------------------------+
```

次の例は、リーダー DB インスタンスが削除された場合でも、プロキシリーダーエンドポイントへの接続がどのように機能するかを示しています。この例では、Aurora クラスターには、instance-5507 と instance-7448 の 2 つのリーダーインスタンスがあります。リーダーエンドポイントへの接続は、リーダーインスタンスの 1 つを使用してスタートします。この例の中で、delete-db-instance コマンドによってこのリーダーインスタンスは削除されます。RDS Proxy は、後続のクエリでは、別のリーダーインスタンスに切り替えます。

```
$ mysql -h reader-demo.endpoint.proxy-demo.us-east-1.rds.amazonaws.com
  -u my_user -p
...
mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-5507      |
+--------------------+

mysql> select @@innodb_read_only;
+--------------------+
| @@innodb_read_only |
+--------------------+
|                  1 |
+--------------------+

mysql> select count(*) from information_schema.tables;
+----------+
| count(*) |
+----------+
|      328 |
+----------+
```

mysql セッションの実行中、次のコマンドは、リーダーエンドポイントが接続されているリーダーインスタンスを削除します。

```
aws rds delete-db-instance --db-instance-identifier instance-5507 --skip-final-snapshot
```

mysql セッションでは、再接続しなくてもクエリが続きます。RDS Proxy は、自動的に別のリーダー DB インスタンスに切り替わります。

```
mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-7448      |
+--------------------+

mysql> select count(*) from information_schema.TABLES;
+----------+
| count(*) |
+----------+
|      328 |
+----------+
```

VPC 間の Aurora データベースへのアクセス

デフォルトでは、Aurora テクノロジースタックのコンポーネントはすべて同じ Amazon VPC にあります。例えば、Amazon EC2 インスタンスで実行されているアプリケーションが Aurora DB クラスター に接続するとします。この場合、アプリケーションサーバーとデータベースは両方とも同じ VPC 内に存在する必要があります。

RDS Proxy により、EC2 インスタンスなどのある VPC のリソースから、別の VPC の Aurora DB クラスター へのアクセスを設定できます。例えば、組織に、同じデータベースリソースにアクセスする複数のアプリケーションがあるとします。各アプリケーションは独自の VPC 内にある場合があります。

クロス VPC アクセスを有効にするには、プロキシの新しいエンドポイントを作成します。プロキシ自体は、 Aurora DB クラスター と同じ VPC に存在します。ただし、クロス VPC エンドポイントは、EC2 インスタンスなどの他のリソースとともに、他の VPC に存在します。クロス VPC エンドポイントは、EC2 および他のリソースと同じ VPC のサブネットおよびセキュリティグループに関連付けられます。このような関連付けにより、VPC の制限によりデータベースにアクセスできないアプリケーションからエンドポイントに接続できます。

次のステップでは、RDS Proxy を使用して VPC 間エンドポイントを作成してアクセスする方法について説明します。

1. 2 つの VPC を作成するか、Aurora の作業に既に使用している 2 つの VPC を選択します。各 VPC には、インターネットゲートウェイ、ルートテーブル、サブネット、セキュリティグループな

ど、独自のネットワークリソースが関連付けられている必要があります。VPC が 1 つしかない場合は、別の VPC をセットアップして正常に Aurora を使用できるようにするステップについて Amazon Aurora の開始方法 と相談できます。Amazon EC2 コンソールで既存の VPC を調べて、相互に接続するリソースの種類を確認することもできます。

2. 接続する Aurora DB クラスター に関連付けられた DB プロキシを作成します。「RDS Proxy の作成」 の手順に従います。
3. RDS コンソールのプロキシの [詳細] ページの [プロキシエンドポイント] セクションで、[エンドポイントの作成] を選択します。「プロキシエンドポイントの作成」 の手順に従います。
4. クロス VPC エンドポイントを読み取り/書き込み可能にするか、読み取り専用にするかを選択します。
5. Aurora DB クラスター と同じ VPC のデフォルトを受け入れる代わりに、別の VPC を選択します。この VPC は、プロキシが存在する VPC と同じ AWS リージョンに存在する必要があります。
6. これで、 Aurora DB クラスターと同じ VPC からサブネットとセキュリティグループのデフォルトを受け入れるのではなく、新しい選択を行います。選択した VPC のサブネットとセキュリティグループに基づいて、これらを作成します。
7. Secrets Manager シークレットの設定を変更する必要はありません。各エンドポイントがどの VPC にあるかに関係なく、プロキシのすべてのエンドポイントで同じ認証情報が機能します。
8. 新しいエンドポイントが Available (利用可能) 状態に達するのを待ちます。
9. 完全なエンドポイント名を書き留めます。これは、データベースアプリケーションの接続文字列の一部として指定する、*Region_name*`.rds.amazonaws.com` で終わる値です。
10. エンドポイントと同じ VPC 内のリソースから新しいエンドポイントにアクセスします。このプロセスをテストする簡単な方法は、この VPC 内に新しい EC2 インスタンスを作成することです。次に、EC2 インスタンスにログインし、`mysql` または `psql` コマンドを実行して、接続文字列のエンドポイント値を使用して接続します。

プロキシエンドポイントの作成

コンソール

プロキシエンドポイントを作成するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。

2. ナビゲーションペインで、[プロキシ] を選択します。
3. 新しいエンドポイントを作成するプロキシの名前をクリックします。

   そのプロキシの詳細ページが表示されます。
4. [プロキシエンドポイント] セクションで、[プロキシエンドポイントの作成] を選択します。

   [プロキシエンドポイントの作成] ウィンドウが表示されます。
5. [プロキシエンドポイント名] に、選択したわかりやすい名前を入力します。
6. [ターゲットロール] で、エンドポイントを読み取り/書き込みにするか、読み取り専用にするかを選択します。

   読み取り/書き込みエンドポイントを使用する接続では、データ定義言語 (DDL) ステートメント、データ操作言語 (DML) ステートメント、クエリなど、あらゆる種類の操作を実行できます。これらのエンドポイントは、常に Aurora クラスターのプライマリインスタンスに接続します。アプリケーションでエンドポイントを 1 つだけ使用する場合は、読み取り/書き込みエンドポイントを一般的なデータベース操作に使用できます。読み取り/書き込みエンドポイントは、管理操作、オンライントランザクション処理 (OLTP) アプリケーション、および抽出変換ロード (ETL) ジョブにも使用できます。

   読み取り専用エンドポイントを使用する接続では、クエリのみを実行できます。Aurora クラスターに複数のリーダーインスタンスがある場合、RDS Proxy はエンドポイントへの接続ごとに異なるリーダーインスタンスを使用できます。そうすれば、クエリを多用するアプリケーションは、Aurora のクラスタリング機能を利用できます。リーダー DB インスタンスを追加することで、クラスターにクエリ容量を追加できます。読み取り専用接続では、クラスターのプライマリインスタンスでオーバーヘッドが発生することはありません。このようにすると、レポートおよび分析クエリによって、OLTP アプリケーションの書き込みオペレーションの速度が低下することはありません。
7. 仮想プライベートクラウド (VPC) では、デフォルトで、プロキシまたはそれに関連付けられたデータベースへの通常のアクセスに使用するのと同じ EC2 インスタンスまたは他のリソースからエンドポイントにアクセスするように選択します。このプロキシに対してクロス VPC アクセスを設定するには、デフォルト以外の VPC を選択します。クロス VPC アクセスの詳細については、「VPC 間の Aurora データベースへのアクセス」を参照してください。
8. Subnets では、RDS Proxy がデフォルトで関連するプロキシと同じサブネットを入力します。VPC のアドレス範囲の一部のみがエンドポイントに接続できるように、エンドポイントへのアクセスを制限するには、1 つまたは複数のサブネットを削除します。

9. [VPC セキュリティグループ] で、既存のセキュリティグループを選択することも、新しいセキュリティグループを作成することもできます。RDS Proxy は、デフォルトで、関連付けられたプロキシと同じセキュリティグループを入力します。プロキシのインバウンドルールとアウトバウンドルールがこのエンドポイントに適切な場合は、デフォルトの選択肢のままにしておきます。

   新しいセキュリティグループを作成することにした場合は、このページでセキュリティグループの名前を指定します。その後、EC2 コンソールからセキュリティグループ設定を編集します。

10. [プロキシエンドポイントの作成] を選択します。

AWS CLI

プロキシエンドポイントを作成するには、AWS CLI create-db-proxy-endpoint コマンドを使用します。

以下の必須パラメータを含めます。

- `--db-proxy-name` *value*
- `--db-proxy-endpoint-name` *value*
- `--vpc-subnet-ids` *list_of_ids*. サブネット ID はスペースで区切ります。VPC 自体の ID は指定しません。

また、次のオプションパラメータを含めることができます。

- `--target-role { READ_WRITE | READ_ONLY }`このパラメータのデフォルトは です。READ_WRITEREAD_ONLY 値は、1 つ以上のリーダー DB インスタンスを含む Aurora プロビジョンドクラスターにのみ影響します。プロキシがライター DB インスタンスのみを含む Aurora クラスターに関連付けられている場合、READ_ONLY を指定することはできません。Aurora クラスターでの読み取り専用エンドポイントの使用目的の詳細については、「Aurora クラスターでのリーダーエンドポイントの使用 」を参照してください。
- `--vpc-security-group-ids` *value*。セキュリティグループ ID はスペースで区切ります。このパラメータを省略すると、RDS Proxy は VPC にデフォルトのセキュリティグループを使用します。RDS Proxy は、`--vpc-subnet-ids` パラメータとして指定したサブネット ID に基づいて VPC を決定します。

Example

次の例では、my-endpoint という名前のプロキシエンドポイントを作成します。

Linux、macOS、Unix の場合:

```
aws rds create-db-proxy-endpoint \
  --db-proxy-name my-proxy \
  --db-proxy-endpoint-name my-endpoint \
  --vpc-subnet-ids subnet_id subnet_id subnet_id ... \
  --target-role READ_ONLY \
  --vpc-security-group-ids security_group_id ]
```

Windows の場合:

```
aws rds create-db-proxy-endpoint ^
  --db-proxy-name my-proxy ^
  --db-proxy-endpoint-name my-endpoint ^
  --vpc-subnet-ids subnet_id_1 subnet_id_2 subnet_id_3 ... ^
  --target-role READ_ONLY ^
  --vpc-security-group-ids security_group_id
```

RDS API

プロキシエンドポイントを作成するには、RDS API CreateDBProxyEndpoint アクションを使用します。

プロキシエンドポイントの表示

コンソール

プロキシエンドポイントの詳細を表示するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[プロキシ] を選択します。
3. リストから、エンドポイントを表示するプロキシを選択します。プロキシ名をクリックして、詳細ページを表示します。
4. で、プロキシエンドポイントセクションで、表示するエンドポイントを選択します。名前をクリックすると、詳細ページが表示されます。

5. 関心のある値を持つパラメータを調べます。次のようなプロパティを確認できます。

   - エンドポイントが読み取り/書き込み可能か読み取り専用か。
   - データベース接続文字列で使用するエンドポイントアドレス。
   - エンドポイントに関連付けられた VPC、サブネットおよびセキュリティグループ。

AWS CLI

1 つ以上の DB プロキシエンドポイントを表示するには、AWS CLI describe-db-proxy-endpoints コマンドを使用します。

以下のオプションのパラメータを含めることができます。

- `--db-proxy-endpoint-name`
- `--db-proxy-name`

次の例では、`my-endpoint` プロキシエンドポイントについて説明します。

Example

Linux、macOS、Unix の場合:

```
aws rds describe-db-proxy-endpoints \
  --db-proxy-endpoint-name my-endpoint
```

Windows の場合:

```
aws rds describe-db-proxy-endpoints ^
  --db-proxy-endpoint-name my-endpoint
```

RDS API

1 つ以上のプロキシエンドポイントを記述するには、RDS API DescribeDBProxyEndpoints オペレーションを使用します。

プロキシエンドポイントの変更

コンソール

1 つまたは複数のプロキシエンドポイントを変更するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[プロキシ] を選択します。
3. リストから、エンドポイントを変更するプロキシを選択します。プロキシ名をクリックして、詳細ページを表示します。
4. [プロキシエンドポイント] セクションで、変更するエンドポイントを選択します。リストから選択するか、名前をクリックして詳細ページを表示できます。
5. プロキシの詳細ページの [プロキシエンドポイント] セクションで、[編集] を選択します。または、プロキシエンドポイントの詳細ページの [アクション] で、[編集] を選択します。
6. 変更するパラメータの値を変更します。
7. [Save changes] (変更を保存) をクリックします。

AWS CLI

プロキシエンドポイントを変更するには、次の必須パラメータで AWS CLI modify-db-proxy-endpoint コマンドを使用します。

- `--db-proxy-endpoint-name`

次のパラメータの 1 つまたは複数を使用して、エンドポイントプロパティの変更を指定します。

- `--new-db-proxy-endpoint-name`
- `--vpc-security-group-ids`。セキュリティグループ ID はスペースで区切ります。

次の例では、`my-endpoint` プロキシエンドポイントの名前を `new-endpoint-name` に変更します。

Example

Linux、macOS、Unix の場合:

```
aws rds modify-db-proxy-endpoint \
  --db-proxy-endpoint-name my-endpoint \
```

```
  --new-db-proxy-endpoint-name new-endpoint-name
```

Windows の場合:

```
aws rds modify-db-proxy-endpoint ^
  --db-proxy-endpoint-name my-endpoint ^
  --new-db-proxy-endpoint-name new-endpoint-name
```

RDS API

プロキシエンドポイントを変更するには、RDS API の ModifyDBProxyEndpoint 操作を使用します。

プロキシエンドポイントの削除

次の手順に従って、コンソールを使用してプロキシのエンドポイントを削除できます。

> **Note**
>
> RDS Proxy がプロキシごとに自動的に作成するデフォルトのプロキシエンドポイントを削除することはできません。
> プロキシを削除すると、RDS Proxy は、関連するすべてのエンドポイントを自動的に削除します。

コンソール

プロキシエンドポイントを削除するにはAWS Management Console

1. ナビゲーションペインで、[プロキシ] を選択します。
2. リストから、エンドポイントを設定するプロキシを選択します。プロキシ名をクリックして、詳細ページを表示します。
3. [プロキシエンドポイント] セクションで、削除するエンドポイントを選択します。リストから 1 つ以上のエンドポイントを選択するか、1 つのエンドポイントの名前をクリックして詳細ページを表示できます。
4. プロキシの詳細ページの [プロキシエンドポイント] セクションで、[削除] を選択します。または、プロキシエンドポイントの詳細ページの [アクション] で、[削除] を選択します。

AWS CLI

プロキシエンドポイントを削除するには、次の必須パラメータを指定して delete-db-proxy-endpoint コマンドを実行します。

- `--db-proxy-endpoint-name`

次のコマンドは、`my-endpoint` という名前のプロキシエンドポイントを削除します。

Linux、macOS、Unix の場合:

```
aws rds delete-db-proxy-endpoint \
  --db-proxy-endpoint-name my-endpoint
```

Windows の場合:

```
aws rds delete-db-proxy-endpoint ^
  --db-proxy-endpoint-name my-endpoint
```

RDS API

RDS API でプロキシエンドポイントを削除するには、DeleteDBProxyEndpoint オペレーションを実行します。`DBProxyEndpointName` パラメータのプロキシエンドポイントの名前を指定します。

プロキシエンドポイントの制限

RDS Proxy エンドポイントには以下の制限があります。

- 各プロキシには、変更できるが作成または削除できないデフォルトのエンドポイントがあります。
- プロキシのユーザー定義エンドポイントの最大数は 20 です。したがって、プロキシには最大 21 個のエンドポイント (デフォルトのエンドポイントとユーザーが作成する 20) を持つことができます。
- 追加のエンドポイントをプロキシに関連付けると、RDS Proxy は、クラスター内のどの DB インスタンスをエンドポイントごとに使用するかを自動的に決定します。Aurora カスタムエンドポイントの場合とは異なり、特定のインスタンスを選択することはできません。
- リーダーエンドポイントは、Aurora マルチライタークラスターでは使用できません。

Amazon CloudWatch を使用した RDS Proxy メトリクスのモニタリング

Amazon CloudWatch を使用して RDS Proxy のモニタリングができます。CloudWatch は、プロキシから raw データを収集し、リアルタイムに近い読み取り可能なメトリクスに加工します。これらのメトリクスを CloudWatch コンソールで確認するには、[Metrics (メトリクス)]、[RDS]、[Per-Proxy Metrics (プロキシごとのメトリクス)] の順に選択します。詳細については、Amazon CloudWatch ユーザーガイド の「Amazon CloudWatch メトリクスの使用」を参照してください。

> **Note**
>
> RDS は、これらのメトリクスを、プロキシに関連付けられている基になる Amazon EC2 インスタンスごとに発行します。1 つのプロキシは、複数の EC2 インスタンスによって処理される場合があります。CloudWatch の統計情報を使用して、すべての関連付けられたインスタンスにわたってプロキシの値を集計します。
> これらのメトリクスの一部は、プロキシによる初期の接続が成功するまで表示されないことがあります。

RDS Proxy ログでは、各エントリの前に、関連付けられたプロキシエンドポイントの名前が付けられます。この名前は、ユーザー定義のエンドポイントに指定した名前、または読み取り/書き込みリクエストを実行するプロキシのデフォルトエンドポイントの特別な名前 `default` にすることができます。

すべての RDS Proxy メトリクスはグループ `proxy` にあります。

各プロキシエンドポイントには独自の CloudWatch メトリクスがあります。各プロキシエンドポイントの使用状況を個別にモニタリングできます。プロキシエンドポイントの詳細については、「Amazon RDS Proxy エンドポイントの操作」を参照してください。

次のいずれかのディメンションセットを使用して、各メトリクスの値を集計できます。例えば、`ProxyName` ディメンションセットを使用すると、特定のプロキシのすべてのトラフィックを分析できます。他のディメンションセットを使用すると、さまざまな方法でメトリクスを分割できます。メトリクスは、各プロキシの異なるエンドポイントまたはターゲットデータベース、または各データベースへの読み取り/書き込みおよび読み取り専用のトラフィックに基づいて分割できます。

- ディメンションセット 1: `ProxyName`
- ディメンションセット 2:`ProxyName,EndpointName`

- ディメンションセット 3: ProxyName、TargetGroup、Target
- ディメンションセット 4: ProxyName、TargetGroup、TargetRole

メトリクス	説明	有効期間	CloudWatch ディメンションセット
AvailabilityPercentage	ディメンションで示されたロールでターゲットグループが利用できた時間の割合。このメトリクスは 1 分ごとにレポートされます。このメトリクスの最も有用な統計は Average です。	1 分	Dimension set 4
ClientConnections	現在のクライアント接続の数。このメトリクスは 1 分ごとにレポートされます。このメトリクスの最も有用な統計は Sum です。	1 分	Dimension set 1, Dimension set 2
ClientConnectionsClosed	閉じられたクライアント接続の数。このメトリクスの最も有用な統計は Sum です。	1 分以上	Dimension set 1, Dimension set 2
ClientConnectionsNoTLS	Transport Layer Security (TLS) を使用しない現在のクライアント接続の数。このメトリクスは 1 分ごとにレポートされます。このメトリク	1 分以上	Dimension set 1, Dimension set 2

メトリクス	説明	有効期間	CloudWatch ディメンションセット
	スの最も有用な統計は Sum です。		
ClientConnectionsReceived	受信したクライアント接続リクエストの数。このメトリクスの最も有用な統計は Sum です。	1 分以上	Dimension set 1, Dimension set 2
ClientConnectionsSetupFailedAuth	認証または TLS の設定ミスのために失敗したクライアント接続の試行回数。このメトリクスの最も有用な統計は Sum です。	1 分以上	Dimension set 1, Dimension set 2
ClientConnectionsSetupSucceeded	TLS の有無にかかわらず、認証機構を使用して正常に確立されたクライアント接続の数。このメトリクスの最も有用な統計は Sum です。	1 分以上	Dimension set 1, Dimension set 2
ClientConnectionsTLS	TLS を使用する現在のクライアント接続の数。このメトリクスは 1 分ごとにレポートされます。このメトリクスの最も有用な統計は Sum です。	1 分以上	Dimension set 1, Dimension set 2

メトリクス	説明	有効期間	CloudWatch ディメンションセット
DatabaseConnectionRequests	データベース接続の作成リクエストの数。このメトリクスの最も有用な統計は Sum です。	1 分以上	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionRequestsWithTLS	TLS を使用してデータベース接続を作成するリクエストの数。このメトリクスの最も有用な統計は Sum です。	1 分以上	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnections	現在のデータベース接続の数。このメトリクスは 1 分ごとにレポートされます。このメトリクスの最も有用な統計は Sum です。	1 分	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsBorrowLatency	モニタリングされているプロキシがデータベース接続を取得するのにかかる時間 (マイクロ秒)。このメトリクスの最も有用な統計は Average です。	1 分以上	Dimension set 1, Dimension set 2

メトリクス	説明	有効期間	CloudWatch ディメンションセット
DatabaseConnectionsCurrentlyBorrowed	借用状態のデータベース接続の現在の数。このメトリクスは 1 分ごとにレポートされます。このメトリクスの最も有用な統計は Sum です。	1 分	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsCurrentlyInTransaction	トランザクションでの現在のデータベース接続の数。このメトリクスは 1 分ごとにレポートされます。このメトリクスの最も有用な統計は Sum です。	1 分	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsCurrentlySessionPinned	セッション状態を変更するクライアントリクエストのオペレーションのために現在固定されているデータベース接続の数。このメトリクスは 1 分ごとにレポートされます。このメトリクスの最も有用な統計は Sum です。	1 分	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsSetupFailed	失敗したデータベース接続リクエストの数。このメトリクスの最も有用な統計は Sum です。	1 分以上	Dimension set 1, Dimension set 3, Dimension set 4

メトリクス	説明	有効期間	CloudWatch ディメンションセット
DatabaseConnectionsSetupSucceeded	TLS の有無にかかわらず、正常に確立されたデータベース接続の数。このメトリクスの最も有用な統計は Sum です。	1 分以上	Dimension set 1, Dimension set 3, Dimension set 4
DatabaseConnectionsWithTLS	TLS を使用する現在のデータベース接続の数。このメトリクスは 1 分ごとにレポートされます。このメトリクスの最も有用な統計は Sum です。	1 分	Dimension set 1, Dimension set 3, Dimension set 4
MaxDatabaseConnectionsAllowed	許可されるデータベース接続の最大数。このメトリクスは 1 分ごとにレポートされます。このメトリクスの最も有用な統計は Sum です。	1 分	Dimension set 1, Dimension set 3, Dimension set 4
QueryDatabaseResponseLatency	データベースがクエリに応答するのにかかった時間 (マイクロ秒)。このメトリクスの最も有用な統計は Average です。	1 分以上	Dimension set 1, Dimension set 2, Dimension set 3, Dimension set 4

メトリクス	説明	有効期間	CloudWatch ディメンションセット
QueryRequests	受信したクエリの数。複数のステートメントを含むクエリは、1 つのクエリとしてカウントされます。このメトリクスの最も有用な統計は Sum です。	1 分以上	Dimension set 1, Dimension set 2
QueryRequestsNoTLS	非 TLS 接続から受信したクエリの数。複数のステートメントを含むクエリは、1 つのクエリとしてカウントされます。このメトリクスの最も有用な統計は Sum です。	1 分以上	Dimension set 1, Dimension set 2
QueryRequestsTLS	TLS 接続から受信したクエリの数。複数のステートメントを含むクエリは、1 つのクエリとしてカウントされます。このメトリクスの最も有用な統計は Sum です。	1 分以上	Dimension set 1, Dimension set 2

メトリクス	説明	有効期間	CloudWatch ディメンションセット
QueryResponseLatency	クエリリクエストを取得してから、プロキシが応答するまでの時間 (マイクロ秒)。このメトリクスの最も有用な統計は Average です。	1 分以上	Dimension set 1, Dimension set 2

RDS Proxy アクティビティのログは、AWS Management Console の CloudWatch にあります。各プロキシには、[ロググループ] ページにエントリがあります。

> **Important**
>
> これらのログは、トラブルシューティングを目的としたもので、プログラムによるアクセス用ではありません。ログの形式と内容は変更される可能性があります。
> 特に、古いログには、各リクエストのエンドポイントを示すプレフィックスは含まれていません。新しいログでは、各エントリの先頭に、関連付けられたプロキシエンドポイントの名前が付けられます。この名前は、ユーザー定義のエンドポイントに指定した名前、またはプロキシのデフォルトのエンドポイントを使用するリクエストの特別な名前 default にすることができます。

RDS Proxy イベントの使用

イベントとは、AWS 環境やサービスまたは Software as a Service (SaaS) パートナーからのアプリケーションなどの環境での変化を示します。あるいは、お客様独自のカスタムアプリケーションやサービスのいずれかの場合があります。例えば、RDS Proxy を作成または変更すると、Amazon Aurora がイベントを生成します。Amazon Aurora は、Amazon EventBridge に対してほぼリアルタイムでイベントを配信します。以下に、サブスクライブできる RDS Proxy イベントのリストと、RDS Proxy イベントの例を示します。

イベントの操作に関する詳細は、以下を参照してください。

- を使用してイベントを表示する方法については、を参照してください。AWS Management Console、AWS CLI、または RDS API については、Amazon RDS イベントの表示を参照してください。
- 構成方法については、こちらをご覧ください。Amazon AuroraEventBridge にイベントを送信するには、「Amazon Aurora イベントでトリガーするルールの作成」を参照してください。

RDS Proxy イベント

ソースタイプが RDS Proxy である場合の、イベントのカテゴリとその一覧を次の表に示します。

カテゴリ	RDS イベント ID	メッセージ	メモ
設定変更	RDS-EVENT-0204	RDS が DB プロキシ *name* を変更しました。	
設定変更	RDS-EVENT-0207	RDS において、DB プロキシ *name* のエンドポイントが修正されました。	
設定変更	RDS-EVENT-0213	RDS が DB インスタンスの追加を検出し、そのインスタンスを DB プロキシ *name* のターゲットグループに自動的に追加しました。	
設定変更	RDS-EVENT-0213	RDS が DB インスタンス *name* の作成を検出し、そのインスタンスを DB プロキシ *name* のターゲットグループ *name* に自動的に追加しました。	
設定変更	RDS-EVENT-0214	RDS が DB インスタンス *name* の削除を検出し、そのインスタンスを DB プロキシ *name* のターゲット	

カテゴリ	RDS イベント ID	メッセージ	メモ
		グループ *name* から自動的に削除しました。	
設定変更	RDS-EVENT-0215	RDS が DB クラスター *name* の削除を検出し、そのインスタンスを DB プロキシ *name* のターゲットグループ *name* から自動的に削除しました。	
作成	RDS-EVENT-0203	RDS は DB プロキシ *name* を作成しました。	
作成	RDS-EVENT-0206	RDS は DB プロキシ *name* のエンドポイント*name*を作成しました。	
削除	RDS-EVENT-0205	RDS は DB プロキシ *name* を削除しました。	
削除	RDS-EVENT-0208	RDS は DB プロキシ*name*のエンドポイント *name* を削除しました。	
失敗	RDS-EVENT-0243	サブネット *name* に十分な IP アドレスがないため、RDS はプロキシの容量をプロビジョニングできませんでした: *name*。この問題を解決するには、RDS プロキシのドキュメントで推奨されているように、サブネットの未使用の IP アドレスが最小限であることを確認してください。	インスタンスクラスの推奨数を決定するには、「IP アドレス容量の計画」を参照してください。

カテゴリ	RDS イベント ID	メッセージ	メモ
失敗	RDS-EVENT-0275	RDS は DB プロキシ#への一部の接続をスロットリングしました。クライアントからプロキシへの同時接続リクエストの数が制限を超えました。	

以下は、JSON 形式の RDS Proxy イベントの例です。このイベントは、my-rds-proxy という名前の RDS Proxy の my-endpoint という名前のエンドポイントを、RDS が変更したことを示します。イベント ID は RDS-EVENT-0207 です。

```
{
  "version": "0",
  "id": "68f6e973-1a0c-d37b-f2f2-94a7f62ffd4e",
  "detail-type": "RDS DB Proxy Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-09-27T22:36:43Z",
  "region": "us-east-1",
  "resources": [
     "arn:aws:rds:us-east-1:123456789012:db-proxy:my-rds-proxy"
  ],
  "detail": {
    "EventCategories": [
      "configuration change"
    ],
    "SourceType": "DB_PROXY",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:db-proxy:my-rds-proxy",
    "Date": "2018-09-27T22:36:43.292Z",
    "Message": "RDS modified endpoint my-endpoint of DB Proxy my-rds-proxy.",
    "SourceIdentifier": "my-endpoint",
    "EventID": "RDS-EVENT-0207"
  }
}
```

RDS Proxy コマンドラインの例

接続コマンドと SQL ステートメントの組み合わせが RDS Proxy とやり取りする方法については、以下の例を参照してください。

例

- Preserving Connections to a MySQL Database Across a Failover
- Adjusting the max_connections Setting for an Aurora DB Cluster

Example フェイルオーバー全体における MySQL データベースへの接続の保持

この MySQL の例では、フェイルオーバー時に開いている接続が動作を継続させる方法を示します。例えば、データベースを再起動する場合や、問題が発生してデータベースを使用できない場合などです。この例では、`the-proxy` という名前のプロキシと、DB インスタンスとして `instance-8898` と `instance-9814` が含まれている Aurora DB クラスターを使用します。Linux コマンドラインから `failover-db-cluster` コマンドを実行すると、プロキシの接続先のライターインスタンスが別の DB インスタンスに変更されます。接続が開いたままで、プロキシに関連付けられている DB インスタンスが変更されることがわかります。

```
$ mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com -u admin_user -p
Enter password:
...

mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-9814      |
+--------------------+
1 row in set (0.01 sec)

mysql>
[1]+  Stopped                 mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com
 -u admin_user -p
$ # Initially, instance-9814 is the writer.
$ aws rds failover-db-cluster --db-cluster-identifier cluster-56-2019-11-14-1399
JSON output
$ # After a short time, the console shows that the failover operation is complete.
$ # Now instance-8898 is the writer.
$ fg
```

```
mysql -h the-proxy.proxy-demo.us.us-east-1.rds.amazonaws.com -u admin_user -p

mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-8898      |
+--------------------+
1 row in set (0.01 sec)

mysql>
[1]+  Stopped                 mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com
 -u admin_user -p
$ aws rds failover-db-cluster --db-cluster-identifier cluster-56-2019-11-14-1399
JSON output
$ # After a short time, the console shows that the failover operation is complete.
$ # Now instance-9814 is the writer again.
$ fg
mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com -u admin_user -p

mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-9814      |
+--------------------+
1 row in set (0.01 sec)
+---------------+---------------+
| Variable_name | Value         |
+---------------+---------------+
| hostname      | ip-10-1-3-178 |
+---------------+---------------+
1 row in set (0.02 sec)
```

Example Aurora DB クラスターの max_connections 設定の調整

この例では、Aurora MySQL DB クラスターの max_connections 設定を調整する方法を示します。そのためには、MySQL 5.7 と互換性のあるクラスターのデフォルトのパラメータ設定に基づいて、独自の DB クラスターパラメータグループを作成します。max_connections 設定の値を指定し、デフォルト値を設定する式を上書きします。DB クラスターパラメータグループを DB クラスターに関連付けます。

```
export REGION=us-east-1
```

```
export CLUSTER_PARAM_GROUP=rds-proxy-mysql-57-max-connections-demo
export CLUSTER_NAME=rds-proxy-mysql-57

aws rds create-db-parameter-group --region $REGION \
  --db-parameter-group-family aurora-mysql5.7 \
  --db-parameter-group-name $CLUSTER_PARAM_GROUP \
  --description "Aurora MySQL 5.7 cluster parameter group for RDS Proxy demo."

aws rds modify-db-cluster --region $REGION \
  --db-cluster-identifier $CLUSTER_NAME \
  --db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP

echo "New cluster param group is assigned to cluster:"
aws rds describe-db-clusters --region $REGION \
  --db-cluster-identifier $CLUSTER_NAME \
  --query '*[*].{DBClusterParameterGroup:DBClusterParameterGroup}'

echo "Current value for max_connections:"
aws rds describe-db-cluster-parameters --region $REGION \
  --db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP \
  --query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
  --output text | grep "^max_connections"

echo -n "Enter number for max_connections setting: "
read answer

aws rds modify-db-cluster-parameter-group --region $REGION --db-cluster-parameter-
group-name $CLUSTER_PARAM_GROUP \
  --parameters "ParameterName=max_connections,ParameterValue=$
$answer,ApplyMethod=immediate"

echo "Updated value for max_connections:"
aws rds describe-db-cluster-parameters --region $REGION \
  --db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP \
  --query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
  --output text | grep "^max_connections"
```

RDS Proxy のトラブルシューティング

以下に、いくつかの一般的な RDS Proxy 問題のトラブルシューティングのヒントと、RDS Proxy の CloudWatch ログに関する情報を示します。

RDS Proxy ログでは、各エントリの前に、関連付けられたプロキシエンドポイントの名前が付けられます。この名前には、ユーザー定義のエンドポイントに指定した名前を使えます。または、読み取り/書き込みリクエストを実行するプロキシのデフォルトエンドポイントの特別な名前 `default` にすることができます。プロキシエンドポイントの詳細については、「Amazon RDS Proxy エンドポイントの操作」を参照してください。

トピック

- プロキシでの接続の検証
- 一般的な の問題と解決策

プロキシでの接続の検証

次のコマンドを使用して、接続内のプロキシ、データベース、コンピューティングインスタンスなどのすべてのコンポーネントが相互に通信できることを確認できます。

describe-db-proxies コマンドを使用して、プロキシ自体を調べます。また、describe-db-proxy-target-groups コマンドを使用して、関連するターゲットグループを確認します。ターゲットの詳細が、プロキシに関連付ける Aurora クラスターと一致していることを確認します。以下のようなコマンドを使用します。

```
aws rds describe-db-proxies --db-proxy-name $DB_PROXY_NAME
aws rds describe-db-proxy-target-groups --db-proxy-name $DB_PROXY_NAME
```

プロキシが基になるデータベースに接続できることを確認するには、describe-db-proxy-targets コマンドを使用して、ターゲットグループで指定されたターゲットを調べます。以下のようなコマンドを使用します。

```
aws rds describe-db-proxy-targets --db-proxy-name $DB_PROXY_NAME
```

describe-db-proxy-targets コマンドの出力には、`TargetHealth` フィールドが含まれます。`State` 内のフィールド `Reason`、`Description`、および `TargetHealth` を調べて、プロキシが基になる DB インスタンスと通信できるかどうかを確認できます。

- `State` の値 `AVAILABLE` は、プロキシが DB インスタンスに接続できることを示します。
- `State` の値 `UNAVAILABLE` は、一時的または永続的な接続の問題を示します。この場合は、`Reason` および `Description` フィールドを調べます。例えば、`Reason` の値が `PENDING_PROXY_CAPACITY` の場合は、プロキシがスケーリングオペレーションを完了した

後で、接続を再試行します。Reason の値が UNREACHABLE、CONNECTION_FAILED、または AUTH_FAILURE の場合は、Description フィールドの説明が問題の診断に役立ちます。

- State フィールドでは、REGISTERING または AVAILABLE に変わるまでの短い間、値が UNAVAILABLE になる場合があります。

次の Netcat コマンド (nc) が成功した場合は、ログインしている EC2 インスタンスや他のシステムからプロキシエンドポイントにアクセスできます。このコマンドは、プロキシおよび関連付けられたデータベースと同じ VPC 内に存在していない場合、失敗を報告します。同じ VPC に存在していなくても、データベースに直接ログインできる場合があります。ただし、同じ VPC 内に存在していない限り、プロキシにはログインできません。

```
nc -zx MySQL_proxy_endpoint 3306

nc -zx PostgreSQL_proxy_endpoint 5432
```

次のコマンドを使用して、EC2 インスタンスに必要なプロパティがあることを確認できます。特に、EC2 インスタンスの VPC は、プロキシが接続する先の RDS DB インスタンス Aurora クラスターの VPC と同じである必要があります。

```
aws ec2 describe-instances --instance-ids your_ec2_instance_id
```

プロキシで使用されている Secrets Manager シークレットを確認します。

```
aws secretsmanager list-secrets
aws secretsmanager get-secret-value --secret-id your_secret_id
```

SecretString によって表示される get-secret-value フィールドが JSON 文字列としてエンコードされ、この文字列に username フィールドと password フィールドが含まれていることを確認します。次の例は、SecretString フィールドの形式を示しています。

```
{
  "ARN": "some_arn",
  "Name": "some_name",
  "VersionId": "some_version_id",
  "SecretString": '{"username":"some_username","password":"some_password"}',
  "VersionStages": [ "some_stage" ],
  "CreatedDate": some_timestamp
}
```

一般的な の問題と解決策

このセクションでは、RDS Proxy を使用する際の一般的な問題と考えられる解決策について説明します。

`aws rds describe-db-proxy-targets` CLI コマンドの実行後、`TargetHealth` の説明に `Proxy does not have any registered credentials` と記載されている場合は、以下を確認してください:

- ユーザーがプロキシにアクセスするための認証情報が登録されています。
- プロキシが使用する Secrets Managerシークレットにアクセスする IAM ロールは有効です。

DB プロキシの作成時や接続時に、次の RDS イベントが発生することがあります。

カテゴリ	RDS イベント ID	説明
失敗	RDS-EVENT-0243	サブネットに十分な IP アドレスがないため、RDS はプロキシの容量をプロビジョニングできませんでした。この問題を解決するには、サブネットの未使用の IP アドレスが最小限であることを確認してください。インスタンスクラスの推奨数を決定するには、「IP アドレス容量の計画」を参照してください。
失敗	RDS-EVENT-0275	RDS は DB プロキシ#への一部の接続をスロットリングしました。クライアントからプロキシへの同時接続リクエストの数が制限を超えました。

新しいプロキシの作成時やプロキシへの接続時に、次の問題が発生することがあります。

エラー	原因または回避策
`403: The security token included in the request is invalid`	新しい IAM ロールを作成せずに、既存の IAM ロールを選択します。

MySQL プロキシへの接続時に次の問題が発生することがあります。

エラー	原因または回避策
`ERROR 1040 (HY000): Connections rate limit exceeded (`*`limit_value`*` )`	クライアントからプロキシへの接続リクエストのレートが制限を超えました。
`ERROR 1040 (HY000): IAM authentication rate limit exceeded`	クライアントからプロキシへの IAM 認証による同時リクエストの数が制限を超えました。
`ERROR 1040 (HY000): Number simultaneous connections exceeded (`*`limit_value`*` )`	クライアントからプロキシへの同時接続リクエストの数が制限を超えました。
`ERROR 1045 (28000): Access denied for user`	プロキシで使用される Secrets Manager シークレットが既存のデータベースユーザーのユーザー名およびパスワードと一致しません。Secrets Manager シークレットの認証情報を更新します。または、データベース

エラー	原因または回避策
'DB_USER'@'%' (usi password: YES)	ユーザーが存在し、そのパスワードがシークレットのものと同じであることを確認します。
ERROR 1105 (HY000): Unknown error	不明なエラーが発生しました。
ERROR 1231 (42000): Variable ''charact er_set_cl ient'' can't be set to the value of *value*	character_set_client パラメータに設定した値が無効です。例えば、値 ucs2 は、MySQL サーバーをクラッシュさせる可能性があるため、有効ではありません。
ERROR 3159 (HY000): This RDS Proxy requires TLS connections.	プロキシで [Transport Layer Security が必要] 設定を有効にしましたが、MySQL クライアントで接続にパラメータ ssl-mode=DISABLED が含まれていました。次のいずれかを行います。 • プロキシの [Transport Layer Security が必要] 設定を無効にします。 • MySQL クライアントで ssl-mode=REQUIRED の最小設定を使用してデータベースに接続します。
ERROR 2026 (HY000): SSL connection error: Internal Server *Error*	プロキシへの TLS ハンドシェイクが失敗しました。次のような原因が考えられます。 • SSL は必須ですが、サーバーではサポートされていません。 • 内部サーバーエラーが発生しました。 • 不正なハンドシェイクが発生しました。

エラー	原因または回避策
`ERROR 9501 (HY000): Timed-out waiting to acquire database connection`	プロキシは、データベース接続の取得を待機中にタイムアウトしました。次のような原因が考えられます。 • 最大接続数に達したため、プロキシはデータベース接続を確立できません • データベースが使用不能であるため、プロキシはデータベース接続を確立できません。

PostgreSQL プロキシへの接続中に次の問題が発生することがあります。

エラー	原因	ソリューション
`IAM authentication is allowed only with SSL connections.`	ユーザーが、PostgreSQL クライアントで `sslmode=disable` を設定して IAM 認証を使用してデータベースに接続しようとしました。	ユーザーは、PostgreSQL クライアントで `sslmode=require` の最小設定を使用して、データベースに接続する必要があります。詳細については、PostgreSQL の SSL サポートに関するドキュメントを参照してください。
`This RDS Proxy requires TLS connections.`	ユーザーは [Transport Layer Security が必要] オプションを有効にしましたが、PostgreSQL クライアントで `sslmode=disable` を使用して接続しようとしました。	このエラーを修正するには、以下のいずれかを行います。 • プロキシの [Transport Layer Security が必要] オプションを無効にします。 • PostgreSQL クライアントで `sslmode=allow` の最小設定を使用してデータベースに接続します。
`IAM authentication failed for user` *`user_name`* `. Check the`	このエラーの原因としては、以下が考えられます。	このエラーを修正する方法は次のとおりです。

エラー	原因	ソリューション
`IAM token for this user and try again.`	• クライアントが不正な IAM ユーザー名を指定した。 • クライアントがユーザーの不正な IAM 認証トークンを指定した。 • クライアントが使用している IAM ポリシーに必要なアクセス許可がない。 • クライアントがユーザーの期限切れの IAM 認証トークンを指定した。	1. 指定した IAM ユーザーが存在することを確認します。 2. IAM 認証トークンが指定した IAM ユーザーに属することを確認します。 3. IAM ポリシーに RDS への適切なアクセス許可があることを確認します。 4. 使用した IAM 認証トークンが有効であることを確認します。
`This RDS proxy has no credentials for the role` *`role_name`* `. Check the credentials for this role and try again.`	このロールには Secrets Manager シークレットはありません。	このロールの Secrets Manager シークレットを追加します。詳細については、「AWS Identity and Access Management (IAM) ポリシーの設定」を参照してください。
`RDS supports only IAM, MD5, or SCRAM authentication.`	プロキシへの接続に使用されているデータベースクライアントが、プロキシで現在サポートされていない認証メカニズムを使用しています。	IAM 認証を使用していない場合は、MD5 または SCRAM パスワード認証を使用してください。
`A user name is missing from the connection startup packet. Provide a user name for this connection.`	プロキシへの接続に使用されているデータベースクライアントが、接続の確立を試みるときにユーザー名を送信していません。	選択した PostgreSQL クライアントを使用してプロキシへの接続を設定するときは、必ずユーザー名を定義してください。

エラー	原因	ソリューション
`Feature not supported: RDS Proxy supports only version 3.0 of the PostgreSQL messaging protocol.`	プロキシへの接続に使用される PostgreSQL クライアントは、3.0 より古いプロトコルを使用します。	3.0 メッセージングプロトコルをサポートする、より新しい PostgreSQL クライアントを使用します。PostgreSQL `psql` CLI を使用している場合は、バージョン 7.4 以降を使用します。
`Feature not supported: RDS Proxy currently doesn't support streaming replication mode.`	プロキシへの接続に使用されている PostgreSQL クライアントが、ストリーミングレプリケーションモードを使用しようとしています。このモードは、現在 RDS Proxy でサポートされていません。	接続に使用されている PostgreSQL クライアントでストリーミングレプリケーションモードをオフにします。
`Feature not supported: RDS Proxy currently doesn't support the option` *`option_name`* `.`	プロキシへの接続に使用されている PostgreSQL クライアントが、起動メッセージを通じて、RDS Proxy で現在サポートされていないオプションをリクエストしています。	接続に使用されている PostgreSQL クライアントで、上記のメッセージから、サポートされていないと表示されているオプションをオフにします。
`The IAM authentication failed because of too many competing requests.`	クライアントからプロキシへの IAM 認証による同時リクエストの数が制限を超えました。	PostgreSQL クライアントからの IAM 認証を使用した接続の確立速度を下げます。
`The maximum number of client connections to the proxy exceeded` *`number_value`* `.`	クライアントからプロキシへの同時接続リクエストの数が制限を超えました。	PostgreSQL クライアントからこの RDS Proxy へのアクティブな接続の数を減らします。
`Rate of connection to proxy exceeded` *`number_value`* `.`	クライアントからプロキシへの接続リクエストのレートが制限を超えました。	PostgreSQL クライアントからの接続の確立速度を下げます。

エラー	原因	ソリューション
`The password that was provided for the role role_name is wrong.`	このロールのパスワードが Secrets Manager シークレットと一致しません。	Secrets Manager でこのロールのシークレットをチェックして、パスワードが PostgreSQL クライアントで使用されているものと同じかどうかを確認します。
`The IAM authentication failed for the role role_name . Check the IAM token for this role and try again.`	IAM 認証に使用される IAM トークンに問題があります。	新しい認証トークンを生成し、新しい接続で使用します。
`IAM is allowed only with SSL connections.`	クライアントが IAM 認証を使用して接続しようとしましたが、SSL が有効になっていませんでした。	PostgreSQL クライアントで SSL を有効にします。
`Unknown error.`	不明なエラーが発生しました。	AWS サポートに連絡して、問題の調査を依頼してください。

エラー	原因	ソリューション
`Timed-out waiting to acquire database connection.`	プロキシは、データベース接続の取得を待機中にタイムアウトしました。次のような原因が考えられます。 • 最大接続数に達したため、プロキシはデータベース接続を確立できません。 • データベースが使用不能であるため、プロキシはデータベース接続を確立できません。	以下の解決策が対象となります。 • RDS DB インスタンス Aurora クラスターステータスのターゲットをチェックして、利用できないかどうかを確認します。 • 実行時間の長いトランザクションやクエリが実行されているかどうかを確認します。接続プールからのデータベース接続を長時間使用できます。
`Request returned an error: database_error .`	プロキシから確立されたデータベース接続がエラーを返しました。	解決策は、具体的なデータベースエラーによって異なります。1 つの例は、`Request returned an error: database "your-database-name" does not exist` です。これは、指定されたデータベース名がデータベースサーバーに存在しないことになります。または、データベース名として使用されているユーザー名 (データベース名が指定されていない場合) がサーバーに存在しないことになります。

RDS Proxy の AWS CloudFormation での使用

RDS Proxy は、AWS CloudFormation で使用できます。これにより、関連するリソースのグループを作成しやすくなります。このようなグループには、新しく作成された Aurora DB クラスターに接続できるプロキシを含めることができます。RDS Proxy の AWS CloudFormation でのサポートには、DBProxy および DBProxyTargetGroup の 2 つの新しいレジストリタイプが含まれます。

以下のリストは、RDS Proxy のサンプル AWS CloudFormation テンプレートを示しています。

```
Resources:
 DBProxy:
   Type: AWS::RDS::DBProxy
   Properties:
     DBProxyName: CanaryProxy
     EngineFamily: MYSQL
     RoleArn:
      Fn::ImportValue: SecretReaderRoleArn
     Auth:
       - {AuthScheme: SECRETS, SecretArn: !ImportValue ProxySecret, IAMAuth: DISABLED}
     VpcSubnetIds:
       Fn::Split: [",", "Fn::ImportValue": SubnetIds]

 ProxyTargetGroup:
   Type: AWS::RDS::DBProxyTargetGroup
   Properties:
     DBProxyName: CanaryProxy
     TargetGroupName: default
     DBInstanceIdentifiers:
       - Fn::ImportValue: DBInstanceName
   DependsOn: DBProxy
```

このサンプルのリソースの詳細については、「DBProxy」と「DBProxyTargetGroup」を参照してください。

AWS CloudFormation を使用して作成できるリソースの詳細については、「RDS リソースタイプのリファレンス」を参照してください。

Aurora グローバルデータベースで RDS Proxy を使用する

Aurora グローバルデータベースは、低レイテンシーでグローバルな読み取り機能と、リージョン全域の機能停止時のディザスタリカバリを備えた、複数の AWS リージョン にまたがる単一のデータ

ベースです。DB インスタンスは、単一の AWS リージョン ではなく、複数のリージョンおよび異なるアベイラビリティーゾーンに依存しているので、デプロイには耐障害性が組み込まれています。詳細については、「Amazon Aurora Global Database の使用」を参照してください。

Aurora グローバルデータベースの任意の DB クラスターで RDS Proxy を使用できます。これらの機能を一緒に使用する前に、次の情報を理解しておくことをお勧めします。

> **Important**
>
> DB クラスターが、書き込み転送が有効になっているグローバルデータベースの一部である場合は、プロキシの `MaxConnectionsPercent` 値を書き込み転送に割り当てられたクォータだけ減らします。書き込み転送クォータは DB クラスターパラメータ `aurora_fwd_writer_max_connections_pct` で設定されます。書き込み転送の詳細については、「Amazon Aurora Global Database の書き込み転送を使用する」を参照してください。

グローバル データベースを使用した RDS Proxy の制約事項

Aurora DB クラスターで書き込み転送が有効になっている場合、RDS Proxy は `aurora_replica_read_consistency` 変数の `SESSION` 値をサポートしません。この値を設定すると、予期しない動作が発生する可能性があります。

RDS Proxy エンドポイントとグローバルデータベースの連携について

RDS Proxy エンドポイントがグローバルデータベースとどのように連携するかを理解すると、これら両方の機能で、Aurora データベースを使用するアプリケーションをより適切に管理できます。

グローバルデータベースのプライマリクラスターを登録ターゲットとするプロキシの場合、プロキシエンドポイントは他の Aurora DB クラスターと同じように機能します。プロキシの読み取り/書き込みエンドポイントでは、すべてのリクエストをクラスターのライターインスタンスに送信します。プロキシの読み取り専用エンドポイントは、すべてのリクエストをリーダーインスタンスに送信します。接続中にリーダーが使用できなくなった場合、RDS プロキシは接続に関する後続のクエリを別のリーダーインスタンスにリダイレクトします。セカンダリクラスターを登録ターゲットとするプロキシの場合、プロキシの読み取り専用エンドポイントに送信されたリクエストはリーダーインスタンスにも送信されます。クラスターにはライターインスタンスがないため、読み取り/書き込みエンドポイントに送信されたリクエストはエラー「`The target group doesn't have any associated read/write instances`」で失敗します。

グローバルデータベーススイッチオーバーおよびフェイルオーバーオペレーションの両方で、プライマリ DB クラスターとセカンダリ DB クラスターのロールを切り替える必要があります。選択したセカンダリクラスターが新しいプライマリクラスターになると、そのリーダーインスタンスの 1 つがライターに昇格します。この DB インスタンスは、グローバルクラスターの新しいライターインスタンスになりました。アプリケーションの書き込みオペレーションを、新しいプライマリクラスターに関連付けられているプロキシの適切な読み取り/書き込みエンドポイントにリダイレクトします。このプロキシエンドポイントは、デフォルトのエンドポイントでも、カスタムの読み取り/書き込みエンドポイントでもかまいません。

RDS プロキシは、読み取り/書き込みエンドポイントを介してすべてのリクエストをキューに入れ、利用可能になり次第、新しいプライマリクラスターのライターインスタンスに送信します。これは、スイッチオーバーまたはフェイルオーバーオペレーションが完了したかどうかに関係なく実行されます。スイッチオーバーまたはフェイルオーバー中も、古いプライマリクラスターのプロキシのデフォルトエンドポイントは引き続き書き込みオペレーションを受け付けます。ただし、そのクラスターがセカンダリクラスターになると、すべての書き込みオペレーションは失敗します。特定のグローバルスイッチオーバーまたはフェイルオーバータスクをいつどのように実行するかについては、以下のトピックを参照してください。

- グローバルデータベースのスイッチオーバー — Amazon Aurora Global Database に対するスイッチオーバーの実行
- グローバルデータベースフェイルオーバー —予期しない停止からの Amazon Aurora Global Database の復旧

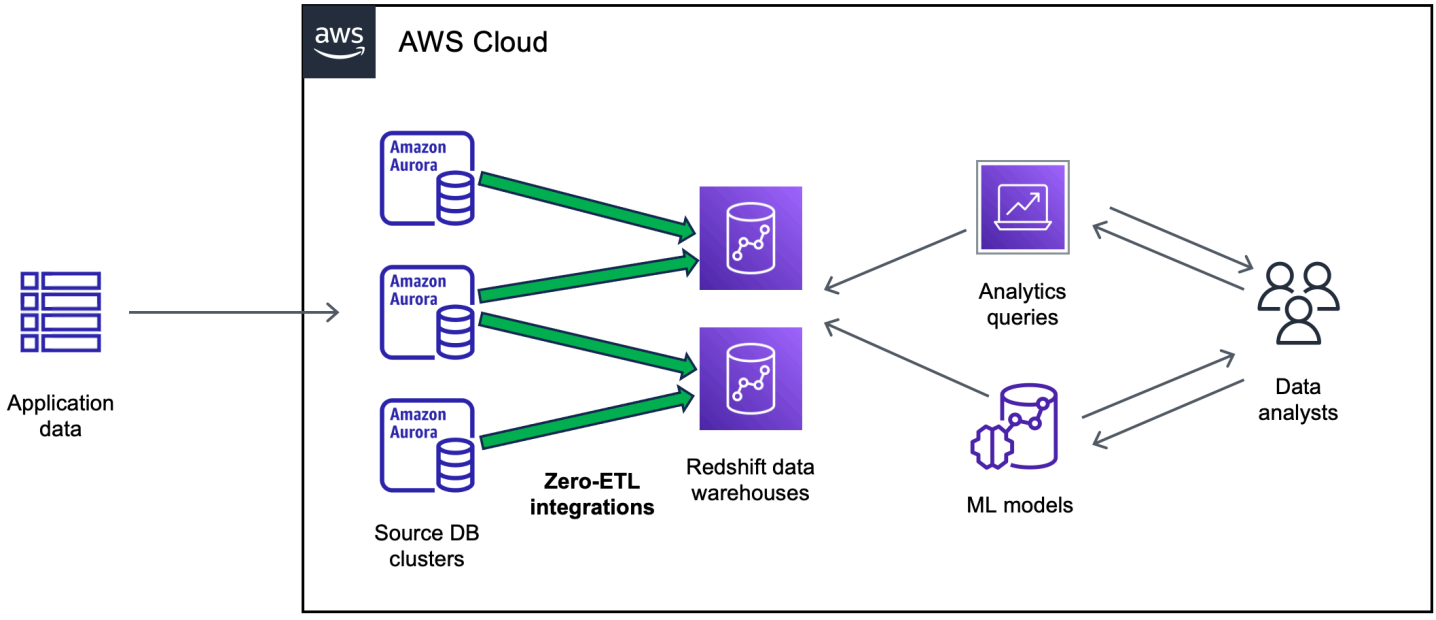
Amazon Redshift との Aurora ゼロ ETL 統合での作業

Amazon Redshift との Aurora ゼロ ETL 統合では、Aurora からの数ペタバイトのトランザクションデータに対して Amazon Redshift を使用して、ほぼリアルタイムの分析と機械学習 (ML) が可能です。これは、トランザクションデータを Aurora DB クラスターに書き込んだ後に Amazon Redshift で利用できるようにするためのフルマネージドソリューションです。抽出、変換、ロード (ETL) は、複数のソースからのデータを大規模な中央のデータウェアハウスにまとめるプロセスです。

ゼロ ETL 統合では、データ更新が書き込まれてからすぐに Aurora DB クラスターのデータが Amazon Redshift でほぼリアルタイムで利用できます。データが Amazon Redshift に格納されると、機械学習、マテリアライズドビュー、データ共有、複数のデータストアやデータレイクへのフェデレーションアクセス、Amazon SageMaker、Amazon QuickSight、その他の AWS のサービス との統合といった Amazon Redshift の組み込み機能を使用して、分析、ML、AI のワークロードを強化できます。

ゼロ ETL 統合を作成するには、Aurora DB クラスターをソースとして指定し、Amazon Redshift データウェアハウスをターゲットとして指定します。統合では、ソースデータベースからターゲットデータウェアハウスにデータがレプリケートされます。

次の図は、この機能を示しています。



統合は、データパイプラインの正常性をモニタリングし、可能な場合は問題から回復します。複数のの Aurora DB クラスターから単一の Amazon Redshift 名前空間に統合を作成できるため、複数のアプリケーションにわたってインサイトを引き出すことができます。

ゼロ ETL 統合の料金の詳細については、「Amazon Aurora の料金」および「Amazon Redshift の料金」を参照してください。

トピック

利点

Amazon Redshift との Aurora ゼロ ETL 統合には、主に次のような利点があります。

- 複数のデータソースから総合的なインサイトを引き出すのに役立ちます。
- 抽出、変換、ロード (ETL) 操作を実行する複雑なデータパイプラインを構築して管理する必要がなくなります。ゼロ ETL 統合は、パイプラインのプロビジョニングと管理を顧客に代わって行うことで、パイプラインの構築と管理に伴う課題を排除します。
- 運用上の負担とコストを削減し、アプリケーションの改善に集中できます。
- Amazon Redshift の分析機能と ML 機能を活用して、トランザクションデータやその他のデータからインサイトを引き出し、重要で時間的制約のあるイベントに効果的に対応できます。

主要なコンセプト

ゼロ ETL 統合を始める際には、以下の概念を検討してください。

Integration

Aurora DB クラスターから Amazon Redshift データウェアハウスにトランザクションデータとスキーマを自動的に複製する、フルマネージドデータパイプライン。

ソース DB クラスター

データがレプリケートされる Aurora DB クラスター。Aurora MySQL では、プロビジョニングされた DB インスタンスまたは Aurora Serverless v2 DB インスタンスをソースとして使用する DB クラスターを指定できます。Aurora PostgreSQL プレビューでは、プロビジョニングされた DB インスタンスを使用するクラスターのみを指定できます。

ターゲットデータウェアハウス

データがレプリケートされる Amazon Redshift データウェアハウス。データウェアハウスには、プロビジョニングされたクラスターデータウェアハウスとサーバーレスデータウェアハウスの 2 種類があります。プロビジョニングされたクラスターデータウェアハウスは、ノードと呼ばれるコンピューティングリソースのコレクションであり、クラスターと呼ばれるグループに編成されています。サーバーレスデータウェアハウスは、コンピューティングリソースを格納するワークグループと、データベースオブジェクトとユーザーを収容する名前空間で構成されています。どちらのデータウェアハウスも Amazon Redshift エンジンを実行し、1 つ以上のデータベースを含んでいます。

複数のソースの DB クラスターを同じターゲットに書き込むことができます。

詳細については、「Amazon Redshift デベロッパーガイド」の「データウェアハウスのシステムアーキテクチャ」を参照してください。

プレビューの制限事項制限事項

Amazon Redshift との Aurora ゼロ ETL 統合には、以下の制限が適用されます。

トピック

- 一般的な制限事項
- Aurora MySQL の制限事項
- Aurora PostgreSQL プレビューの制限事項
- Amazon Redshift の制限事項

一般的な制限事項

- ソース DB クラスターは、ターゲット Amazon Redshift データウェアハウスと同じリージョンにある必要があります。
- クラスターに既存の統合がある場合、DB クラスターやそのインスタンスの名前を変更することはできません。
- 既存の統合がある DB クラスターは削除できません。まず、関連する統合をすべて削除する必要があります。
- ソース Aurora DB クラスターを停止すると、クラスターを再開するまで、最後のいくつかのトランザクションがターゲットデータウェアハウスにレプリケートされない場合があります。
- クラスターがブルー/グリーンデプロイのソースである場合、ブルー環境とグリーン環境の切り替え中に既存のゼロ ETL 統合を置くことはできません。最初に統合を削除してから切り替えて、再作成する必要があります。
- DB クラスターが統合のソースになるには、1 つ以上の DB インスタンスが含まれている必要があります。
- ソースクラスターが Aurora グローバルデータベースのプライマリ DB クラスターであり、セカンダリクラスターの 1 つにフェールオーバーすると、統合は非アクティブになります。統合を手動で削除して再作成する必要があります。
- 統合を作成中のソースデータベースで別の統合を作成することはできません。
- 初めて統合を作成するとき、またはテーブルを再同期するとき、ソースデータベースのサイズによっては、ソースからターゲットへのデータシードに 20 ~ 25 分以上かかる場合があります。この遅延により、レプリカラグが長くなる可能性があります。
- 一部のデータ型はサポートされていません。詳細については、「the section called “データ型の相違点”」を参照してください。
- 定義済みのテーブル更新を伴う外部キー参照はサポートされていません。具体的には、`ON DELETE` および `ON UPDATE` ルールは、`CASCADE`、`SET NULL`、および `SET DEFAULT` アクションではサポートされていません。別のテーブルへの参照を含むテーブルを作成または更新しようとすると、テーブルは失敗状態になります。
- `ALTER TABLE` パーティション操作では、Aurora から Amazon Redshift にデータをリロードするためにテーブルが再同期されます。再同期中は、テーブルをクエリすることはできません。詳細については、「the section called “1 つ以上の Amazon Redshift テーブルを再同期する必要がある”」を参照してください。
- XA トランザクションはサポートされていません。

- オブジェクト識別子 (データベース名、テーブル名、列名などを含む) には、英数字、数字、$、_ (アンダースコア) のみを使用できます。

Aurora MySQL の制限事項

- ソース DB クラスターは Aurora MySQL バージョン 3.05 (MySQL 8.0.32 と互換) 以降を実行している必要があります。
- ゼロ ETL 統合では、MySQL バイナリロギング (binlog) を利用して継続的なデータ変更をキャプチャします。バイナリログベースのデータフィルタリングは使用しないでください。ソースとターゲットのデータベース間でデータの不整合が生じる可能性があります。
- Aurora MySQL システムテーブル、一時テーブル、ビューは Amazon Redshift にレプリケートされません。
- ゼロ ETL 統合は、InnoDB ストレージエンジンを使用するように設定されたデータベースでのみサポートされています。

Aurora PostgreSQL プレビューの制限事項

> **Important**
>
> Aurora PostgreSQL の Amazon Redshift とのゼロ ETL 統合はプレビューリリースです。ドキュメントと機能はどちらも変更されることがあります。この機能については、テスト環境のみで使用でき、本番環境では使用できません。プレビューの利用規約については、「AWS のサービス条件」の「ベータ版とプレビュー」を参照してください。

- ソース DB クラスターは Aurora PostgreSQL を実行している必要があります (PostgreSQL 15.4 およびゼロ ETL サポートと互換性があります) 。
- Aurora PostgreSQL のゼロ ETL 統合を作成および管理できるのは、米国東部 (オハイオ) (us-east-2) AWS リージョン の Amazon RDS データベースプレビュー環境のみです。プレビュー環境を使用して、PostgreSQL データベースエンジンソフトウェアのベータ、リリース候補、および初期の本番バージョンをテストできます。
- Aurora PostgreSQL の統合の作成と管理は、AWS Management Console を使用してのみ行えます。AWS Command Line Interface (2)、Amazon RDS API、または任意の AWS SDK を使用することはできません。

- ソース DB クラスターを作成するときは、選択するパラメータグループに、必要な DB クラスターパラメータ値が既に設定されている必要があります。後で新しいパラメータグループを作成してクラスターに関連付けることはできません。必要なパラメータのリストについては、the section called “ステップ 1: カスタム DB クラスターのパラメータグループを作成する” を参照してください。
- 統合は、作成した後で変更することはできません。特定の設定を変更する必要がある場合は、統合を削除して再作成する必要があります。
- 現在、統合のソースである Aurora PostgreSQL DB クラスターは、論理レプリケーションデータのガベージコレクションを実行しません。
- ソース Aurora PostgreSQL DB クラスター内で作成されるすべてのデータベースは、UTF-8 エンコーディングを使用する必要があります。
- 列名には、カンマ (,)、セミコロン (;)、括弧 ()、中括弧 { }、改行 (\n)、タブ (\t)、等号 (=)、スペースの文字を含めることはできません。
- Aurora PostgreSQL とのゼロ ETL 統合では、以下はサポートされていません。
  - Aurora Serverless v2 DB インスタンス。ソース DB クラスターは、プロビジョニングされた DB インスタンスを使用する必要があります。
  - カスタムデータ型、または拡張機能によって作成されたデータ型。
  - ソース DB クラスターのサブトランザクション。
  - ソース DB クラスター内のスキーマまたはデータベースの名前の変更。
  - DB クラスタースナップショットからの復元、または Aurora クローンを使用したソース DB クラスターの作成。既存のデータをプレビュークラスターに取り込む場合は、`pg_dump` または `pg_restore` ユーティリティを使用する必要があります。
  - ソース DB クラスターのライターインスタンスでの論理レプリケーションスロットの作成。
  - オーバーサイズ属性ストレージ技術 (TOAST) を必要とする大きなフィールド値。
  - `ALTER TABLE` パーティション操作。これらの操作により、テーブルが再同期され、最終的に `Failed` 状態になる可能性があります。テーブルに障害が発生した場合は、テーブルを削除して再作成する必要があります。

Amazon Redshift の制限事項

ゼロ ETL 統合に関連する Amazon Redshift の制限の一覧については、「Amazon Redshift 管理ガイド」の「考慮事項」を参照してください。

クォータ

お客様のアカウントには、Amazon Redshift との Aurora ゼロ ETL 統合に関連する以下のクォータが設定されています。特に指定がない限り、各クォータはリージョンあたりです。

名前	デフォルト	説明
統合	100	AWS アカウント 内の統合の総数。
ターゲットデータウェアハウスあたりの統合数	50	1 つのターゲット Amazon Redshift データウェアハウスにデータを送信する統合の数。
ソースクラスターごとの統合	Aurora MySQL では 5、Aurora PostgreSQL では 1	単一のソース DB クラスターからデータを送信する統合の数。

さらに、Amazon Redshift は、各 DB インスタンスまたはクラスターノードで使用できるテーブルの数に一定の制限を設けています。詳細については、「Amazon Redshift 管理ガイド」の「Amazon Redshift のクォータと制限」を参照してください。

サポートされるリージョン

Amazon Redshift との Aurora ゼロ ETL 統合は、AWS リージョン のサブセットで利用できます。サポートされているリージョンのリストについては「the section called “ゼロ ETL 統合 ”」を参照してください。

Amazon Redshift との Aurora ゼロ ETL 統合の開始方法

Amazon Redshift とのゼロ ETL 統合を作成する前に、必要なパラメータとアクセス許可で Aurora DB クラスターと Amazon Redshift データウェアハウスを設定します。セットアップ時には、以下の手順を完了します。

1. カスタム DB クラスターのパラメータグループを作成します。

2. ソース DB クラスターを作成します。
3. ターゲット Amazon Redshift データウェアハウスを作成する

これらのタスクが完了したら、the section called “ゼロ ETL 統合の作成” に進みます。

AWS SDK を使用して、セットアッププロセスを自動化できます。詳細については、「the section called “AWS SDK を使用して統合をセットアップする (Aurora MySQL のみ) ”」を参照してください。

ステップ 1: カスタム DB クラスターのパラメータグループを作成する

Amazon Redshift との Aurora ゼロ ETL 統合には、レプリケーションを制御する DB クラスターパラメータに特定の値が必要です。具体的には、Aurora MySQL には拡張バイナリログ (aurora_enhanced_binlog) が必要であり、Aurora PostgreSQL には拡張論理レプリケーション (aurora.enhanced_logical_replication) が必要です。

バイナリロギングまたは論理レプリケーションを設定するには、まずカスタム DB クラスターパラメータグループを作成し、それをソース DB クラスターに関連付ける必要があります。

ソース DB エンジンに応じて、以下の設定でカスタム DB クラスターパラメータグループを作成します。カスタムパラメータグループを作成するには、「the section called “DB クラスターパラメータグループを使用する”」を参照してください。

Aurora MySQL (aurora-mysql8.0 ファミリー):

- aurora_enhanced_binlog=1
- binlog_backup=0
- binlog_format=ROW
- binlog_replication_globaldb=0
- binlog_row_image=full
- binlog_row_metadata=full

さらに、binlog_transaction_compressionパラメータが ON に設定されていないこと、および binlog_row_value_options パラメータが PARTIAL_JSON に設定されていないことを確認してください。

Aurora MySQL 拡張バイナリログの詳細については、「the section called “拡張バイナリログ記録の設定”」を参照してください。

Aurora PostgreSQL (aurora-postgresql15 ファミリー):

> **Note**
>
> Aurora PostgreSQL DB クラスターの場合、米国東部 (オハイオ) (us-east-2) AWS リージョン の Amazon RDS データベースプレビュー環境 内にカスタムパラメータグループを作成する必要があります。

- `rds.logical_replication=1`
- `aurora.enhanced_logical_replication=1`
- `aurora.logical_replication_backup=0`
- `aurora.logical_replication_globaldb=0`

拡張論理レプリケーション (`aurora.enhanced_logical_replication`) を有効にすると、`REPLICA IDENTITY` パラメータが自動的に `FULL` に設定されます。つまり、すべての列の値が先書きログ (WAL) に書き込まれます。これにより、ソース DB クラスターの IOPS が増加します。

ステップ 2: ソース DB クラスターを選択または作成する

カスタム DB クラスターのパラメータグループを作成したら、Aurora MySQL または Aurora PostgreSQL DB クラスターを選択または作成します。この のクラスターは、Amazon Redshift へのデータレプリケーションのソースになります。

クラスターは、Aurora MySQL バージョン 3.05 以降または Aurora PostgreSQL (PostgreSQL 15.4 およびゼロ ETL サポートと互換) を実行している必要があります。の DB クラスターの作成手順については、「the section called “DB クラスターの作成”」を参照してください。

> **Note**
>
> 米国東部 (オハイオ) (us-east-2) AWS リージョン の Amazon RDS データベースプレビュー環境 内に Aurora PostgreSQL DB クラスターを作成する必要があります。

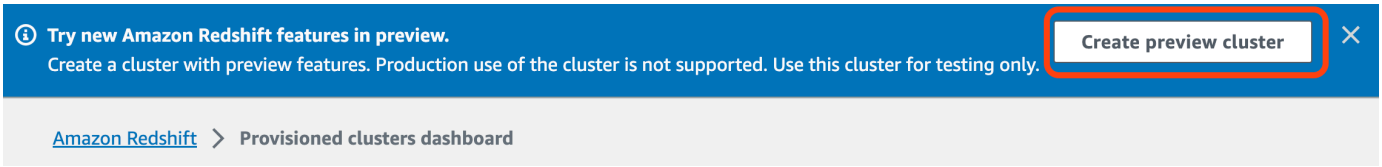
[追加設定] で、デフォルトの DB クラスターパラメータグループを、前のステップで作成したカスタムパラメータグループに変更します。

> **Note**
>
> Aurora MySQL の場合、DB クラスターの作成後にパラメータグループをクラスターに関連付ける場合は、ゼロ ETL 統合を作成する前にクラスター内のプライマリ DB インスタンスを再起動して変更を適用する必要があります。手順については、the section called “Aurora DB クラスターまたはインスタンスの再起動” を参照してください。
> Amazon Redshift との Aurora PostgreSQL ゼロ ETL 統合のプレビューリリースでは、クラスターの作成中にクラスターをカスタム DB クラスターパラメータグループに関連付ける必要があります。ソース DB クラスターの作成後は、このアクションを実行できません。作成していない場合は、クラスターを削除して再作成する必要があります。

ステップ 3: ターゲット Amazon Redshift データウェアハウスを作成する

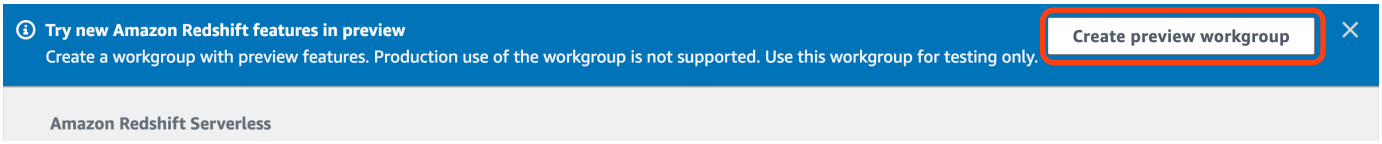
ソース DB クラスターを作成した後、Amazon Redshift でターゲットのデータウェアハウスを作成して設定する必要があります。データウェアハウスは、以下の要件を満たしている必要があります。

- プレビューで作成 (Aurora PostgreSQL ソースのみ)。Aurora MySQL ソースの場合、本番クラスターとワークグループを作成する必要があります。
  - プロビジョニングされたクラスターをプレビューで作成するには、プロビジョニングされたクラスターダッシュボードのバナーから [プレビュークラスターの作成] を選択します。詳細については、「プレビュークラスターの作成」を参照してください。



    クラスターを作成するときは、プレビュートラックを `preview_2023` に設定します。

  - Redshift Serverless ワークグループをプレビューで作成するには、Serverless ダッシュボードのバナーから [プレビューワークグループの作成] を選択します。詳細については、「プレビューワークグループの作成」を参照してください。



- RA3 ノードタイプ (`ra3.xlplus`、`ra3.4xlarge`、`ra3.16xlarge` のいずれか)、または Redshift Serverless を使用している。

- 暗号化されている (プロビジョニングされたクラスターを使用している場合) 詳細については、「Amazon Redshift データベースの暗号化」を参照してください。

データウェアハウスを作成する手順については、プロビジョニングされたクラスター用の「クラスターの作成」またはRedshift Serverless 用の「名前空間を使用したワークグループの作成」を参照してください。

データウェアハウスで大文字と小文字の区別を有効にします。

統合を正常に行うには、データウェアハウスで大文字と小文字を区別するパラメータ (enable_case_sensitive_identifier) を有効にする必要があります。デフォルトでは、プロビジョニング済みクラスターと Redshift Serverless ワークグループの大文字と小文字の区別は無効になっています。

大文字と小文字の区別を有効にするには、データウェアハウスのタイプに応じて以下の手順を実行します。

- プロビジョニングされたクラスター — プロビジョニングされたクラスターで大文字と小文字の区別を有効にするには、enable_case_sensitive_identifier パラメータを有効にしたカスタムパラメータグループを作成します。次に、そのパラメータグループとクラスターを関連付けます。手順については、「コンソールを使用したパラメータグループの管理」または「AWS CLI を使用したパラメータ値の設定」を参照してください。

  > **Note**
  >
  > クラスターにパラメータグループを関連付けたら、クラスターを再起動することを忘れないでください。

- Serverless ワークグループ — Redshift Serverless ワークグループで大文字と小文字の区別を有効にするには、AWS CLI を使用する必要があります。Amazon Redshift コンソールは現在、Redshift Serverless パラメータ値の変更をサポートしていません。次の update-workgroup リクエストを送信します。

  ```
  aws redshift-serverless update-workgroup \
    --workgroup-name target-workgroup \
    --config-parameters
   parameterKey=enable_case_sensitive_identifier,parameterValue=true
  ```

  パラメータ値を変更した後、ワークグループを再起動する必要はありません。

データウェアハウスの認証を設定します。

データウェアハウスを作成したら、ソース Aurora DB クラスターを承認済みの統合ソースとして設定する必要があります。手順については、「Amazon Redshift データウェアハウスの認証を設定する」を参照してください。

AWS SDK を使用して統合をセットアップする (Aurora MySQL のみ)

各リソースを手動でセットアップするのではなく、次の Python スクリプトを実行して、必要なリソースを自動的にセットアップできます。このコード例では AWS SDK for Python (Boto3) を使用してソース Aurora MySQL DB クラスターとターゲット Amazon Redshift データウェアハウスを作成し、それぞれに必要なパラメータ値を指定します。次に、クラスターが使用可能になるまで待ってから、クラスター間にゼロ ETL 統合を作成します。設定する必要があるリソースに応じて、さまざまな関数をコメントアウトできます。

必要な従属関係をインストールには、次のコマンドを実行します。

```
pip install boto3
pip install time
```

スクリプト内で、オプションでソース、ターゲット、パラメータグループの名前を変更します。最後の関数は、リソースのセットアップ後に `my-integration` という名前の統合を作成します。

Python コード例

```
import boto3
import time

# Build the client using the default credential configuration.
# You can use the CLI and run 'aws configure' to set access key, secret
# key, and default Region.

rds = boto3.client('rds')
redshift = boto3.client('redshift')
sts = boto3.client('sts')

source_cluster_name = 'my-source-cluster' # A name for the source cluster
source_param_group_name = 'my-source-param-group' # A name for the source parameter
 group
target_cluster_name = 'my-target-cluster' # A name for the target cluster
target_param_group_name = 'my-target-param-group' # A name for the target parameter
 group
```

```
def create_source_cluster(*args):
    """Creates a source Aurora MySQL DB cluster"""

    response = rds.create_db_cluster_parameter_group(
        DBClusterParameterGroupName=source_param_group_name,
        DBParameterGroupFamily='aurora-mysql8.0',
        Description='For Aurora MySQL zero-ETL integrations'
    )
    print('Created source parameter group: ' + response['DBClusterParameterGroup']
['DBClusterParameterGroupName'])

    response = rds.modify_db_cluster_parameter_group(
        DBClusterParameterGroupName=source_param_group_name,
        Parameters=[
            {
                'ParameterName': 'aurora_enhanced_binlog',
                'ParameterValue': '1',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_backup',
                'ParameterValue': '0',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_format',
                'ParameterValue': 'ROW',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_replication_globaldb',
                'ParameterValue': '0',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_row_image',
                'ParameterValue': 'full',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_row_metadata',
                'ParameterValue': 'full',
                'ApplyMethod': 'pending-reboot'
```

```
            }
        ]
    )
    print('Modified source parameter group: ' +
 response['DBClusterParameterGroupName'])

    response = rds.create_db_cluster(
        DBClusterIdentifier=source_cluster_name,
        DBClusterParameterGroupName=source_param_group_name,
        Engine='aurora-mysql',
        EngineVersion='8.0.mysql_aurora.3.05.2',
        DatabaseName='myauroradb',
        MasterUsername='username',
        MasterUserPassword='Password01**'
    )
    print('Creating source cluster: ' + response['DBCluster']['DBClusterIdentifier'])
    source_arn = (response['DBCluster']['DBClusterArn'])
    create_target_cluster(target_cluster_name, source_arn, target_param_group_name)

    response = rds.create_db_instance(
        DBInstanceClass='db.r6g.2xlarge',
        DBClusterIdentifier=source_cluster_name,
        DBInstanceIdentifier=source_cluster_name + '-instance',
        Engine='aurora-mysql'
    )
    return(response)

def create_target_cluster(target_cluster_name, source_arn, target_param_group_name):
    """Creates a target Redshift cluster"""

    response = redshift.create_cluster_parameter_group(
        ParameterGroupName=target_param_group_name,
        ParameterGroupFamily='redshift-1.0',
        Description='For Aurora MySQL zero-ETL integrations'
    )
    print('Created target parameter group: ' + response['ClusterParameterGroup']
['ParameterGroupName'])

    response = redshift.modify_cluster_parameter_group(
        ParameterGroupName=target_param_group_name,
        Parameters=[
            {
                'ParameterName': 'enable_case_sensitive_identifier',
                'ParameterValue': 'true'
```

```
            }
        ]
    )
    print('Modified target parameter group: ' + response['ParameterGroupName'])

    response = redshift.create_cluster(
        ClusterIdentifier=target_cluster_name,
        NodeType='ra3.4xlarge',
        NumberOfNodes=2,
        Encrypted=True,
        MasterUsername='username',
        MasterUserPassword='Password01**',
        ClusterParameterGroupName=target_param_group_name
    )
    print('Creating target cluster: ' + response['Cluster']['ClusterIdentifier'])

    # Retrieve the target cluster ARN
    response = redshift.describe_clusters(
        ClusterIdentifier=target_cluster_name
    )
    target_arn = response['Clusters'][0]['ClusterNamespaceArn']

    # Retrieve the current user's account ID
    response = sts.get_caller_identity()
    account_id = response['Account']

    # Create a resource policy specifying cluster ARN and account ID
    response = redshift.put_resource_policy(
        ResourceArn=target_arn,
        Policy='''
        {
            \"Version\":\"2012-10-17\",
            \"Statement\":[
                {\"Effect\":\"Allow\",
                \"Principal\":{
                    \"Service\":\"redshift.amazonaws.com\"
                },
                \"Action\":[\"redshift:AuthorizeInboundIntegration\"],
                \"Condition\":{
                    \"StringEquals\":{
                        \"aws:SourceArn\":\"%s\"}
                    }
                },
                {\"Effect\":\"Allow\",
```

```
                \"Principal\":{
                    \"AWS\":\"arn:aws:iam::%s:root\"},
                \"Action\":\"redshift:CreateInboundIntegration\"}
            ]
        }
        ''' % (source_arn, account_id)
    )
    return(response)

def wait_for_cluster_availability(*args):
    """Waits for both clusters to be available"""

    print('Waiting for clusters to be available...')

    response = rds.describe_db_clusters(
        DBClusterIdentifier=source_cluster_name,
    )
    source_status = response['DBClusters'][0]['Status']
    source_arn = response['DBClusters'][0]['DBClusterArn']

    response = rds.describe_db_instances(
        DBInstanceIdentifier=source_cluster_name + '-instance',
    )
    source_instance_status = response['DBInstances'][0]['DBInstanceStatus']

    response = redshift.describe_clusters(
        ClusterIdentifier=target_cluster_name,
    )
    target_status = response['Clusters'][0]['ClusterStatus']
    target_arn = response['Clusters'][0]['ClusterNamespaceArn']

    # Every 60 seconds, check whether the clusters are available.
    if source_status != 'available' or target_status != 'available' or
 source_instance_status != 'available':
        time.sleep(60)
        response = wait_for_cluster_availability(
            source_cluster_name, target_cluster_name)
    else:
        print('Clusters available. Ready to create zero-ETL integration.')
        create_integration(source_arn, target_arn)
        return

def create_integration(source_arn, target_arn):
    """Creates a zero-ETL integration using the source and target clusters"""
```

```
    response = rds.create_integration(
        SourceArn=source_arn,
        TargetArn=target_arn,
        IntegrationName='my-integration'
    )
    print('Creating integration: ' + response['IntegrationName'])

def main():
    """main function"""
    create_source_cluster(source_cluster_name, source_param_group_name)
    wait_for_cluster_availability(source_cluster_name, target_cluster_name)

if __name__ == "__main__":
    main()
```

次のステップ

ソース Aurora DB クラスターと Amazon Redshift ターゲットデータウェアハウスを作成したので、ゼロ ETL 統合を作成してデータのレプリケーションを開始できます。手順については、the section called “ゼロ ETL 統合の作成” を参照してください。

Amazon Redshift との Amazon ゼロ ETL 統合の作成

Aurora ゼロ ETL 統合を作成するには、ソース Aurora DB クラスターとターゲットの Amazon Redshift データウェアハウスを指定します。暗号化設定をカスタマイズし、タグを追加することもできます。Aurora はソース DB クラスターとそのターゲットの間の統合を作成します。統合がアクティブになると、ソース DB クラスターに挿入したデータはすべて、設定された Amazon Redshift ターゲットにレプリケートされます。

トピック

前提条件

ゼロ ETL 統合を作成する前に、ソースの DB クラスターとターゲットの Amazon Redshift データウェアハウスを作成する必要があります。また、DB クラスターを承認済みの統合ソースとして追加することによって、データウェアハウスへのレプリケーションを許可する必要があります。

これらの各手順の実行方法については、「the section called “ゼロ ETL 統合の開始方法”」を参照してください。

必要なアクセス許可

ゼロ ETL 統合を作成するには、特定の IAM アクセス権限が必要です。少なくとも、次のアクションを実行するためのアクセス権限が必要です。

- ソースの Aurora DB クラスターのゼロ ETL 統合を作成します。
- すべてのゼロ ETL 統合を表示および削除します。
- ターゲットデータウェアハウスへのインバウンド統合を作成します。同じアカウントが Amazon Redshift データウェアハウスを所有していて、このアカウントがそのデータウェアハウスの承認済みプリンシパルである場合は、このアクセス許可が不要となります。承認済みプリンシパルの追加については、「Amazon Redshift データウェアハウスの承認の設定」を参照してください。

以下のサンプルポリシーは、インテグレーションの作成と管理に必要な最小特権を示しています。ユーザーまたはロールが AdministratorAccess マネージドポリシーなど、より広範なアクセス許可を持つ場合、これらの正確なアクセス許可を必要としない場合があります。

> **Note**
>
> Redshift Amazon リソースネーム (ARN) の形式は次のとおりです。サーバーレス名前空間 UUID の前にコロン (:) ではなくフォワードスラッシュ (/) を使用していることに注意してください。
>
> - プロビジョニング済みクラスタ — `arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid`
> - サーバーレス - `arn:aws:redshift-serverless:{region}:{account-id}:namespace/namespace-uuid`

ポリシーの例

> **Important**
>
> Aurora PostgreSQL プレビューでは、Amazon RDS データベースプレビュー環境内のすべての ARN とアクションは、サービス名前空間への -preview が追加されました。例えば、rds-*preview*:CreateIntegration と arn:aws:rds-*preview*:… です。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "rds:CreateIntegration"
        ],
        "Resource": [
            "arn:aws:rds:{region}:{account-id}:cluster:source-db",
            "arn:aws:rds:{region}:{account-id}:integration:*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "rds:DescribeIntegrations"
        ],
        "Resource": ["*"]
    },
    {
        "Effect": "Allow",
        "Action": [
            "rds:DeleteIntegration",
            "rds:ModifyIntegration"
        ],
        "Resource": [
            "arn:aws:rds:{region}:{account-id}:integration:*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "redshift:CreateInboundIntegration"
```

```
        ],
        "Resource": [
            "arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid"
        ]
    }]
}
```

別のアカウントでターゲットデータウェアハウスを選択する

別の AWS アカウント にあるターゲット Amazon Redshift データウェアハウスを指定する場合は、現在のアカウントのユーザーがターゲットアカウントのリソースにアクセスするのを許可するロールを作成する必要があります。詳細については、「所有している別の AWS アカウント の IAM ユーザーにアクセス権を付与する」を参照してください。

ロールには以下のアクセス許可が必要です。これにより、ユーザーは使用可能な Amazon Redshift のプロビジョニング済みクラスターとターゲットアカウントの Redshift Serverless 名前空間を表示できます。

必要なアクセス許可ポリシー

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "redshift:DescribeClusters",
            "redshift-serverless:ListNamespaces"
         ],
         "Resource":[
            "*"
         ]
      }
   ]
}
```

ロールには、ターゲットアカウント ID を指定する次の信頼ポリシーが必要です。

```
{
   "Version":"2012-10-17",
   "Statement":[
```

```
      {
         "Effect":"Allow",
         "Principal":{
            "AWS": "arn:aws:iam::{external-account-id}:root"
         },
         "Action":"sts:AssumeRole"
      }
   ]
}
```

ロールを作成する手順については、「カスタム信頼ポリシーを使用したロールの作成」を参照してください。

ゼロ ETL 統合の作成

AWS Management Console、AWS CLI、または RDS API を使用して、Aurora MySQL ゼロ ETL 統合を作成できます。Aurora PostgreSQL 統合を作成するには、AWS Management Console を使用する必要があります。

RDS コンソール

ゼロ ETL 統合を作成するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。

   統合のソースとして Aurora PostgreSQL DB クラスターを使用する場合、https://us-east-2.console.aws.amazon.com/rds-preview/home?region=us-east-2#databases で Amazon RDS データベースプレビュー環境にサインインする必要があります。
2. 左側のナビゲーションペインから、[ゼロ ETL 統合] を選択します。
3. [ゼロ ETL 統合の作成] を選択します。
4. [統合 ID] に、統合の名前を入力します。名前には最大 63 文字の英数字を使用でき、ハイフンを含めることができます。
5. [Next] を選択します。
6. [ソース] で、データの送信元となる Aurora DB クラスターを選択します。クラスターは、Aurora MySQL バージョン 3.05 以降または Aurora PostgreSQL (PostgreSQL 15.4 およびゼロ ETL サポートと互換) を実行している必要があります。

> **Note**
>
> MySQL ソースでは、DB クラスターパラメータが正しく設定されていないと、RDS から通知されます。このメッセージを受け取った場合は、[Fix it for me] を選択するか、手動で設定することができます。手動で修正する手順については、「the section called “ステップ 1: カスタム DB クラスターのパラメータグループを作成する”」を参照してください。
> DB クラスターパラメータを変更するには再起動が必要です。統合を作成する前に、再起動が完了し、新しいパラメータ値ののクラスターへの適用が正常に完了している必要があります。

7. Aurora PostgreSQL ソースクラスターを選択した場合、[名前付きデータベース] で、統合のソースとして使用する名前付きデータベースを指定します。PostgreSQL リソースモデルでは、1 つの DB クラスター内に複数のデータベースを作成できますが、ゼロ ETL 統合ごとに使用できるデータベースは 1 つだけです。

   名前付きデータベースは `template1` から作成する必要があります。詳細については、PostgreSQL ドキュメントの「テンプレートデータベース」を参照してください。

8. (オプション) Aurora MySQL ソース DB クラスターを選択した場合は、[データフィルタリングオプションのカスタマイズ] を選択して、統合にデータフィルターを追加します。データフィルターを使用して、ターゲットデータウェアハウスへのレプリケーションの範囲を定義できます。詳細については、「the section called “ゼロ ETL 統合でのデータフィルタリング”」を参照してください。

9. ソース DB クラスターが正常に設定されたら、[次へ] を選択します。

10. [ターゲット] で、以下を実行します。

    1. (オプション) Amazon Redshift ターゲットとして別の AWS アカウント を使用するには、[別のアカウントを指定] を選択します。次に、データウェアハウスを表示するアクセス許可を持つ IAM ロールの ARN を入力します。IAM ロールの作成手順については、「the section called “別のアカウントでターゲットデータウェアハウスを選択する”」を参照してください。
    2. [Amazon Redshift データウェアハウス] で、ソース DB クラスターからのレプリケートデータのターゲットを選択します。ターゲットとして、プロビジョニングされた Amazon Redshift クラスターまたは Redshift Serverless 名前空間を選択できます。

> **Note**
>
> 指定したデータウェアハウスのリソースポリシーまたは大文字と小文字の区別の設定が正しく構成されていないと、RDS から通知されます。このメッセージを受け取った場合は、[Fix it for me] を選択するか、手動で設定することができます。手動で修正する手順については、Amazon Redshift 管理ガイドの「データウェアハウスで大文字と小文字の区別を有効にする」と「データウェアハウスの認証を設定する」を参照してください。
> プロビジョニングされた Redshift クラスターの大文字と小文字の区別を変更するには、再起動が必要です。インテグレーションを作成する前に、再起動を完了し、新しいパラメータ値をクラスターに正常に適用する必要があります。
> 選択したソースとターゲットの AWS アカウント が異なる場合、Amazon RDS はこれらの設定を自動的に修正できません。他のアカウントに移動し、Amazon Redshift で手動で修正する必要があります。

11. ターゲットデータウェアハウスを正しく設定したら、[次へ] を選択します。
12. (オプション) [タグ] で、1 つ以上のタグを統合に追加します。詳細については、「the section called “RDS リソースのタグ付け”」を参照してください。
13. [暗号化] として、統合の暗号化方法を指定します。デフォルトでは、RDS はすべての統合を AWS 所有のキー で暗号化します。代わりにカスタマーマネージドキーを選択するには、[暗号化設定のカスタマイズ] を有効にして、暗号化に使用する KMS キーを選択します。詳細については、「the section called “Amazon Aurora リソースの暗号化”」を参照してください。

    > **Note**
    >
    > カスタム KMS キーを指定する場合、キーポリシーで Amazon Redshift サービスプリンシパル (`redshift.amazonaws.com`) `kms:CreateGrant` アクションが許可されている必要があります。詳細については、「AWS Key Management Service デベロッパーガイド」の「キーポリシーの作成」を参照してください。

    オプションで、暗号化コンテキストを追加します。詳しくは、AWS Key Management Service デベロッパーガイドの Encryption context を参照してください。
14. [Next] を選択します。
15. 統合設定を確認し、[ゼロ ETL 統合を作成] を選択します。

作成に失敗した場合は、トラブルシューティングの手順について「the section called “ゼロ ETL 統合を作成できない”」を参照してください。

統合のステータスは、作成中は `Creating` であり、ターゲットの Amazon Redshift データウェアハウスのステータスは `Modifying` です。この間、データウェアハウスをクエリしたり、設定を変更したりすることはできません。

統合が正常に作成されると、統合とターゲットの Amazon Redshift データウェアハウスの両方のステータスが `Active` に変わります。

AWS CLI

> **Note**
>
> Aurora PostgreSQL ゼロ ETL 統合のプレビュー中は、AWS Management Console を介してのみ統合を作成できます。AWS CLI、Amazon RDS API、または任意の SDK を使用することはできません。

AWS CLI を使用してゼロ ETL 統合を作成するには、create-integration コマンドに以下のオプションを指定して使用します。

- `--integration-name` — 統合の名前を指定します。
- `--source-arn` — 統合のソースとなる Aurora DB クラスターの ARN を指定します。
- `--target-arn` — 統合のターゲットとなる Amazon Redshift データウェアハウスの ARN を指定します。

Example

Linux、macOS、Unix の場合:

```
aws rds create-integration \
    --integration-name my-integration \
    --source-arn arn:aws:rds:{region}:{account-id}:my-cluster \
    --target-arn arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid
```

Windows の場合:

```
aws rds create-integration ^
    --integration-name my-integration ^
    --source-arn arn:aws:rds:{region}:{account-id}:my-cluster ^
    --target-arn arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid
```

RDS API

> **Note**
>
> Aurora PostgreSQL ゼロ ETL 統合のプレビュー中は、AWS Management Console を介してのみ統合を作成できます。AWS CLI、Amazon RDS API、または任意の SDK を使用することはできません。

Amazon RDS API を使用してゼロ ETL 統合を作成するには、以下のパラメータを指定して `CreateIntegration` オペレーションを使用します。

- `IntegrationName` — 統合の名前を指定します。
- `SourceArn` – 統合のソースとなる Aurora DB クラスターの ARN を指定します。
- `TargetArn` — インテグレーションのターゲットとなる Amazon Redshift データウェアハウスの ARN を指定します。

次のステップ

ゼロ ETL 統合を正常に作成した後、ターゲット Amazon Redshift クラスターまたはワークグループ内にデスティネーションデータベースを作成する必要があります。これで、ソースの Aurora DB クラスターにデータを追加し、Amazon Redshift でクエリを実行できるようになります。手順については、「Amazon Redshift でのデスティネーションデータベースの作成」を参照してください。

Amazon Redshift との Aurora ゼロ ETL 統合でのデータフィルタリング

Aurora ゼロ ETL 統合でのデータフィルタリングを使用して、ソースの Aurora DB クラスターからターゲットの Amazon Redshift データウェアハウスへのレプリケーションの範囲を定義できます。すべてのデータをターゲットにレプリケートするのではなく、単一または複数のフィルターを定義して、特定のテーブルを選択的にレプリケーションの対象に含めたり除外したりできます。ゼロ

ETL 統合の場合、フィルタリングはデータベースレベルとテーブルレベルでのみ使用できます。列や行でのフィルタリングはできません。

データフィルタリングは、次のような場合に便利です。

- 2 つ以上の異なるソースクラスターの特定のテーブルを結合し、いずれのクラスターのデータ全体は必要ない場合。
- データベース全体ではなく、テーブルのサブセットのみを使用して分析を行うことで、コストを節約する場合。
- 電話番号、住所、クレジットカード情報などの機密情報を特定のテーブルから除外する場合。

ゼロ ETL 統合には、AWS Management Console、AWS Command Line Interface (AWS CLI)、または RDS API を使用して、データフィルターを追加できます。

統合でプロビジョンした Amazon Redshift クラスターをターゲットとして使用している場合、クラスターは パッチ 180 以降である必要があります。

> **Note**
>
> 現在、データフィルタリングは、Aurora MySQL ソースを持つ統合でのみ実行できます。Aurora PostgreSQL と Amazon Redshift とのゼロ ETL 統合はプレビューリリースで、データフィルタリングはサポートされていません。

トピック

- データフィルターの形式
- フィルター論理
- フィルターの優先順位
- 例
- 統合へのデータフィルターの追加
- 統合からのデータフィルターの削除

データフィルターの形式

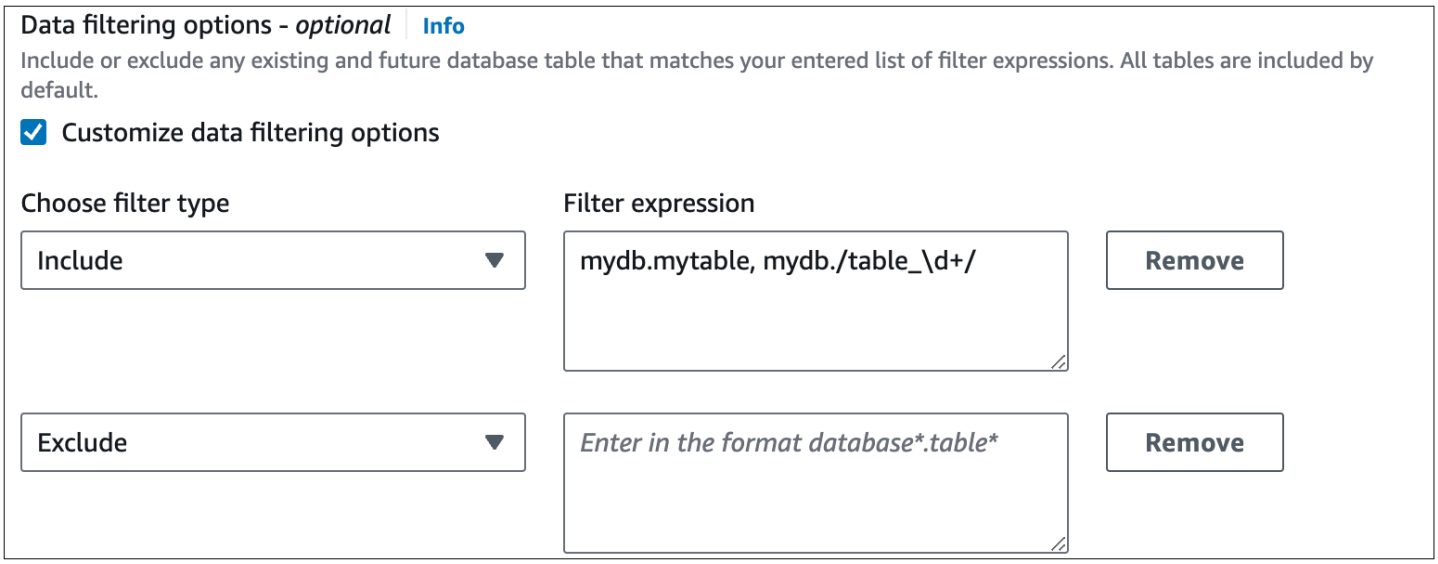
1 つの統合に対して複数のフィルターを定義できます。各フィルターは、フィルター式のパターンのいずれかに一致する既存および今後利用するデータベーステーブルを含めるまたは除外します。Aurora ゼロ ETL 統合では、データフィルタリングに Maxwell フィルター構文を使用します。

各フィルターには以下の要素が含まれます。

要素	説明
フィルタータイプ	`Include` フィルタータイプは、フィルター式のパターンのいずれかに一致するすべてのテーブルを含めます。`Exclude` フィルタータイプは、いずれかのパターンに一致するすべてのテーブルを除外します。
フィルター式	コンマ区切りのパターンのリスト。式では Maxwell フィルター構文を使用する必要があります。
パターン	*`database`*`.`*`table`* 形式のフィルターパターン。リテラルデータベース名とテーブル名 (`mydb.mytable` など) を指定したり、ワイルドカード (*) を使用したりできます。データベース名とテーブル名に正規表現を使用することもできます。 Aurora は、データベースレベルとテーブルレベルでのみフィルタリングをサポートしています。列レベルのフィルター (`database.table.column`) やブラックリスト (`blacklist: bad_db.*`) は使用できません。 1 つの統合に含めることができるパターンの合計数は、最大 99 個です。コンソールでは、1 つのフィルター式にパターンを含めることも、複数の式に分散させることもできま

要素	説明
	す。1 つのパターンの長さは 256 文字を超えることはできません。

次の図は、コンソールでのデータフィルターの構造を示しています。



> **Important**
>
> フィルターパターンには、個人を特定する情報、または機密情報を含めないでください。

AWS CLI のデータフィルター

AWS CLI を使用してデータフィルターを追加する場合、構文はコンソールと少し異なります。各パターンには、それぞれ独自のフィルタータイプ (`Include` または `Exclude`) を関連付ける必要があります。複数のパターンを 1 つのフィルタータイプにグループ化することはできません。

例えば、コンソールでは、以下のカンマで区切られたパターンを 1 つの `Include` ステートメントにまとめることができます。

```
mydb.mytable, mydb./table_\d+/
```

ただし AWS CLI を使用する際は、データフィルターを次のように記述する必要があります。

```
'include: mydb.mytable, include: mydb./table_\d+/'
```

フィルター論理

統合でデータフィルターを指定しない場合、Aurora は include:*.* をデフォルトのフィルターと見なし、すべてのテーブルをターゲットデータウェアハウスに複製します。ただし、1 つまたは複数のフィルターを指定すると、ロジックは exclude:*.* を前提として開始されます。そのため、すべてのテーブルが自動的にレプリケーションから除外されます。これにより、どのテーブルとデータベースを含めるかを直接定義できます。

例えば、次のフィルターを定義する場合、

```
'include: db.table1, include: db.table2'
```

Aurora は、フィルターを次のように解釈します。

```
'exclude:*.*, include: db.table1, include: db.table2'
```

結果として、db という名前のデータベースから table1 と table2 のみがターゲットデータウェアハウスにレプリケートされます。

フィルターの優先順位

Aurora は、指定された順番にデータフィルターを適用します。AWS Management Console では、これは Aurora がフィルター式を左から右、上から下に適用することを意味します。1 つ目のフィルターに特定のパターンを指定しても、2 つ目のフィルターや、その直後に指定された個別のパターンでそのパターンを上書きできます。

例えば、1 つ目のフィルターが Include books.stephenking で、books データベース内の stephenking という名前のテーブルが 1 つ含まれているとします。ただし、2 つ目のフィルター Exclude books.* を追加すると、その前に定義されている Include フィルターが上書きされます。結果として、books インデックスのテーブルは Amazon Redshift に複製されません。

1 つまたは複数のフィルターを指定すると、ロジックは exclude:*.* を前提として開始されます。そのため、すべてのテーブルが自動的にレプリケーションから除外されます。そのため、一般的なベストプラクティスとしては、最も範囲の広いフィルターから最も範囲の狭いフィルターの順番で定義します。例えば、1 つまたは複数の Include ステートメントを使用して、複製するデータをすべ

て定義します。次に、特定のテーブルを複製から選択的に除外する Exclude フィルターを追加します。

AWS CLI を使用して定義するフィルターにも同じ原則が適用されます。Aurora は、これらのフィルターパターンを指定された順番で適用するため、あるパターンによってその前に指定されたフィルターパターンが上書きされる場合があります。

例

以下の例は、ゼロ ETL 統合でのデータフィルタリングの仕組みを示しています。

- すべてのデータベースとすべてのテーブルを含める。

```
'include: *.*'
```

- books データベース内のすべてのテーブルを含める。

```
'include: books.*'
```

- mystery という名前のすべてのテーブルを除外します。

```
'include: *.*, exclude: *.mystery'
```

- books データベース内の 2 つの特定のテーブルを含める。

```
'include: books.stephen_king, include: books.carolyn_keene'
```

- books データベース内のすべてのテーブルを含める (ただし、単語 mystery を含むテーブルを除く)。

```
'include: books.*, exclude: books./mystery/'
```

- books データベース内の、table_ が付いているすべてのテーブルを含める (ただし、table_stephen_king という名前のテーブルを除く) 例えば、table_movies や mytable_books はレプリケートされますが、table_stephen_king はレプリケートされません。

```
'include: books./table_.*/, exclude: books.table_stephen_king'
```

統合へのデータフィルターの追加

AWS Management Console、AWS CLI、または Amazon RDS API を使用してデータフィルタリングを設定できます。

> **Important**
>
> 統合の作成後にフィルターを追加すると、Aurora はフィルターがもともと存在していたものであるかのようにフィルターを適用し始めます。ターゲットの Amazon Redshift データウェアハウスに現在あるデータのうち、新しいフィルタリング条件に一致しないものは削除されます。このアクションにより、影響を受けるすべてのテーブルが再同期されます。

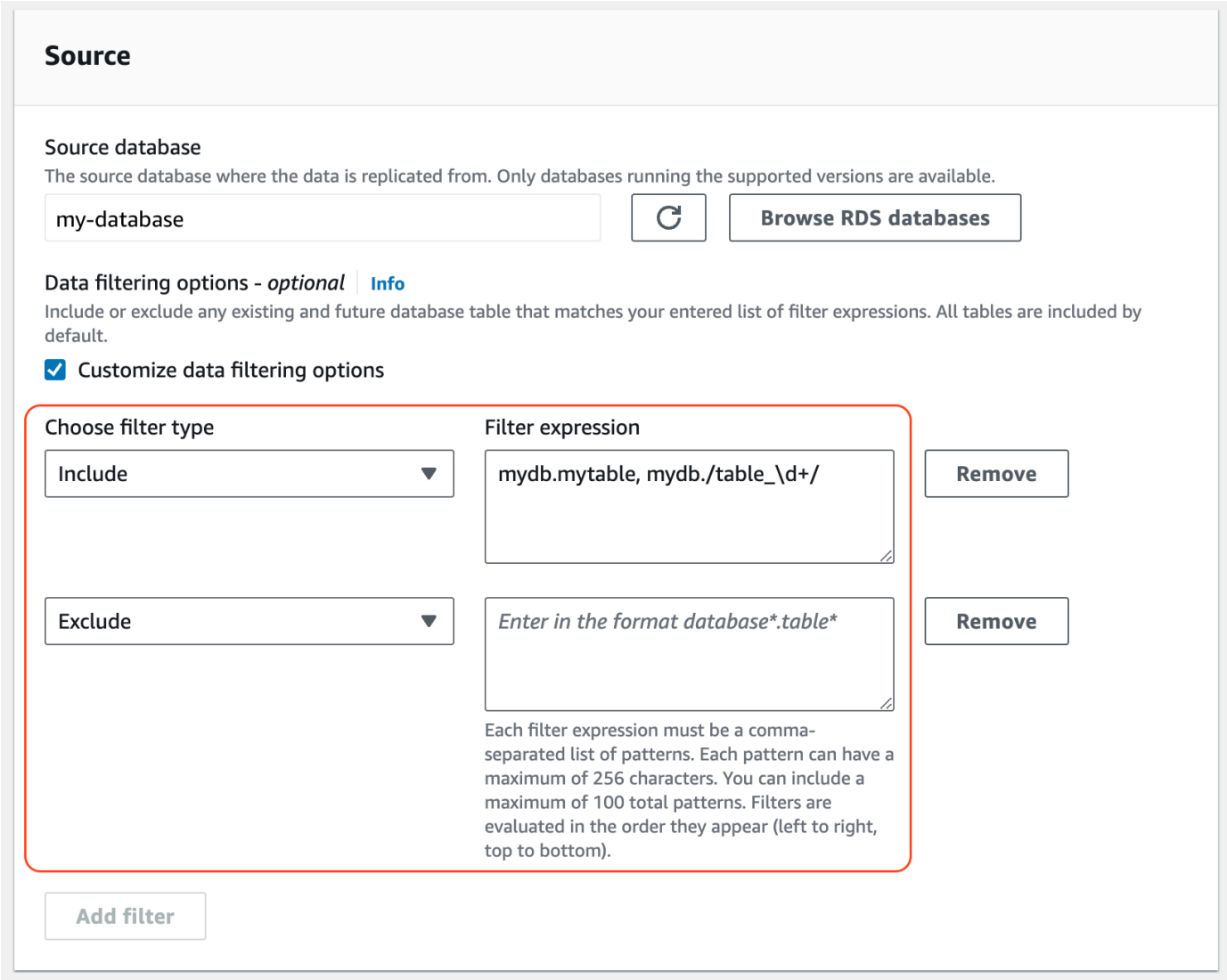
現在、データフィルタリングは Aurora MySQL ソースを持つ統合でのみ実行できます。Aurora PostgreSQL と Amazon Redshift とのゼロ ETL 統合はプレビューリリースで、データフィルタリングはサポートされていません。

RDS コンソール

ゼロ ETL 統合にデータフィルターを追加するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインから、[ゼロ ETL 統合] を選択します。データフィルターを追加する統合を選択して [変更] を選択します。
3. [ソース] で、1 つまたは複数の `Include` ステートメントと `Exclude` ステートメントを追加します。

   次の図は、統合のデータフィルターの例を示しています。



4. すべての変更を完了したら、[続行]、[変更を保存] と選択します。

AWS CLI

AWS CLI を使用してゼロ ETL 統合にデータフィルターを追加するには、modify-integration コマンドを呼び出します。統合 ID に加えて、`Include` および `Exclude` の Maxwell フィルターのカンマ区切りリストで `--data-filter` パラメーターを指定します。

Example

次の例は、`my-integration` にフィルターパターンを追加します。

Linux、macOS、Unix の場合:

```
aws rds modify-integration \
```

```
    --integration-identifier my-integration \
    --data-filter 'include: foodb.*, exclude: foodb.tbl, exclude: foodb./table_\d+/'
```

Windows の場合:

```
aws rds modify-integration ^
    --integration-identifier my-integration ^
    --data-filter 'include: foodb.*, exclude: foodb.tbl, exclude: foodb./table_\d+/'
```

RDS API

RDS API を使用してゼロ ETL 統合を変更するには、ModifyIntegration オペレーションを呼び出します。統合 ID を指定し、フィルターパターンのカンマ区切りリストを指定します。

統合からのデータフィルターの削除

統合からデータフィルターを削除すると、Aurora は削除したフィルターが存在しなかったかのようにそれ以外のフィルターを適用し始めます。その後、Aurora は、以前はフィルタリング条件に一致しなかったが現在は一致しているすべてのデータをターゲットの Amazon Redshift データウェアハウスに複製します。

1 つ以上のデータフィルターを削除すると、影響を受けるすべてのテーブルが再同期されます。

ソース Aurora DB クラスターへのデータの追加と、Amazon Redshift でのクエリ

Amazon Aurora から Amazon Redshift にデータをレプリケートするゼロ ETL 統合の作成を終了するには、Amazon Redshift に送信先データベースを作成する必要があります。

まず、Amazon Redshift クラスターまたはワークグループに接続し、統合識別子を参照してデータベースを作成します。これで、ソースの Aurora DB クラスターにデータを追加し、Amazon Redshift でクエリを実行できます。

トピック

- Amazon Redshift での送信先データベースの作成
- ソース DB クラスターへのデータの追加
- Amazon Redshift での Aurora データのクエリ

- データベースのデータタイプの違い

Amazon Redshift での送信先データベースの作成

Amazon Redshift へのデータの複製を開始する前に、統合を作成後、送信先データベースをターゲットのデータウェアハウスに作成する必要があります。この送信先データベースには、統合識別子への参照が含まれている必要があります。Amazon Redshift コンソールまたはクエリエディタ v2 を使用して、データベースを作成することができます。

デスティネーションデータベースを作成する手順については、「Amazon Redshift にデスティネーションデータベースを作成する」を参照してください。

ソース DB クラスターへのデータの追加

統合を設定した後で、Amazon Redshift データウェアハウスにレプリケートするデータを Aurora DB クラスターに追加できます。

> **Note**
>
> Amazon Aurora と Amazon Redshift のデータ型には違いがあります。データ型マッピングの表については、「the section called “データ型の相違点”」を参照してください。

まず、任意の MySQL または PostgreSQL クライアントを使用して、ソース DB クラスターに接続します。手順については、the section called “DB クラスターへの接続” を参照してください。

次に、テーブルを作成し、1 行のサンプルデータを挿入します。

> **Important**
>
> テーブルにプライマリキーがあることを確認してください。そうしないと、ターゲットのデータウェアハウスに複製できません。

pg_dump および pg_restore PostgreSQL ユーティリティは、最初にプライマリキーなしでテーブルを作成し、後で追加します。これらのユーティリティのいずれかを使用している場合は、まずスキーマを作成し、次に別のコマンドでデータをロードすることをお勧めします。

MySQL

次の例では、MySQL Workbench ユーティリティを使用しています。

```
CREATE DATABASE my_db;

USE my_db;

CREATE TABLE books_table (ID int NOT NULL, Title VARCHAR(50) NOT NULL, Author
 VARCHAR(50) NOT NULL,
Copyright INT NOT NULL, Genre VARCHAR(50) NOT NULL, PRIMARY KEY (ID));

INSERT INTO books_table VALUES (1, 'The Shining', 'Stephen King', 1977, 'Supernatural
 fiction');
```

PostgreSQL

次の例では、psql PostgreSQL インタラクティブターミナルを使用しています。クラスターに接続するときは、統合の作成時に指定した名前付きデータベースを含めます。

```
psql -h mycluster.cluster-123456789012.us-east-2.rds.amazonaws.com -p 5432 -U username
 -d named_db;

named_db=> CREATE TABLE books_table (ID int NOT NULL, Title VARCHAR(50) NOT NULL,
 Author VARCHAR(50) NOT NULL,
Copyright INT NOT NULL, Genre VARCHAR(50) NOT NULL, PRIMARY KEY (ID));

named_db=> INSERT INTO books_table VALUES (1, "The Shining", "Stephen King", 1977,
 "Supernatural fiction");
```

Amazon Redshift での Aurora データのクエリ

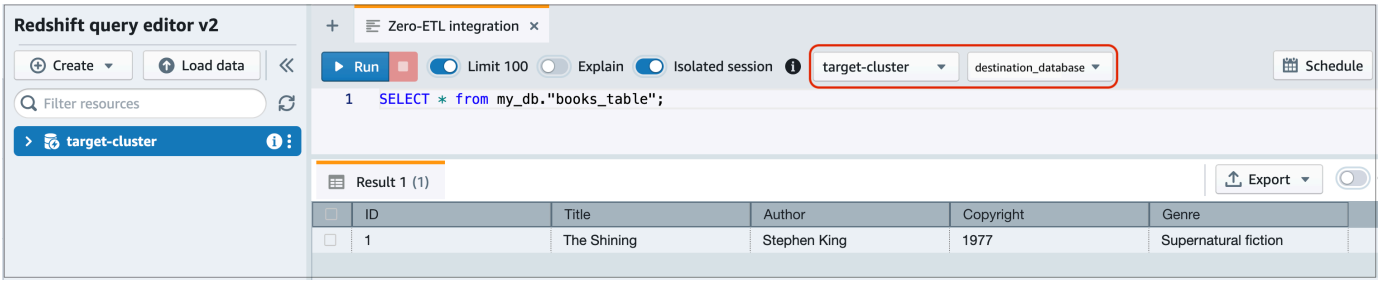
Aurora DB クラスターにデータを追加すると、Amazon Redshift にレプリケートされ、クエリを実行できるようになります。

複製されたデータをクエリするには

1. Amazon Redshift コンソールに移動し、左側のナビゲーションペインから [クエリエディタ v2] を選択します。
2. クラスターまたはワークグループに接続し、ドロップダウンメニュー (この例では destination_database) から送信先データベース (統合から作成したもの) を選択します。デスティネーションデータベースを作成する手順については、「Amazon Redshift にデスティネーションデータベースを作成する」を参照してください。

3. SELECT ステートメントを使用してデータをクエリします。この例では、次のコマンドを実行して、ソースの Aurora DB クラスターで作成したテーブルからすべてのデータを選択します。

```
SELECT * from my_db."books_table";
```



- *my_db* は Aurora データベーススキーマ名です。このオプションは MySQL データベースにのみ必要です。
- *books_table* は Aurora テーブル名です。

コマンドラインクライアントを使用してデータをクエリすることもできます。例:

```
destination_database=# select * from my_db."books_table";

 ID |       Title |        Author |   Copyright |                 Genre |  txn_seq |
 txn_id
----+-------------+---------------+-------------+-----------------------+----------
+--------+
  1 | The Shining |  Stephen King |        1977 |   Supernatural fiction |        2 |
 12192
```

> **Note**
>
> 大文字と小文字を区別するには、スキーマ、テーブル、および列の名前を二重引用符 (" ") で囲みます。詳細については、「enable_case_sensitive_identifier」を参照してください。

データベースのデータタイプの違い

次のAurora MySQL または Aurora PostgreSQL データタイプの対応する Amazon Redshift データタイプへのマッピングを示しています。Amazon Aurora は現在、ゼロ ETL 統合ではこれらのデータ型のみをサポートしています。

ソース DB クラスターのテーブルにサポートされていないデータ型が含まれている場合、そのテーブルは同期されず、Amazon Redshift ターゲットで使用できなくなります。ソースからターゲットへのストリーミングは継続されますが、サポートされていないデータ型のテーブルは使用できません。テーブルを修正して Amazon Redshift で使用できるようにするには、変更内容を手動で元に戻し、`ALTER DATABASE...INTEGRATION REFRESH` を実行して統合を更新する必要があります。

トピック

- Aurora MySQL
- Aurora PostgreSQL

Aurora MySQL

Aurora MySQL データ型	Amazon Redshift のデータ型	説明	制限事項
INT	INTEGER	符号付き 4 バイト整数	
SMALLINT	SMALLINT	符号付き 2 バイト整数	
TINYINT	SMALLINT	符号付き 2 バイト整数	
MEDIUMINT	INTEGER	符号付き 4 バイト整数	
BIGINT	BIGINT	符号付き 8 バイト整数	
INT UNSIGNED	BIGINT	符号付き 8 バイト整数	
TINYINT UNSIGNED	SMALLINT	符号付き 2 バイト整数	
MEDIUMINT UNSIGNED	INTEGER	符号付き 4 バイト整数	

Aurora MySQL データ型	Amazon Redshift のデータ型	説明	制限事項
BIGINT UNSIGNED	DECIMAL(20,0)	精度の選択が可能な真数	
DECIMAL(p,s) = NUMERIC(p,s)	DECIMAL(p,s)	精度の選択が可能な真数	精度が 38 より大きく、スケールが 37 より大きい場合、サポートされない
DECIMAL(p,s) UNSIGNED = NUMERIC(p,s) UNSIGNED	DECIMAL(p,s)	精度の選択が可能な真数	精度が 38 より大きく、スケールが 37 より大きい場合、サポートされない
FLOAT4/REAL	REAL	単精度浮動小数点数	
FLOAT4/REAL UNSIGNED	REAL	単精度浮動小数点数	
DOUBLE/REAL/FLOAT8	DOUBLE PRECISION	倍精度浮動小数点数	
DOUBLE/REAL/FLOAT8 UNSIGNED	DOUBLE PRECISION	倍精度浮動小数点数	
BIT(n)	VARBYTE(8)	可変長バイナリ値	
BINARY(n)	VARBYTE(n)	可変長バイナリ値	
VARBINARY (n)	VARBYTE(n)	可変長バイナリ値	

Aurora MySQL データ型	Amazon Redshift のデータ型	説明	制限事項
CHAR(n)	VARCHAR(n)	可変長文字列値	
VARCHAR(n)	VARCHAR(n)	可変長文字列値	
TEXT	VARCHAR(65,535)	最大 65535 バイトの可変長文字列値	
TINYTEXT	VARCHAR(255)	最大 255 バイトの可変長文字列値	
MEDIUMTEXT	VARCHAR(65,535)	最大 65535 バイトの可変長文字列値	
LONGTEXT	VARCHAR(65,535)	最大 65535 バイトの可変長文字列値	
ENUM	VARCHAR(1,020)	最大 1020 バイトの可変長文字列値	
SET	VARCHAR(1,020)	最大 1020 バイトの可変長文字列値	
DATE	DATE	カレンダー日付 (年、月、日)	
DATETIME	TIMESTAMP	日付と時刻 (タイムゾーンなし)	
TIMESTAMP(p)	TIMESTAMP	日付と時刻 (タイムゾーンなし)	

Aurora MySQL データ型	Amazon Redshift のデータ型	説明	制限事項
TIME	VARCHAR(18)	最大 18 バイトの可変長文字列値	
YEAR	VARCHAR(4)	最大 4 バイトの可変長文字列値	
JSON	SUPER	値としての半構造化データまたは文書	

Aurora PostgreSQL

Aurora PostgreSQL のゼロ ETL 統合は、カスタムデータ型または 拡張機能によって作成されたデータ型をサポートしていません。

> **Important**
>
> Aurora PostgreSQL の Amazon Redshift とのゼロ ETL 統合はプレビューリリースです。ドキュメントと機能はどちらも変更されることがあります。この機能については、テスト環境のみで使用でき、本番環境では使用できません。プレビューの利用規約については、「AWS のサービス条件」の「ベータ版とプレビュー」を参照してください。

Aurora PostgreSQL データ型	Amazon Redshift のデータ型	説明	制限事項
bigint	BIGINT	符号付き 8 バイト整数	
bigserial	BIGINT	符号付き 8 バイト整数	
bit(n)	VARBYTE(n)	可変長バイナリ値	

Aurora PostgreSQL データ型	Amazon Redshift のデータ型	説明	制限事項
bit varying(n)	VARBYTE(n)	可変長バイナリ値	
bit	VARBYTE(1,024,000)	最大 1,024,000 バイトの可変長文字列値	
ブール値	BOOLEAN	論理ブール演算型 (true/false)	
bytea	VARBYTE(1,024,000)	最大 1,024,000 バイトの可変長文字列値	
character(n)	CHAR(n)	固定長のキャラクタ文字列	
character varying(n)	VARCHAR(65,535)	可変長文字列値	
date	DATE	カレンダー日付 (年、月、日)	• サポートされていない 9999-12-31 より大きい値 • B.C. 値はサポートされていません
double precision	DOUBLE PRECISION	倍精度浮動小数点数	サポートされていない非正規化値
integer	INTEGER	符号付き 4 バイト整数	
money	DECIMAL(20,3)	通貨額	

Aurora PostgreSQL データ型	Amazon Redshift のデータ型	説明	制限事項
numeric(p,s)	DECIMAL(p,s)	可変長文字列値	• NaN 値はサポートされていません • 精度が 38 より大きく、スケールが 37 より大きい場合、サポートされない • 負のスケールはサポートされていません
real	REAL	単精度浮動小数点数	
smallint	SMALLINT	符号付き 2 バイト整数	
smallserial	SMALLINT	符号付き 2 バイト整数	
シリアル	INTEGER	符号付き 4 バイト整数	
text	VARCHAR(65,535)	最大 65,535 バイトの可変長文字列値	
time [(p)] [without time zone]	VARCHAR(19)	最大 19 バイトの可変長文字列値	Infinity および -Infinity 値はサポートされていません

Aurora PostgreSQL データ型	Amazon Redshift のデータ型	説明	制限事項
time [(p)] with time zone	VARCHAR(22)	最大 22 バイトの可変長文字列値	• `Infinity` および `-Infinity` 値はサポートされていません
timestamp [(p)] [without timezone]	TIMESTAMP	日付と時刻 (タイムゾーンなし)	• `Infinity` および `-Infinity` 値はサポートされていません • サポートされていない `9999-12-31` より大きい値 • B.C. 値はサポートされていません
timestamp [(p)] with time zone	TIMESTAMPTZ	日付と時刻 (タイムゾーンあり)	• `Infinity` および `-Infinity` 値はサポートされていません • サポートされていない `9999-12-31` より大きい値 • B.C. 値はサポートされていません

Amazon Redshift との Aurora ゼロ ETL 統合の表示と監視

Amazon Aurora ゼロ ETL 統合の詳細を表示して、その設定情報と現在のステータスを確認できます。Amazon Redshift の特定のシステムビューをクエリすることで、ゼロ ETL 統合をモニタリングすることもできます。さらに、Amazon Redshift は統合関連の特定のメトリクスを Amazon CloudWatch に公開します。これらは Amazon Redshift コンソールで確認できます。

トピック

統合の表示

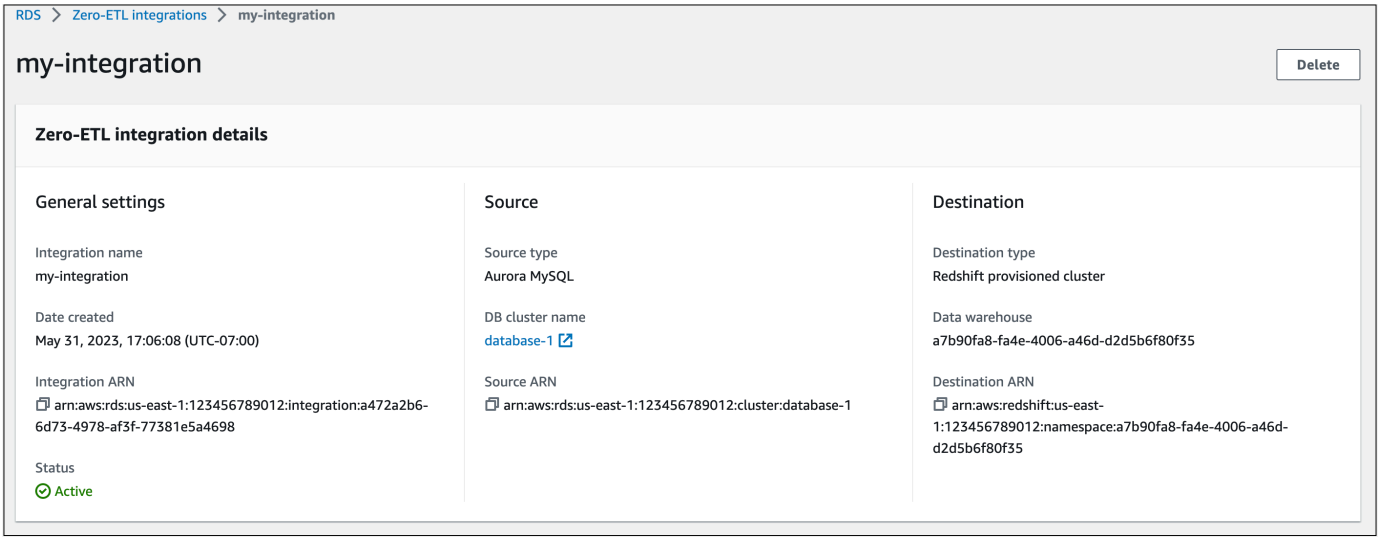
AWS Management Console、AWS CLI、または RDS API を使用して、Amazon Redshift との Amazon Aurora ゼロ ETL 統合を作成できます。

コンソール

ゼロ ETL 統合の詳細を表示するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。

   統合に Aurora PostgreSQL ソース DB クラスターがある場合、https://us-east-2.console.aws.amazon.com/rds-preview/home?region=us-east-2#databases で Amazon RDS データベースプレビュー環境にサインインする必要があります。
2. 左側のナビゲーションペインから、[ゼロ ETL 統合] を選択します。
3. そのソース DB クラスターやターゲットのデータウェアハウスなどの詳細を表示するには、統合を選択します。



統合は、以下のようなステータスを持ちます。

- `Creating` — 統合は作成中です。
- `Active` — 統合はトランザクションデータをターゲットデータウェアハウスに送信中です。
- `Syncing` — 統合で回復可能なエラーが発生したため、データを再シードしています。影響を受けるテーブルは、再同期が完了するまで Amazon Redshift でクエリを実行できません。
- `Needs attention` — 統合でイベントまたはエラーが発生したため、解決するには手動介入が必要です。問題を解決するには、統合詳細ページでエラーメッセージの指示に従ってください。
- `Failed` — 統合で、修正できない回復不能なイベントまたはエラーが発生しました。統合を手動で削除して再作成する必要があります。
- `Deleting` — 統合を削除中です。

AWS CLI

AWS CLIを使用して現在のアカウントの Zero-ETL 統合をすべて表示するには、describe-integrations コマンドを使用して `--integration-identifier` オプションを指定します。

Example

Linux、macOS、Unix の場合:

```
aws rds describe-integrations \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

Windows の場合:

```
aws rds describe-integrations ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

RDS API

Amazon RDS API を使用してゼロ ETL 統合を表示するには、`IntegrationIdentifier` パラメータを指定して `DescribeIntegrations` オペレーションを使用します。

システムテーブルを使用して統合をモニタリングします。

Amazon Redshift には、システムの動作に関する情報を含むシステムテーブルとビューがあります。これらのシステムテーブルとビューには、その他のデータベーステーブルと同じ方法でクエリを実行できます。Amazon Redshift のシステムテーブルとビューの詳細については、「Amazon Redshift データベースデベロッパーガイド」の「システムテーブルリファレンス」を参照してください。

以下のシステムビューとテーブルにクエリして、Amazon Redshift との Aurora ゼロ ETL 統合に関する情報を取得できます。

- SVV_INTEGRATION — 統合の設定の詳細を提供します。
- SVV_INTEGRATION_TABLE_STATE — 統合内の各テーブルの状態を記述します。
- SYS_INTEGRATION_TABLE_STATE_CHANGE — 統合のテーブルステート変更ログを表示します。
- SYS_INTEGRATION_ACTIVITY — 完了した統合実行に関する情報を提供します。

統合関連の CloudWatch メトリクスはすべて Amazon Redshift から生成されます。詳細については、「Amazon Redshift 管理ガイド」の「ゼロ ETL 統合のモニタリング」を参照してください。現在、Amazon Aurora は統合関連のメトリクスを CloudWatch に公開していません。

Amazon EventBridge との統合のモニタリング

Amazon Redshift ridge は統合関連のイベントを Amazon EventBridge に送信します。イベントとそれに対応するイベント ID のリストについては、Amazon Redshift 管理ガイドの「Amazon EventBridge によるゼロ ETL 統合イベント通知」を参照してください。

Amazon Redshift との Aurora ゼロ ETL 統合の変更

Amazon Redshift とのゼロ ETL 統合では、名前、説明、データフィルタリングオプションのみを変更できます。統合の暗号化に使用される AWS KMS キー、ソースデータベースまたはターゲットデータベースは変更できません。

既存の統合にデータフィルターを追加すると、Aurora は、そのフィルターが本来存在していたものであるかのように再評価します。ターゲットの Amazon Redshift データウェアハウス内に現在あるデータのうち、新しいフィルタリング条件に一致しないものは削除されます。統合からデータフィルターを削除すると、以前はフィルター条件に一致していなかった (が現在は一致している) データがターゲットデータウェアハウスにレプリケートされます。詳細については、「the section called “ゼロ ETL 統合でのデータフィルタリング”」を参照してください。

AWS Management Console、AWS CLI、または Amazon RDS API を使用して、ゼロ ETL 統合を変更できます。

> **Note**
>
> 現在は、Aurora MySQL ソース DB クラスターを含む統合のみを変更できます。Amazon Redshift と Aurora PostgreSQL のゼロ ETL 統合のプレビューリリースでは、統合の変更はサポートされていません。

RDS コンソール

ゼロ ETL 統合を変更するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[ゼロ ETL 統合] を選択して、変更する統合を選択します。
3. [変更] を選択して、変更可能な設定をすべて変更します。
4. すべての変更が完了したら、[変更] を選択します。

AWS CLI

AWS CLI を使用してゼロ ETL 統合を変更するには、modify-integration コマンドを呼び出します。`--integration-identifier` に加えて、次のいずれかのオプションを指定します。

- `--integration-name` — 統合の新しい名前を指定します。
- `--description` — 統合の新しい説明を指定します。
- `--data-filter` — 統合のデータフィルタリングオプションを指定します。詳細については、「the section called “ゼロ ETL 統合でのデータフィルタリング”」を参照してください。

Example

次のリクエストは既存のインテグレーションを変更します。

Linux、macOS、Unix の場合:

```
aws rds modify-integration \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374 \
    --integration-name my-renamed-integration
```

Windows の場合:

```
aws rds modify-integration ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374 ^
    --integration-name my-renamed-integration
```

RDS API

RDS API を使用してゼロ ETL 統合を変更するには、ModifyIntegration オペレーションを呼び出します。統合識別子と変更するパラメータを指定します。

Amazon Redshift との Amazon ゼロ ETL 統合の削除

ゼロ ETL 統合を削除すると、Amazon Aurora はソースの Aurora DB クラスター からその ETL 統合を削除します。トランザクションデータが Amazon Aurora または Amazon Redshift から削除されることはありませんが、Aurora は新しいデータを Amazon Redshift に送信しません。

統合は、ステータスが `Active`、`Failed`、`Syncing`、または `Needs attention` のときにのみ削除できます。

AWS Management Console、AWS CLI、または RDS API を使用してゼロ ETL 統合を削除できます。

コンソール

ゼロ ETL 統合を削除するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。

   統合に Aurora PostgreSQL ソース DB クラスターがある場合、https://us-east-2.console.aws.amazon.com/rds-preview/home?region=us-east-2#databases で Amazon RDS データベースプレビュー環境にサインインする必要があります。
2. 左側のナビゲーションペインから、[ゼロ ETL 統合] を選択します。
3. 削除するゼロ ETL 統合を選択します。
4. [アクション]、[削除] を選択し、削除を確定します。

AWS CLI

> **Note**
>
> Aurora PostgreSQL ゼロ ETL 統合のプレビュー中は、AWS Management Console を介してのみ統合を削除できます。AWS CLI、Amazon RDS API、または任意の SDK を使用することはできません。

ゼロ ETL 統合を削除するには、delete-integration コマンドを使用して `--integration-identifier` オプションを指定します。

Example

Linux、macOS、Unix の場合:

```
aws rds delete-integration \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

Windows の場合:

```
aws rds delete-integration ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

RDS API

> **Note**
>
> Aurora PostgreSQL ゼロ ETL 統合のプレビュー中は、AWS Management Console を介してのみ統合を削除できます。AWS CLI、Amazon RDS API、または任意の SDK を使用することはできません。

Amazon RDS API を使用してゼロ ETL 統合を削除するには、`IntegrationIdentifier` パラメータを指定して `DeleteIntegration` オペレーションを使用します。

Aurora ゼロ ETL 統合のトラブルシューティング

Amazon Redshift の SVV_INTEGRATION システムテーブルにクエリを実行することで、ゼロ ETL 統合の状態を確認できます。`state` 列の値が `ErrorState` の場合、何か問題があることを意味します。詳細については、「the section called “システムテーブルを使ったモニタリング”」を参照してください。

以下の情報を使用して、Aurora ゼロ ETL 統合に関する一般的な問題をトラブルシューティングしてください。

トピック

- ゼロ ETL 統合を作成できない
- 統合が Syncing の状態でスタックしている
- テーブルが Amazon Redshift にレプリケートされない
- 1 つ以上の Amazon Redshift テーブルを再同期する必要がある

ゼロ ETL 統合を作成できない

ゼロ ETL 統合を作成できない場合は、ソース DB クラスターについて以下が正しいことを確認してください。

- ソースクラスターは、Aurora MySQL バージョン 3.05 (MySQL 8.0.32 互換) 以降または Aurora PostgreSQL (PostgreSQL 15.4 およびゼロ ETL サポートと互換) を実行しています。エンジンバージョンを確認するには、DB クラスターの [設定] タブを選択して、[エンジンバージョン] を確認します。

- DB クラスターのパラメータを正しく設定しました。必須パラメータが正しく設定されていないか、クラスターに関連付けられていない場合、作成は失敗します。「the section called “ステップ 1: カスタム DB クラスターのパラメータグループを作成する”」を参照してください。

さらに、ターゲットデータウェアハウスについて、以下が正しいことを確認してください。

- 大文字と小文字の区別が有効になっている。「データウェアハウスの大文字と小文字の区別を有効にする」を参照してください。
- 正しい承認済みプリンシパルと統合ソースを追加した。「Amazon Redshift データウェアハウスの認証を設定する」を参照してください。
- データウェアハウスは暗号化されている (プロビジョニングされたクラスターの場合)。「Amazon Redshift データベース暗号化」を参照してください。

統合が Syncing の状態でスタックしている

必須 DB パラメータのいずれかの値を変更すると、統合のステータスが常に Syncing と表示されることがあります。

この問題を解決するには、ソース DB クラスターに関連付けられているパラメータグループのパラメータの値をチェックして、必要な値と一致していることを確認します。詳細については、「the section called “ステップ 1: カスタム DB クラスターのパラメータグループを作成する”」を参照してください。

パラメータを変更した場合は、必ず DB クラスターを再起動して変更を適用してください。

テーブルが Amazon Redshift にレプリケートされない

1 つ以上のソーステーブルにプライマリキーがないため、データがレプリケートされていない可能性があります。Amazon Redshift のモニタリングダッシュボードには、これらのテーブルのステータスが Failed と表示され、ゼロ ETL 統合全体のステータスが Needs attention に変わります。

この問題を解決するには、プライマリキーとなる既存のキーをテーブル内で特定するか、合成プライマリキーを追加することができます。詳細な解決策については、以下のリソースを参照してください。

- Amazon Aurora MySQL または Amazon RDS for MySQL と Amazon Redshift とのゼロ ETL 統合を作成する際に、プライマリキーがないテーブルを処理する

- Amazon Aurora PostgreSQL と Amazon Redshift とのゼロ ETL 統合を作成する際に、プライマリキーがないテーブルを処理する

1 つ以上の Amazon Redshift テーブルを再同期する必要がある

ソース DB クラスターに対して特定のコマンドを実行するには、テーブルの再同期が必要になる場合があります。このような場合、SVV_INTEGRATION_TABLE_STATE システムビューには `table_state` が `ResyncRequired` と表示されます。つまり、統合は MySQL から Amazon Redshift に特定のテーブルを完全にリロードする必要があります。

テーブルが再同期を開始すると、`Syncing` の状態はになります。テーブルを再同期するために手動で操作を行う必要はありません。テーブルデータの再同期中は、Amazon Redshift からデータにアクセスすることはできません。

以下に、テーブルを `ResyncRequired` 状態にする操作の例と、検討すべき代替案をいくつか示します。

操作	例	代替

特定の位置への列の追加	`ALTER TABLE table_name`
	`  ADD COLUMN column_name  INTEGER`
	`  NOT NULL first;`
		Amazon Redshift は、`first` または `after` キーワードを使用して特定の位置に列を追加することをサポートしていません。ターゲットテーブルの列の順序が重要でない場合は、より簡単なコマンドを使用してテーブルの最

操作	例	代替
		後に列を追加します。 `ALTER TABLE table_name ADD COLUMN column_name column_type ;`

操作	例	代替

デフォルトの CURRENT_TIMESTAMP でのタイムスタンプ列の追加	`ALTER TABLE table_name`
	`  ADD COLUMN column_name  TIMESTAMP`
	`  NOT NULL DEFAULT CURRENT_TIMESTAMP;`
		既存のテーブル行の CURRENT_TIMESTAMP 値は Aurora MySQL によって計算されるため、テーブルデータを完全に再同期しない限り、Amazon Redshift でシミュレートすることはできません。
		
		可能であれば、テーブルが利用可能になるまでの待ち時間を避けるために、デフォルト値を 2023-01-01 00:00:15 などのリテラル定数に切り替えてください。

操作	例	代替

1 つのコマンドで複数の列操作を実行する	`ALTER TABLE table_name`
	`  ADD COLUMN column_1,`
	`  RENAME COLUMN column_2 TO column_3;`
		コマンドを ADD と RENAME の 2 つの操作に分割することを検討してください。この場合、再同期は不要です。

Aurora Serverless v2 を使用する

Aurora Serverless v2 は、Amazon Aurora 用のオンデマンドのオートスケーリング設定です。Aurora Serverless v2 によって、ワークロードをモニタリングし、データベースの容量を調整するプロセスを自動化しやすくなります。容量は、アプリケーションの需要に応じて自動的に調整されます。DB クラスターが消費するリソースに対してのみ課金されます。このように、Aurora Serverless v2 によって予算内に収め、使用しないコンピュータリソースに対して料金を支払うことを避けるのに役立ちます。

この種の自動化は、マルチテナントデータベース、分散型データベース、開発、テストシステムなど、さらにワークロードが大きく変動し、予測不可能な環境において特に有効です。

トピック

Aurora Serverless v2 ユースケース

Aurora Serverless v2 は、多くのタイプのデータベースワークロードをサポートしています。その対象は、開発環境やテスト環境から、予測不可能なワークロードのあるウェブサイトやアプリケーション、高い拡張性と可用性を必要とする最も要求の厳しいビジネスクリティカルなアプリケーションまで多岐にわたります。

Aurora Serverless v2 は、以下のユースケースに特に役立ちます。

- 変動するワークロード - 突然で予測不可能なアクティビティの増加が発生するようなワークロードを実行している場合。雨が降り出したときにアクティビティが急増 (サージ) するトラフィックサイトなどが該当します。別のケースとして、販売や特別なプロモーションを行うことで、トラ

フィックが増加する e コマースサイトがあります。Aurora Serverless v2 では、アプリケーションのピーク時に必要なロードに合わせて、データベースの容量がオートスケーリングされ、アクティビティのサージが終了した時点でスケールダウンして元に戻ります。Aurora Serverless v2 を導入することで、ピーク容量や平均容量に合わせてプロビジョニングする必要はなくなります。最悪の状況に対応するために容量の上限を指定でき、その容量は必要な場合以外使用されません。

Aurora Serverless v2 のスケーリングの詳細度は、データベースのニーズに合わせて容量を細かく調整しやすくします。プロビジョン済みクラスターの場合、スケールアップには、完全に新しい DB インスタンスを追加する必要があります。Aurora Serverless v1 クラスターの場合、スケールアップするには、クラスターの Aurora 容量単位 (ACU) の数を 16 から 32、32 から 64 のように 2 倍にする必要があります。一方、Aurora Serverless v2 では、あと少しだけ容量が必要な場合、ACU を半分追加できます。ワークロードの増加に対応するために追加が必要な容量によって、0.5、1、1.5、2、または半分の ACU を追加できます。また、ワークロードが減少し、その容量が不要になった場合、0.5、1、1.5、2、または追加した半分の ACU を削除できます。

- マルチテナントアプリケーション - Aurora Serverless v2 を使用することで、フリート内のアプリケーションごとのデータベース容量を、ユーザーが個別に管理する必要はなくなります。個々のデータベース容量は、Aurora Serverless v2 により自動的に管理されます。

  テナントごとにクラスターを作成できます。これにより、クローン、スナップショットリストア、Aurora グローバルデータベースなどの機能を使用して、テナントごとに高可用性と災害対策を強化できます。

  各テナントには、時間帯、時期、プロモーションイベントなどに応じて、繁忙期と休止期間が設定される場合があります。各クラスターには、広い範囲で容量を指定できます。これにより、アクティビティの少ないクラスターでは DB インスタンスの料金を最小限に抑えることができます。どのクラスターでも、アクティビティの高い期間に対応できるように迅速にスケールアップできます。

- 新しいアプリケーション - 現在デプロイ中で、必要な DB インスタンスサイズが明確でない、新しいアプリケーション。Aurora Serverless v2 を使用して、1 つまたは複数の DB インスタンスでクラスターを設定し、アプリケーションの容量の要件に応じてデータベースをオートスケーリングできます。

- 複合用途のアプリケーション – オンライントランザクション処理 (OLTP) アプリケーションを使用しているが、クエリトラフィックが定期的に急増することがある場合。クラスター内の Aurora Serverless v2 DB インスタンスに昇格階層を指定することで、リーダー DB インスタンスがライター DB インスタンスと独立してスケーリングして、追加のロードを処理できるようにクラスターを構成できます。使用率の急増が収まったら、リーダー DB インスタンスによってライター DB インスタンスの容量に合わせてスケールダウンします。

- 容量計画 – 通常、クラスター内のすべての DB インスタンスの DB インスタンスクラスを変更して、データベース容量を調整するか、ワークロードに最適なデータベース容量を検証します。Aurora Serverless v2 では、この管理オーバーヘッドを回避できます。ワークロードを実行し、DB インスタンスが実際にスケールする量をチェックすることで、適切な最小容量と最大容量を決定できます。

  既存の DB インスタンスを、プロビジョン済みから Aurora Serverless v2 に、または Aurora Serverless v2 からプロビジョン済みに変更できます。このような場合、新しいクラスターや新しい DB インスタンスを作成する必要はありません。

  Aurora グローバルデータベースでは、セカンダリクラスターには、プライマリクラスターと同程度の容量は必要ない場合があります。Aurora Serverless v2 セカンダリクラスター内の DB インスタンスを使用できます。これにより、セカンダリリージョンが昇格してアプリケーションのワークロードを引き継いだ場合に、クラスターの容量をスケールアップできます。

- 開発とテスト — 最も要求の厳しいアプリケーションの実行に加えて、開発環境やテスト環境にも Aurora Serverless v2 を使用できます。Aurora Serverless v2 により、バースト db.t* DB インスタンスクラスを使用する代わりに、最小容量が小さい DB インスタンスを作成できます。最大容量を大きく設定することで、これらの DB インスタンスのメモリが不足せず、大量のワークロードを実行できます。データベースが使用されていない場合は、すべての DB インスタンスがスケールダウンすることで、不要な料金が発生しないようにします。

  > **Tip**
  >
  > 環境やテスト環境で Aurora Serverless v2 を便利に使用できるように、AWS Management Console では、新しいクラスターを作成する場合に Easy create ショートカットを提供しています。開発/テストオプションを選択した場合、Aurora では Aurora Serverless v2 DB インスタンスと、開発およびテストシステム向けに一般的な容量範囲のクラスターを作成します。

既存のプロビジョニング済みワークロードに Aurora Serverless v2 を使用

プロビジョン済みクラスターで既に Aurora アプリケーションが実行されているとします。リーダー DB インスタンスとして、1 つまたは複数の Aurora Serverless v2 DB インスタンスを追加することにより、Aurora Serverless v2 によってアプリケーションがどのように動作するか確認できます。リーダー DB インスタンスのスケールアップとスケールダウンの頻度を確認できます。Aurora フェイルオーバーメカニズムを使用して、Aurora Serverless v2 DB インスタンスをライターに昇格さ

せ、読み取り/書き込みワークロードの処理方法を確認できます。これにより、クライアントアプリケーションが使用するエンドポイントを変更することなく、最小限のダウンタイムで切り替えが可能です。既存のクラスターを Aurora Serverless v2 に変換する手順の詳細については、「Aurora Serverless v2 への移行」を参照してください。

Aurora Serverless v2 の利点

Aurora Serverless v2 は、可変または「スパイキー」ワークロードを対象としています。このような予測不可能なワークロードでは、データベース容量を変更するタイミングを計画するのが難しい場合があります。また、容量を迅速に変更するために、DB インスタンスの追加や DB インスタンスクラスの変更など、使い慣れたメカニズムでは十分ではない場合があります。Aurora Serverless v2 には、このようなユースケースに役立つ以下のような利点があります。

- プロビジョニングよりも簡単な容量管理 – Aurora Serverless v2ワークロードの変化に応じて DB インスタンスのサイズを計画したり、DB インスタンスのサイズを変更したりするための労力を削減します。また、クラスター内のすべての DB インスタンスの容量を一定に維持するための労力が削減されます。
- 高アクティビティ時のスケーリングを高速かつ簡単に実行 – Aurora Serverless v2クライアントトランザクションやワークロード全体を中断することなく、必要に応じてコンピューティング性能とメモリ容量をスケーリングします。Aurora Serverless v2 でリーダー DB インスタンスを使用できることで、垂直方向のスケーリングに加え、水平方向のスケーリングも利用できます。Aurora グローバルデータベースを使用できるということは、Aurora Serverless v2 の読み取りワークロードを複数の AWS リージョン に分散できるということです。この機能は、プロビジョン済みクラスターのスケーリングメカニズムよりも便利です。また、Aurora Serverless v1 のスケーリング機能よりも高速で、詳細になっています。
- アクティビティの少ない期間の費用対効果が高い – Aurora Serverless v2 DB インスタンスのオーバープロビジョニングを回避するのに役立ちます。Aurora Serverless v2 DB インスタンスのスケールアップ時に、リソースをきめ細かい単位で追加します。消費したデータベースリソースのみ料金をお支払いいただきます。Aurora Serverless v2 リソースの使用量は、秒単位で計測します。これにより、DB インスタンスがスケールダウンすると、削減されたリソース使用量がすぐに登録されます。
- プロビジョニングと同等以上の機能 — Aurora Serverless v2 で Aurora の多くの機能を使用できます。なお、Aurora Serverless v1 では利用できません。例えば、Aurora Serverless v2 と指定すると、リーダー DB インスタンス、グローバルデータベース、AWS Identity and Access Management(IAM) データベース認証、パフォーマンスインサイトが使用できます。また、Aurora Serverless v1 の場合よりも多くの設定パラメータを使用することもできます。

特に Aurora Serverless v2 では、プロビジョン済みクラスターによって、以下機能を活用できます。

- リーダー DB インスタンス – Aurora Serverless v2 では、リーダー DB インスタンスを活用して水平方向にスケーリングできます。クラスターに 1 つまたは複数のリーダー DB インスタンスが含まれている場合、ライター DB インスタンスに問題が発生した場合に、クラスターはすぐにフェイルオーバーできます。これは、Aurora Serverless v1 では使用できない機能です。
- マルチ AZ クラスター — クラスターの Aurora Serverless v2 DB インスタンスは、複数のアベイラビリティーゾーン (AZ) に分散できます。マルチ AZ クラスターを設定することで、AZ 全体に影響する問題が発生するようなまれなケースでも、ビジネスの継続性を確保できます。これは、Aurora Serverless v1 では使用できない機能です。
- グローバルデータベース — Aurora Serverless v2 を Aurora グローバルデータベースと組み合わせて使用することで、災害対策用として他の AWS リージョン にクラスターの読み取り専用のコピーを追加で作成できます。
- RDS Proxy – Amazon RDS Proxy を使用すると、アプリケーションでデータベース接続をプールおよび共有して、アプリケーションのスケーリング能力を向上させることができます。

- Aurora Serverless v1 より高速、詳細で、中断が少ないスケーリング – Aurora Serverless v2 によって、より速くスケールアップ、スケールダウンできます。スケーリングでは、ACU の数を 2 倍または半分にする代わりに、0.5 ACU という少ない単位で容量を変更できます。スケーリングは通常、処理を一度も中断することなく行われます。スケーリングでは、Aurora Serverless v1 のように注意しなければならないイベントは発生しません。クワイエットポイントを待つ必要はなく、SQL ステートメントの実行中やトランザクションが開いている間にスケーリングを行うことができます。

Aurora Serverless v2 の働き

以下の概要では、Aurora Serverless v2 の仕組みについて説明します。

トピック

- Aurora クラスターの設定パラメータ

Aurora Serverless v2 の概要

Amazon Aurora Serverless v2 は、最も変化が激しく、要求の厳しいワークロードに適しています。用途の例としては、データベースの使用負荷が短時間の間だけ増大し、その後に軽いアクティビティが長時間続くか、またはアクティビティがまったく発生しなくなるケースが挙げられます。例えば、定期的に販売促進イベントを行う小売り、ゲーム、スポーツなどのウェブサイト、必要なときにレポートを作成するレポートデータベースなどがあります。また、開発やテスト環境、また、新しいアプリケーションでは急激に利用が増加することがあります。他にも多くが考えられますが、このようなケースに対してプロビジョニングされたモデルを使用しても、事前に容量を正しく指定できるとは限りません。また、過剰なプロビジョニングを行い、使用しない容量が生じた場合には、コストが高くなる可能性もあります。

一方で、プロビジョン済み Aurora クラスターは、安定したワークロードに適しています。プロビジョン済みクラスターでは、メモリサイズ、CPU パワー、I/O 帯域幅などが事前定義された DB インスタンスクラスを選択します。ワークロードが変更された場合、ライターとリーダーのインスタンスクラスを手動で変更します。プロビジョン済みモデルは、消費パターンが予想され、事前に容量を調整できる場合に有効です。クラスター内のライターとリーダーのインスタンスクラスを変更しながら、短時間の停止の発生が許容される場合は有効に機能します。

Aurora Serverless v2 は、すぐにスケーリング可能なサーバーレス DB クラスターをサポートするためにゼロから設計されています。また、Aurora Serverless v2 は、プロビジョン済みライターやリーダーと同レベルのセキュリティと分離を提供するように設計されています。このような側面は、マルチテナントのサーバーレスクラウド環境では非常に重要です。データベースのワークロードの変更に迅速に対応できるように、動的スケーリングメカニズムにはオーバーヘッドがほとんどありません。また、処理需要の劇的な増加に対応するための、十分な能力も備わっています。

Aurora Serverless v2 を使用することにより、各ライターとリーダーにある特定のデータベース容量の制約を受けずに Aurora DB クラスターを作成できます。お客様は、最小容量と最大容量の範囲を指定します。Aurora では、その容量範囲内のクラスター内の各 Aurora Serverless v2 ライターまたはリーダーをスケーリングします。各ライターまたはリーダーによって動的にスケーリングできるマルチ AZ クラスターを使用することで、動的スケーリングと高可用性を活用できます。

Aurora Serverless v2 では、最小容量と最大容量の仕様に基づいて、データベースリソースを自動的にスケーリングします。ほとんどのスケーリングイベントのオペレーションは、ライターまたはリーダーが同じホスト上で保持されるため、高速にスケーリングします。Aurora Serverless v2 ライターまたはリーダーが、あるホストから別のホストに移動するまれなケースでも、Aurora Serverless v2

では自動的に接続を管理します。データベースクライアントアプリケーションのコードやデータベースの接続文字列を変更する必要はありません。

Aurora Serverless v2 では、プロビジョン済みクラスターと同様に、ストレージ容量とコンピューティング性能は別々になっています。Aurora Serverless v2 容量やスケーリングに言及した場合、増減するのは常にコンピューティング性能です。したがって、CPU やメモリの容量がスケールダウンしても、クラスターには数テラバイトのデータを格納できます。

プロビジョニングやデータベースサーバーを管理する代わりに、データベース容量を指定します。Aurora Serverless v2 の容量についての詳細は、「Aurora Serverless v2 の容量」を参照してください。各 Aurora Serverless v2 ライターまたはリーダーの実際の容量は、ワークロードによって時間とともに変化します。このメカニズムの詳細については、「Aurora Serverless v2 でのスケーリング」を参照してください。

> **Important**
>
> Aurora Serverless v1 では、クラスターには、最小容量から最大容量の値の間でスケーリングできるコンピューティング性能の単一のメジャーがあります。Aurora Serverless v2 では、クラスターにはライターに加えてリーダーを含めることができます。各 Aurora Serverless v2 ライターとリーダーは、最小容量から最大容量までの値をスケーリングできます。したがって、Aurora Serverless v2 クラスターの全容量は、DB クラスターに定義された容量範囲と、クラスター内のライターとリーダーの数の両方で決まります。どの時点でも、Aurora DB クラスターで実際に使用している Aurora Serverless v2 の容量に対してのみ課金されます。

Aurora DB クラスターの設定

Aurora DB クラスターごとに、Aurora Serverless v2 の容量およびプロビジョン済み容量、またはその両方を自由に組み合わせて選択できます。

混在設定クラスターと呼ばれる Aurora Serverless v2 とプロビジョン済み容量の両方を含むクラスターを設定できます。例えば、Aurora Serverless v2 ライターで利用可能な容量よりも、多くの読み取り/書き込み容量が必要だとします。この場合、非常に大きいプロビジョン済みライターを使用してクラスターをセットアップできます。その場合でも、引き続き Aurora Serverless v2 リーダーを使用できます。また、クラスターの書き込みワークロードは変化しているが、読み取りワークロードは安定しているとします。この場合、クラスターに 1 つの Aurora Serverless v2 ライターと 1 つまたは複数のプロビジョン済みリーダーをセットアップできます。

Aurora Serverless v2 によってすべての容量が管理される DB クラスターをセットアップすることもできます。これを行うには、新しいクラスターを作成し、最初から Aurora Serverless v2 を使用します。または、Aurora Serverless v2 で既存のクラスター内でプロビジョン済みのすべての容量を置き換えることができます。例えば、古いバージョンのエンジンからのアップグレードパスには、プロビジョン済みライターで開始して、Aurora Serverless v2 ライターで置き換えが必要なものもあります。Aurora Serverless v2 で新しい DB クラスターを作成するか、または既存の DB クラスターを Aurora Serverless v2 に切り替える手順の詳細については、「Aurora Serverless v2 DB クラスターの作成」および「プロビジョニングされたクラスターから Aurora Serverless v2 への切り替え」を参照してください。

DB クラスターで Aurora Serverless v2 をまったく使用しない場合、DB クラスター内のすべてのライターとリーダーはプロビジョン済みになります。これは、ほとんどのユーザーがよく知っている、最も古く、最も一般的な種類の DB クラスターです。実際に、Aurora Serverless の前には、このような種類の Aurora DB クラスターには特別な名前はありませんでした。プロビジョン済み容量は一定です。料金は比較的簡単に予測できます。ただし、必要な容量を事前に予測する必要があります。場合によっては、予測が不正確だったり、容量のニーズが変わったりすることもあります。このような場合、DB クラスターがプロビジョニングされない (希望よりも遅い)、またはオーバープロビジョニング (必要以上に高価) になる可能性があります。

Aurora Serverless v2 の容量

Aurora Serverless v2 の単位は Aurora 容量単位 (ACU).です。Aurora Serverless v2 の容量は、プロビジョン済みクラスターに使用する DB インスタンスクラスと関連性はありません。

各 ACU では、約 2 ギビバイト (GiB) のメモリと、対応する CPU、ネットワークが組み合わせられています。この単位を使用して、データベース容量の範囲を指定します。`ServerlessDatabaseCapacity` および `ACUUtilization` メトリクスは、データベースが実際に使用している容量と、その容量が指定された範囲内のどこにあるかを判断するのに役立ちます。

いつどんな場合でも、各 Aurora Serverless v2 DB ライターまたはリーダーは容量を持っています。容量は、ACU を示す浮動小数点数で表されます。容量は、ライターまたはリーダーがスケーリングするごとに増減します。この値は毎秒測定されます。Aurora Serverless v2 を使用する予定の各 DB クラスターに、各 Aurora Serverless v2 ライターまたはリーダーの間でスケーリングできる最小容量および最大容量の値である容量範囲を定義します。各 Aurora Serverless v2DB クラスターのライターまたはリーダーでは、容量範囲は同じです。Aurora Serverless v2 ライターまたはリーダーは、それぞれの範囲内の容量を持っています。

定義可能な最大 Aurora Serverless v2 容量は 128 ACU です。最大容量値を選択する際のすべての考慮事項については、「クラスターに Aurora Serverless v2 の最大容量設定を選択する」を参照してください。

定義可能な最小 Aurora Serverless v2 容量は 0.5 ACU です。最大容量値以下であれば、それ以上の数値を指定できます。最小容量を小さい量に設定することで、低ロードの DB クラスターでは最小限のコンピューティングリソースを消費できます。同時に、直ちに接続を受け入れ、ビジーになったらスケールアップする準備ができています。

各 DB ライターまたはリーダーがバッファプール内のアプリケーションのワーキングセットを保持できる最小値に設定することをお勧めします。これにより、アイドル中にバッファプールの内容が破棄されることはありません。最小容量値を選択する際のすべての考慮事項については、「クラスターに Aurora Serverless v2 の最小容量設定を選択する」を参照してください。

マルチ AZ DB クラスターでリーダーの設定方法に応じて、その容量はライターの容量に関連付けることも、独立させることもできます。これを行う方法については、「Aurora Serverless v2 でのスケーリング」を参照してください。

Aurora Serverless v2 のモニタリングでは、DB クラスター内のライターとリーダーの容量値を経時的に測定します。データベースが最小容量にスケールダウンしない場合は、最小値の調整やデータベースアプリケーションの最適化などのアクションを実行できます。データベースが常に最大容量に達している場合は、最大容量を増やすなどのアクションを実行できます。また、データベースアプリケーションを最適化し、クエリのロードをより多くのリーダーに分散させることもできます。

Aurora Serverless v2 容量の料金は ACU の時間で測定されます。Aurora Serverless v2 料金の計算方法については、「Aurora 料金のページ」をご覧ください。

クラスター内のライターとリーダーの合計が N であるとします。この場合、データベースのオペレーションを実行していない場合のクラスターの消費量は約 *n* x *minimum ACUs* です。Aurora 自体がモニタリングやメンテナンスオペレーションを行うことで、わずかなロードがかかることがあります。そのクラスターでは、データベースを全容量で実行しても *n* x *maximum ACUs* を超える消費はしません。

適切な最小と最大の ACU 値の選択の詳細については、「Aurora クラスターの Aurora Serverless v2 での容量範囲の選択」を参照してください。また、指定した ACU の最小値と最大値は、Aurora Serverless v2 の 一部の Aurora 設定パラメータ動作方法に影響します。容量範囲と設定パラメータ間のやり取りの詳細については、「Aurora Serverless v2 のパラメータグループを使用する」を参照してください。

Aurora Serverless v2 でのスケーリング

Aurora では、各 Aurora Serverless v2 ライターやリーダーの CPU、メモリ、ネットワークなどのリソースの使用率を継続的に追跡しています。これらの測定値を総称してロードと呼びます。ロードには、アプリケーションによって実行されるデータベースオペレーションが含まれます。また、データベースサーバーと Aurora 管理タスクのバックグラウンド処理も含まれます。これらのいずれかによって容量が制約されると、Aurora Serverless v2 がスケールアップします。また、スケールアップすることで解決可能なパフォーマンスの問題を検出した場合、Aurora Serverless v2 もスケールアップします。「Aurora Serverless v2 の重要な Amazon CloudWatch メトリクス」および「パフォーマンスインサイトで Aurora Serverless v2 のパフォーマンスをモニタリングする」の手順で、リソース使用率と Aurora Serverless v2 のスケーリングの影響をモニタリングできます。

ロードは、DB クラスターのライターとリーダーによって異なる場合があります。ライターは、`CREATE TABLE`、`ALTER TABLE`、`DROP TABLE` のようなすべてのデータ定義言語 (DDL) ステートメントを処理します。ライターは、`INSERT` および `UPDATE` のようなすべてのデータ操作言語 (DML) ステートメントも処理します。リーダーは、`SELECT` クエリのような読み取り専用ステートメントを処理できます。

スケーリングは、データベースの Aurora Serverless v2 容量を増減するオペレーションです。Aurora Serverless v2 には、各ライターとリーダーに ACU で測定された独自の現在の容量値を持っています。Aurora Serverless v2 は、現在の容量が小さすぎてロードを処理できない場合に、ライターまたはリーダーをより大きな容量にスケールアップします。ライターまたはリーダーの現在の容量が必要以上に大きい場合、小さい容量にスケールダウンします。

Aurora Serverless v1 では、DB クラスターがしきい値に達するたびに容量が 2 倍にスケーリングされますが、Aurora Serverless v2 では容量を段階的に増やすことができます。ワークロードの需要が、ライターやリーダーの現在のデータベース容量に達し始めると、Aurora Serverless v2 はそのライターまたはリーダーの ACU の数を増やします。Aurora Serverless v2 は、消費されるリソースに対し最適なパフォーマンスを実現するために必要な増分で、容量をスケールアップします。スケーリングは 0.5 ACU という小さい増分で行われます。現在の容量が大きいほど、スケーリングの増分が大きくなり、そのため、スケーリングがより高速になります。

Aurora Serverless v2 のスケーリングは高頻度、詳細、無停止であるため、Aurora Serverless v1 のように AWS Management Console で離散的なイベントが発生することはありません。代わりに、`ServerlessDatabaseCapacity` や `ACUUtilization` のような Amazon CloudWatch メトリクスを測定し、その最小値、最大値、平均値を経時的に追跡できます。Aurora メトリクスの詳細については、「Amazon Aurora クラスターでのメトリクスのモニタリング」を参照してくださ

い。Aurora Serverless v2 のモニタリングに関するヒントについては、「Aurora Serverless v2 の重要な Amazon CloudWatch メトリクス」を参照してください。

リーダースケールは、関連するライターと同時に作成するか、ライターから独立して作成するかを選択できます。これを行うには、そのリーダーの昇格階層を指定します。

- 昇格階層 0 および 1 のリーダーは、ライターと同じタイミングでスケーリングします。このスケーリング動作によって、優先階層 0 および 1 のリーダーは可用性に最適です。これは、フェイルオーバー時にライターからワークロードを引き継ぐために、常に適切な容量に合わせてサイズ調整されているためです。
- 昇格階層 2 ~ 15 のリーダーは、ライターとは独立してスケーリングできます。各リーダーは、クラスターに指定した ACU の最小値と最大値の範囲内に収まります。リーダーが関連するライター DB とは独立してスケーリングすると、ライターが大量のトランザクションを処理し続けている間に、リーダーがアイドル状態になってスケールダウンする場合があります。下位の昇格階層で他のリーダーが利用できない場合でも、フェイルオーバーターゲットとして利用できます。ただし、ライターに昇格した場合は、ライターのワークロード全体を処理するためにスケールアップが必要になる場合があります。

昇格階層の詳細については、「Aurora Serverless v2 リーダーの昇格階層の選択」を参照してください。

Aurora Serverless v1 のスケーリングポイントおよび関連するタイムアウト時間の概念は、Aurora Serverless v2 には適用されません。Aurora Serverless v2 では、データベースの接続中、SQL トランザクション処理中、テーブルロック中、一時テーブルの使用中にスケーリングできます。Aurora Serverless v2 は、スケーリングを開始するためにクワイエットポイントを待ちません。スケーリングによって、進行中のデータベースオペレーションが中断されることはありません。

ワークロードで単一のライターと単一のリーダーで使用できるよりも多くの読み取り容量が必要な場合、クラスターに 複数の Aurora Serverless v2 リーダーを追加できます。各 Aurora Serverless v2 リーダーは、DB クラスターに指定した最小から最大の容量値でスケーリングできます。クラスターのリーダーエンドポイントを使用して、読み取り専用のセッションをリーダーに送信し、ライターのロードを軽減できます。

Aurora Serverless v2 がスケーリングを実行するどうか、また、スケーリング開始後の速度は、クラスターの最小および最大 ACU 設定によって異なります。さらに、リーダーとライターが一緒にスケーリングするように設定されているか、ライターとは独立してスケーリングするように設定されているかによって異なります。Aurora Serverless v2 のスケーリング影響を受ける要因の詳細については、「Aurora Serverless v2 でのパフォーマンスとスケーリング」を参照してください。

> **Note**
>
> 現在、Aurora Serverless v2 ライターとリーダーは ゼロ ACU までスケールダウンしません。アイドル状態の Aurora Serverless v2 ライターとリーダーは、クラスターに指定した最小 ACU 値にスケールダウンできます。
> この動作は、Aurora Serverless v1 が一定期間アイドル状態になると一時停止することがありますが、新しい接続を開いた場合、再開するのに時間がかかります。Aurora Serverless v2 の容量を持つ DB クラスターがしばらく必要ない場合、プロビジョン済み DB クラスターと同様にクラスターを停止および開始できます。クラスターの停止と開始についての詳細は、「Amazon Aurora DB クラスターの停止と開始」を参照してください。

Aurora Serverless v2 と高可用性

Aurora DB クラスターの高可用性を確立する方法として、マルチ AZ DB クラスターにすることがあります。マルチ AZ Aurora DB クラスターは、複数のアベイラビリティーゾーン (AZ) で常に利用可能なコンピューティング性能を持っています。この設定により、大規模な機能停止が発生した場合でも、データベースを稼働し続けます。Aurora は、ライターや AZ 全体に影響を及ぼす問題が発生した場合、自動的にフェイルオーバーを実行します。Aurora Serverless v2 では、ライターの容量と、待機しているコンピューティング性能のスケールアップとスケールダウンを選択できます。これにより、2 番目の AZ のコンピューティング性能は、いつでも現在のワークロードを引き継ぐことができます。同時に、データベースがアイドル状態になると、すべての AZ のコンピューティング性能をスケールダウンすることができます。AWS リージョン での Aurora の動作の仕組みとアベイラビリティーゾーンの詳細については、「Aurora DB インスタンスの高可用性」を参照してください。

Aurora Serverless v2 マルチ AZ 機能は、ライターに加えてリーダーも使用します。リーダーへのサポートは、Aurora Serverless v1 と比較して Aurora Serverless v2 は新しいです。Aurora DB クラスターには、3 つの AZ にわたる最大 15 の Aurora Serverless v2 リーダーを追加できます。

クラスター全体または AWS リージョン全体に影響を及ぼす問題が発生した場合でも、可用性を維持する必要があるビジネスクリティカルなアプリケーションには、Aurora グローバルデータベースを設定できます。災害対策の間、Aurora Serverless v2 容量を使用することで、セカンダリクラスターに引き継ぐことができます。また、データベースがビジー状態でないときは、スケールダウンできます。Aurora グローバルデータベースについての詳細は、「Amazon Aurora Global Database の使用」を参照してください。

Aurora Serverless v2 は、フェイルオーバーなどの高可用性を実現するためにプロビジョン済みのものと同じように動作します。詳細については、「Amazon Aurora の高可用性」を参照してください。

Aurora Serverless v2 クラスターで最大の可用性を確保するとします。ライターに加えてリーダーを作成することもできます。リーダーを昇格階層 0 または 1 に割り当てると、ライターに行われるスケーリングがリーダーにも行われます。そうすれば、フェイルオーバーが発生した場合にも、同じ容量のリーダーがライターを引き継ぐことができます。

クラスターがトランザクションの処理を継続するのと同時に、ビジネスの四半期レポートを実行するとします。Aurora Serverless v2 リーダーをクラスターに追加し、2 ~ 15 の昇格階層に割り当てると、そのリーダーに直接接続してレポートを実行できます。レポートクエリのメモリ負荷と CPU の負荷量に応じて、そのリーダーはワークロードに対応できるようにスケールアップできます。レポートが終了すると、再びスケールダウンできます。

Aurora Serverless v2 とストレージ

各 Aurora DB クラスターのストレージは、3 つの AZ に分散したすべてのデータの 6 つのコピーで構成されます。この組み込みデータレプリケーションは、DB クラスターにライター以外にリーダーが含まれているかどうかにかかわらず適用されます。そうすれば、クラスターのコンピューティング性能に影響を与える問題からも、データを保護できます。

Aurora Serverless v2 ストレージには、「Amazon Aurora ストレージと信頼性」で説明したものと同じように信頼性と耐久性を有しています。これは、Aurora DB クラスターのストレージは、コンピューティング性能によって Aurora Serverless v2 を使用しても、プロビジョン済みのものを使用しても同じように動作するためです。

Aurora クラスターの設定パラメータ

プロビジョン済み DB クラスターとして、Aurora Serverless v2 容量のクラスターと同じクラスターおよびデータベースの設定パラメータをすべて調整できます。ただし、容量に関連する一部のパラメータは、Aurora Serverless v2 については取り扱いが異なります。混合設定クラスターでは、それらの容量に関連するパラメータに指定したパラメータ値は、プロビジョン済みライターおよびリーダーに引き続き適用されます。

ほとんどすべてのパラメータは、Aurora Serverless v2 ライターとリーダーに対して、プロビジョン済みのものと同じように動作します。例外として、Aurora がスケーリング中に自動的に調整する一部のパラメータと、Aurora が最大容量設定に応じて固定値で保持するパラメータがあります。

例えば、バッファキャッシュ用に予約されているメモリ量は、ライターまたはリーダーがスケールアップすると増加し、スケールダウンすると減少します。そうすれば、データベースがビジー状態でないときにメモリを解放できます。逆に、Aurora では、最大容量設定に基づいて、自動的に適切な最大接続数を設定します。そうすれば、ロードが低下して Aurora Serverless v2 がスケールダウンしても、アクティブな接続が切断されません。Aurora Serverless v2 による特定のパラメータの取り扱いについての詳細は、「Aurora Serverless v2 のパラメータグループを使用する」を参照してください。

Aurora Serverless v2 の要件と制限

Aurora Serverless v2 DB インスタンスを使用する予定のクラスターを作成する際には、以下の要件と制限に注意してください。

トピック

- リージョンとバージョンの可用性
- Aurora Serverless v2 を使用するクラスターは、容量範囲を指定する必要があります
- 一部のプロビジョン済み機能は Aurora Serverless v2 でサポートされていません。
- Aurora Serverless v2 での側面は、一部 Aurora Serverless v1 とは異なります。

リージョンとバージョンの可用性

利用できる機能とそのサポートは、各 Aurora データベースエンジンの特定のバージョン、および AWS リージョン によって異なります。Aurora と Aurora Serverless v2 でのバージョンとリージョン可用性については、「Aurora Serverless v2 でサポートされているリージョンと Aurora DB エンジン」を参照してください。

以下の例では、特定の AWS リージョン に Aurora Serverless v2 で使用できる正確な DB エンジンの値を確認するための AWS CLI コマンドを示しています。Aurora Serverless v2 の `--db-instance-class` パラメータは常に `db.serverless` です。`--engine` パラメータには `aurora-mysql` または `aurora-postgresql` が使用できます。適切な `--region` および `--engine` 値を代入して、使用可能な `--engine-version` の値を確認します。コマンドによって何も出力されない場合は、Aurora Serverless v2 は、その AWS リージョン と DB エンジンの組み合わせでは使用できません。

```
aws rds describe-orderable-db-instance-options --engine aurora-mysql --db-instance-
class db.serverless \
```

```
  --region my_region --query 'OrderableDBInstanceOptions[].[EngineVersion]' --output
 text

aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-
instance-class db.serverless \
  --region my_region --query 'OrderableDBInstanceOptions[].[EngineVersion]' --output
 text
```

Aurora Serverless v2 を使用するクラスターは、容量範囲を指定する必要があります

db.serverless DB インスタンスクラスを使用する DB インスタンスを追加する前に、Aurora クラスターが ServerlessV2ScalingConfiguration 属性を持っている必要があります。この属性には容量範囲を指定します。Aurora Serverless v2 容量は、最小 0.5 Aurora 容量単位 (ACU) から 最大 128 ACU までの範囲で、0.5 ACU 単位で指定します。各 ACU は、約 2 ギビバイト (GiB) の RAM と、関連する CPU とネットワーキングに相当します。Aurora Serverless v2 の容量範囲設定の使用方法についての詳細は、「Aurora Serverless v2 の働き」を参照してください。

ACU の最小値と最大値は、クラスターと関連する Aurora Serverless v2 DB インスタンスを作成するときに AWS Management Console で指定できます。AWS CLI の --serverless-v2-scaling-configuration オプションでも指定できます。または、Amazon RDS API で ServerlessV2ScalingConfiguration パラメータを指定します。この属性は、クラスターの作成時、または既存のクラスターの変更時に指定できます。容量範囲を設定する手順については、「Aurora Serverless v2 クラスターの容量設定」を参照してください。最小容量と最大容量値を選択する方法と、それらの設定が一部のデータベースパラメータに与える影響の詳細については、「Aurora クラスターの Aurora Serverless v2 での容量範囲の選択」を参照してください。

一部のプロビジョン済み機能は Aurora Serverless v2 でサポートされていません。

Aurora プロビジョン済み DB インスタンスの以下の機能は、現在 Amazon Aurora Serverless v2 では使用できません。

- データベースアクティビティストリーム (DAS)。
- Aurora PostgreSQL のクラスターキャッシュ管理。apg_ccm_enabled 設定パラメータは Aurora Serverless v2 DB インスタンスには適用されません。

Aurora 機能の一部は Aurora Serverless v2 で動作するものもありますが、容量範囲が特定のワークロードの機能に必要なメモリに必要な容量よりも小さい場合は、問題が発生する可能性があります。その場合、通常よりもデータベースのパフォーマンスが低下するか、メモリ不足のエラーが発生する可能性があります。適切な容量範囲の設定に関する推奨事項については、「Aurora クラスターの Aurora Serverless v2 での容量範囲の選択」を参照してください。容量範囲の設定が間違っているためにデータベースでメモリ不足のエラーが発生した場合のトラブルシューティング情報については、「メモリ不足エラーを回避する」を参照してください。

Aurora Auto Scaling はサポートされていません。これは、CPU の使用率に基づき、新しいリーダーを追加して、読み取り回数の多いワークロードを追加で処理するタイプのスケーリングです。ただし、CPU 使用率に基づくスケーリングは、Aurora Serverless v2 に対しては意味を持ちません。代わりに、事前に Aurora Serverless v2 リーダー DB インスタンスを作成して、小容量にスケールダウンすることもできます。これは、クラスターの読み取り容量をスケーリングするために、新しい DB インスタンスを動的に追加するよりも高速で、中断の少ない方法です。

Aurora Serverless v2 での側面は、一部 Aurora Serverless v1 とは異なります。

Aurora Serverless v1 ユーザーで 初めて Aurora Serverless v2 を使用する場合は、Aurora Serverless v2 と Aurora Serverless v1 の違いを参照して、Aurora Serverless v1 と Aurora Serverless v2 で要件がどのように異なるかを理解してください。

Aurora Serverless v2 を使用する DB クラスターの作成

Aurora Serverless v2 DB インスタンスを追加できる Aurora クラスターを作成するには、Amazon Aurora DB クラスターの作成 と同じ手順に従います。Aurora Serverless v2 では、クラスターはプロビジョン済みクラスターと交換可能です。一部の DB インスタンスで Aurora Serverless v2 を使用し、一部の DB インスタンスではプロビジョン済みクラスターを持てます。

トピック

- Aurora Serverless v2 DB クラスターの設定
- Aurora Serverless v2 DB クラスターの作成
- Aurora Serverless v2 ライター DB インスタンスの作成

Aurora Serverless v2 DB クラスターの設定

クラスターの初期設定が、「Aurora Serverless v2 の要件と制限」に記載されている要件を満たしていることを確認します。次の設定を行い、Aurora Serverless v2 DB インスタンスをクラスターに作成できるようにします。

AWS リージョン

Aurora Serverless v2 DB インスタンスが利用可能な AWS リージョン でクラスターを作成します。利用可能なリージョンの詳細については、「Aurora Serverless v2 でサポートされているリージョンと Aurora DB エンジン」を参照してください。

DB エンジンバージョン

Aurora Serverless v2 と互換性のあるエンジンバージョンを選択します。Aurora Serverless v2 のバージョンの要件の詳細については、「Aurora Serverless v2 の要件と制限」を参照してください。

DB インスタンスクラス

AWS Management Console を使用してクラスターを作成する場合は、ライター DB インスタンスの DB インスタンスクラスを同時に選択します。[Serverless] (サーバーレス) DB インスタンスクラスを選択します。その DB インスタンスクラスを選択した場合は、ライター DB インスタンスの容量範囲も指定します。この容量範囲は、そのクラスターに追加される他のすべての Aurora Serverless v2 DB インスタンスにも適用されます。

DB インスタンスクラスに [サーバーレス] の選択肢がない場合は、Aurora Serverless v2 でサポートされているリージョンと Aurora DB エンジン に対してサポートされている DB エンジンのバージョンを選択していることを確認してください。

AWS CLI または Amazon RDS API を使用する場合、DB インスタンスクラスに指定するパラメータは `db.serverless` です。

容量範囲

クラスター内のすべての DB インスタンスに適用される Aurora 容量単位 (ACU) の最小値と最大値を入力します。このオプションは、DB インスタンスクラスに [Serverless] (サーバーレス) を選択した場合、[Create cluster] (クラスターの作成) と [Add reader] (リーダーの追加) のコンソールページの両方で利用できます。

最小値と最大値の ACU フィールドが表示されない場合は、ライター DB インスタンスに[サーバーレス] インスタンスクラスが選択されていることを確認してください。

最初にプロビジョン済み DB インスタンスでクラスターを作成する場合は、最小と最大の ACU を指定しません。この場合、後でクラスターを変更して、その設定を追加できます。また、Aurora Serverless v2 リーダー DB インスタンスをクラスター追加することができます。そのプロセスの一部として、容量範囲を指定します。

クラスターの容量範囲を指定するまで、AWS CLI または RDS API を使用して、クラスターに Aurora Serverless v2 DB インスタンスを追加することはできません。Aurora Serverless v2 DB インスタンスを追加しようとした場合、エラーが発生します。AWS CLI または RDS API の手順では、容量範囲は `ServerlessV2ScalingConfiguration` 属性で表されます。

複数のリーダー DB インスタンスを含むクラスターの場合は、各 Aurora Serverless v2 リーダー DB インスタンスのフェイルオーバーの優先度によって、その DB インスタンスのスケールアップとスケールダウンに重要な役割を果たします。最初にクラスターを作成するときに、優先度を指定することはできません。2 番目以降のリーダー DB インスタンスをクラスターに追加する場合は、この特性に注意してください。詳細については、「Aurora Serverless v2 リーダーの昇格階層の選択」を参照してください。

Aurora Serverless v2 DB クラスターの作成

AWS Management Console、AWS CLI、または RDS API を使用して Aurora Serverless v2 DB クラスターを作成できます。

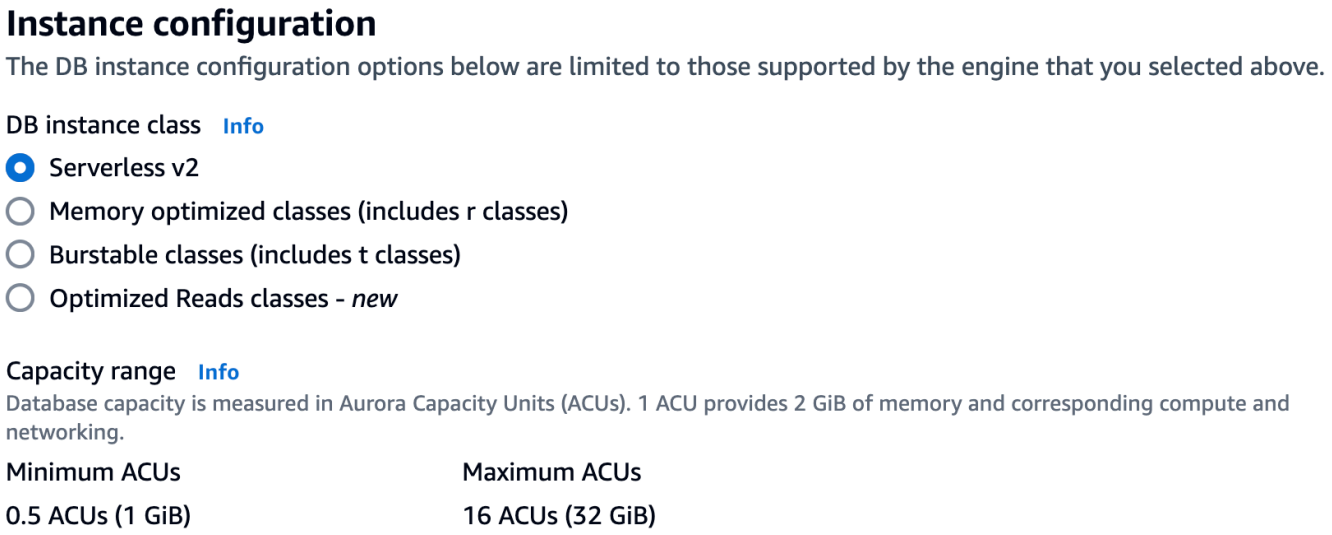
コンソール

Aurora Serverless v2 ライターを使用してクラスターを作成するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. [データベースの作成] を選択します。表示されたページで、以下のオプションを選択します。
   - [エンジンタイプ] として、[Aurora (MySQL 互換)] または [Aurora (PostgreSQL 互換)] を選択します。
   - [バージョン] では、Aurora Serverless v2 でサポートされているリージョンと Aurora DB エンジン でサポートされているバージョンのいずれかを選択します。
4. [DB インスタンスクラス] では、[サーバーレス v2] を選択します。

5. [容量設定] では、デフォルトの範囲をそのまま使用できます。または、最小と最大のキャパシティーユニット数に、それぞれ他の値を指定することもできます。最小 0.5 ACU から 最大 128 ACU まで、0.5 ACU 単位で選択できます。

   Aurora Serverless v2 の容量単位の詳細については、「Aurora Serverless v2 の容量」および「Aurora Serverless v2 でのパフォーマンスとスケーリング」を参照してください。



6. 「Aurora DB クラスターの設定」の説明に従い、他の DB クラスター設定を選択します。
7. [データベースの作成] を選択して、Aurora Serverless v2 DB インスタンスで Aurora DB クラスター (別名: プライマリ DB インスタンス) を作成します。

CLI

AWS CLI を使用して Aurora Serverless v2 と互換性のある DB クラスターを作成するには、「Amazon Aurora DB クラスターの作成」にある CLI の手順に従います。`create-db-cluster` コマンドに次のパラメータを含めます:

- --region *AWS_Region_where_Aurora Serverless v2_instances_are_available*
- --engine-version *serverless_v2_compatible_engine_version*
- --serverless-v2-scaling-configuration MinCapacity=*minimum_capacity*,MaxCapacity=*maximum_capacity*

以下の例は、DB クラスター Aurora Serverless v2 の作成を示しています。

```
aws rds create-db-cluster \
```

```
    --db-cluster-identifier my-serverless-v2-cluster \
    --region eu-central-1 \
    --engine aurora-mysql \
    --engine-version 8.0.mysql_aurora.3.04.1 \
    --serverless-v2-scaling-configuration MinCapacity=1,MaxCapacity=4 \
    --master-username myuser \
    --manage-master-user-password
```

> **Note**
>
> AWS CLI を使用して Aurora Serverless v2 DB クラスターを作成すると、出力のエンジンモードには serverless ではなく provisioned が表示されます。serverless エンジンモードについては、「Aurora Serverless v1」を参照してください。

この例では、マスターユーザーパスワードを生成して Secrets Manager で管理する --manage-master-user-password オプションを指定しています。詳細については、「Amazon Aurora および AWS Secrets Manager によるパスワード管理」を参照してください。または、--master-password オプションを使用して、自分でパスワードを指定して管理することもできます。

Aurora Serverless v2 のバージョンの要件の詳細については、「Aurora Serverless v2 の要件と制限」を参照してください。容量範囲に許可される番号と、数字の意味については、「Aurora Serverless v2 の容量」および「Aurora Serverless v2 でのパフォーマンスとスケーリング」を参照してください。

既存のクラスターに容量設定が指定されているかどうかを確認するには、ServerlessV2ScalingConfiguration 属性 を describe-db-clusters コマンドの出力でチェックします。この属性は、以下の例のようになります。

```
"ServerlessV2ScalingConfiguration": {
    "MinCapacity": 1.5,
    "MaxCapacity": 24.0
}
```

> **Tip**
>
> クラスターの作成時に最小および最大 ACU を指定しない場合は、後で modify-db-cluster コマンドを使用してその設定を追加できます。これを行うまでは、Aurora

Serverless v2 DB インスタンスをクラスターに追加できません。db.serverless DB インスタンスを追加しようとした場合、エラーが発生します。

API

RDS API を使用して Aurora Serverless v2 DB インスタンスと互換性のある DB クラスターを作成するには、「Amazon Aurora DB クラスターの作成」にある API の手順に従います。以下の設定を選択します。CreateDBCluster オペレーションに以下のパラメータが含まれていることを確認してください。

```
EngineVersion serverless_v2_compatible_engine_version
ServerlessV2ScalingConfiguration with MinCapacity=minimum_capacity and
 MaxCapacity=maximum_capacity
```

Aurora Serverless v2 のバージョンの要件の詳細については、「Aurora Serverless v2 の要件と制限」を参照してください。容量範囲に許可される番号と、数字の意味については、「Aurora Serverless v2 の容量」および「Aurora Serverless v2 でのパフォーマンスとスケーリング」を参照してください。

既存のクラスターに容量設定が指定されているかどうかを確認するには、ServerlessV2ScalingConfiguration 属性 を DescribeDBClusters オペレーションの出力でチェックします。この属性は、以下の例のようになります。

```
"ServerlessV2ScalingConfiguration": {
    "MinCapacity": 1.5,
    "MaxCapacity": 24.0
}
```

Tip

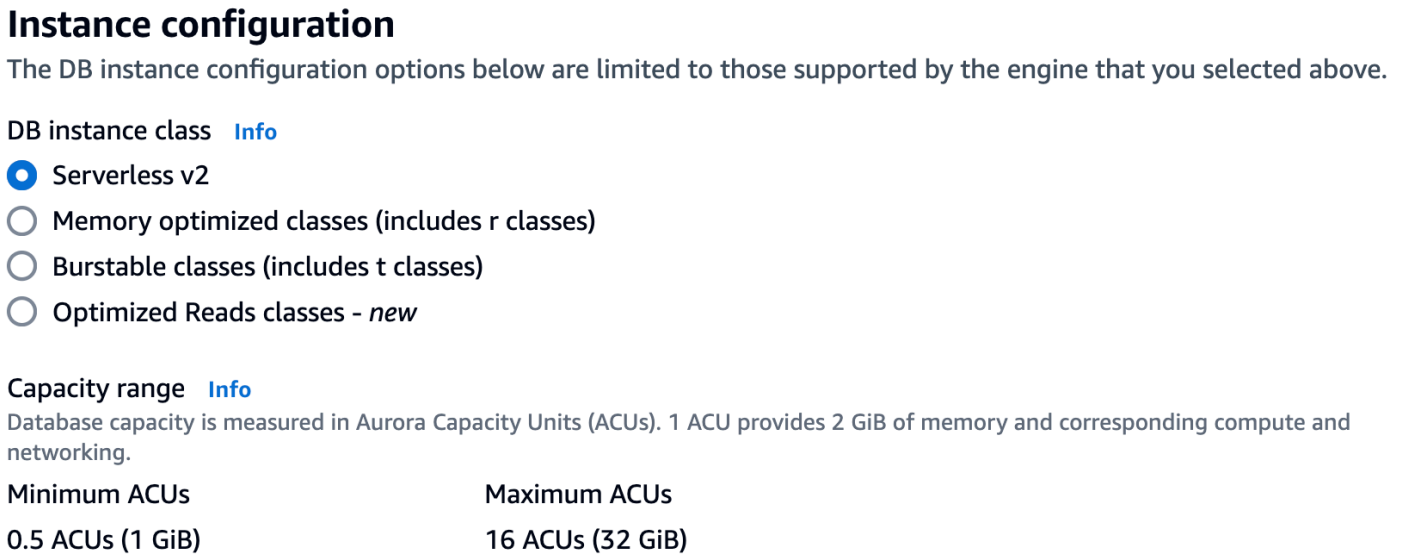
クラスターの作成時に最小および最大 ACU を指定しない場合は、後で ModifyDBCluster オペレーションを使用してその設定を追加できます。これを行うまでは、Aurora Serverless v2 DB インスタンスをクラスターに追加できません。db.serverless DB インスタンスを追加しようとした場合、エラーが発生します。

Aurora Serverless v2 ライター DB インスタンスの作成

コンソール

AWS Management Console を使用してDB クラスターを作成する場合は、ライター DB インスタンスのプロパティを同時に指定します。Aurora Serverless v2 を使用してライター DB インスタンスを作成するには、サーバーレス DB インスタンスクラスを選択します。

次に、Aurora 容量単位 (ACU) の最小値と最大値を指定し、クラスターの容量範囲を設定します。クラスター内の各 Aurora Serverless v2 DB インスタンスには、これらの最小値と最大値が適用されます。



クラスターを最初に作成するときに Aurora Serverless v2 DB インスタンスを作成しない場合、後で 1 つまたは複数の Aurora Serverless v2 DB インスタンスを追加できます。そのためには、「Aurora Serverless v2 リーダーの追加」および「プロビジョン済みライターまたはリーダーを Aurora Serverless v2 に変換する」の手順に従います。容量範囲は、クラスターに最初の Aurora Serverless v2 DB インスタンス容量を追加したときに指定します。「Aurora Serverless v2 クラスターの容量設定」の手順に従って、容量範囲を後で変更できます。

CLI

AWS CLI を使用して DB クラスター Aurora Serverless v2 を作成するとき、create-db-instance コマンドを使用して書き込み DB インスタンスを明示的に追加します。次のパラメータを含めます。

- `--db-instance-class db.serverless`

以下の例は、書き込み DB インスタンス Aurora Serverless v2 の作成を示しています。

```
aws rds create-db-instance \
    --db-cluster-identifier my-serverless-v2-cluster \
    --db-instance-identifier my-serverless-v2-instance \
    --db-instance-class db.serverless \
    --engine aurora-mysql
```

Aurora Serverless v2 DB クラスターの管理

Aurora Serverless v2 では、クラスターはプロビジョン済みクラスターと交換可能です。Aurora Serverless v2 プロパティは、クラスター内の 1 つまたは複数の DB インスタンスに適用されます。したがって、クラスターの作成、クラスターの変更、スナップショットの作成と復元などの手順は、他の種類の Aurora クラスターと基本的に同じです。Aurora クラスターと DB インスタンスを管理するための一般的な手順については、「Amazon Aurora DB クラスターの管理」を参照してください。

次のトピックでは、Aurora Serverless v2 DB インスタンスを含むクラスターの管理に関する考慮事項について説明します。

トピック

- Aurora Serverless v2 クラスターの容量設定
- Aurora Serverless v2 の容量範囲の確認
- Aurora Serverless v2 リーダーの追加
- プロビジョン済みライターまたはリーダーを Aurora Serverless v2 に変換する
- Aurora Serverless v2 ライターまたはリーダーをプロビジョン済みに変換する
- Aurora Serverless v2 リーダーの昇格階層の選択
- Aurora Serverless v2 での TLS/SSL の使用
- Aurora Serverless v2 ライターとライターの表示
- Aurora Serverless v2 のログ記録

Aurora Serverless v2 クラスターの容量設定

Aurora Serverless v2 DB インスタンスを含むクラスターの設定パラメータやその他の設定、または DB インスタンス自体を変更するには、プロビジョン済みクラスターの場合と同様の一般的な手順に従います。詳細については、「Amazon Aurora DB クラスターの変更」を参照してください。

Aurora Serverless v2 に固有の最も重要な設定は容量範囲です。Aurora クラスターの Aurora 容量単位 (ACU) の最小値と最大値を設定した後は、クラスター内の Aurora Serverless v2 DB インスタンスを積極的に調整する必要はありません。それは Aurora が行います。この設定は、クラスターレベルで管理されます。クラスター内の各 Aurora Serverless v2 DB インスタンスには、同一の最小 ACU と最大 ACU が適用されます。

以下の特定の値を設定できます。

- [Minimum ACUs] (最小 ACU) – Aurora Serverless v2 DB インスタンスは、この ACU の数まで容量を減らすことができます。
- [Maximum ACUs] (最大 ACU) – Aurora Serverless v2 DB インスタンスは、この ACU の数まで容量を増やすことができます。

> **Note**
>
> Aurora Serverless v2 DB クラスターの容量範囲を変更すると、すぐに適用するか、次回の定期メンテナンス期間中に適用するかに関係なく、変更はすぐに反映されます。

容量範囲の影響と、容量範囲をモニタリング、微調整する方法については、「Aurora Serverless v2 の重要な Amazon CloudWatch メトリクス」および「Aurora Serverless v2 でのパフォーマンスとスケーリング」を参照してください。目標は、ワークロードの急増時にはクラスターの容量を最大化し、クラスターがビジー状態でない場合は、コストを最小限に抑えるために容量を最小化することです。

モニタリングに基づき、クラスターの ACU の範囲を今よりも大きく、小さく、広く、狭くする必要があると判断したとします。AWS Management Console、AWS CLI、Amazon RDS により、Aurora クラスターの容量を ACU の特定の範囲に設定できます。この容量範囲は、クラスター内のすべての Aurora Serverless v2 DB インスタンスに適用されます。

例えば、クラスターの容量範囲が 1 ~ 16 ACU で、2 つの Aurora Serverless v2 DB インスタンスが含まれているとします。このとき、クラスター全体で 2 ACU (アイドル時) から 32 ACU (フル使用時) の間のどこかで消費されます。容量範囲を 8 から 20.5 ACU に変更すると、クラスターではアイドル時に 16 ACU を消費し、フル使用時には最大 41 ACU を消費します。

Aurora は Aurora Serverless v2 DB インスタンスの特定のパラメータを、容量範囲内の最大 ACU 値によって決定される値に自動的に設定します。このようなパラメータのリストについては、「Aurora Serverless v2 の最大接続数」を参照してください。このタイプの計算によって決定される静的パラ

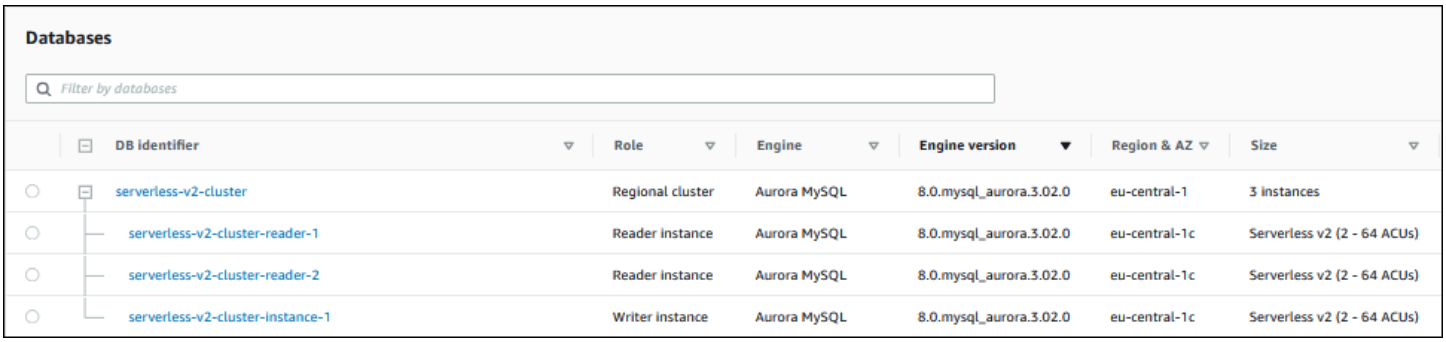
メータの場合は、DB インスタンスを再起動すると、値が再評価されます。したがって、容量範囲を変更した後に DB インスタンスを再起動すれば、このようなパラメータの値を更新できます。このようなパラメータの変更を反映するために、DB インスタンスの再起動が必要かどうかを確認するには、DB インスタンスの `ParameterApplyStatus` 属性をチェックしてください。`pending-reboot` の値は、再起動によって一部のパラメータ値に変更が適用されることを示しています。

コンソール

AWS Management Console を使用することで、Aurora Serverless v2 DB インスタインスを含むクラスターの容量範囲を設定できます。

コンソールを使用する場合、最初の Aurora Serverless v2 DB インスタンスをクラスターに追加した時点で、そのクラスターに容量範囲を設定します。このような設定は、クラスター作成時、ライター DB インスタンスに Serverless v2 DB インスタンスクラスを選択した場合に行う場合があります。または、クラスターに Aurora Serverless v2 リーダー DB インスタンスを追加する場合、サーバーレス DB インスタンスクラスを選択するときに行うこともあります。さらに、クラスター内の既存のプロビジョン済み DB インスタンスをサーバーレス DB インスタンスクラスに変換するときに行うこともあります。これらの手順の完全版については、「Aurora Serverless v2 ライター DB インスタンスの作成」、「Aurora Serverless v2 リーダーの追加」および「プロビジョン済みライターまたはリーダーを Aurora Serverless v2 に変換する」を参照してください。

クラスターレベルで設定した容量範囲は、クラスター内のすべての Aurora Serverless v2 DB インスタンスに適用されます。以下のイメージは、複数の Aurora Serverless v2 リーダー DB インスタンスによるクラスターを示しています。それぞれ、2 ~ 64 ACU の同じ容量範囲を持っています。



Aurora Serverless v2 クラスターの容量範囲を変更するには

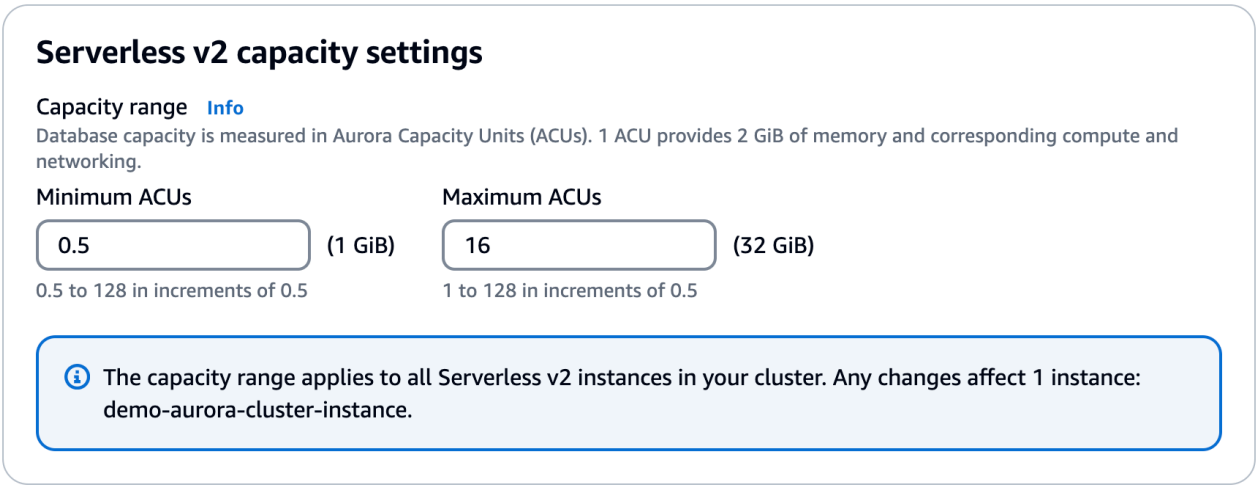
1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. リストから、Aurora Serverless v2 DB インスタンスを含むクラスターを選択します。クラスターには、少なくとも 1 つの Aurora Serverless v2 DB インスタンスが含まれていなければな

りません。それ以外の場合、Aurora では [Capacity range] (容量範囲) セクションを表示しません。

4. [アクション]、[変更] の順に選択します。
5. [Capacity range] (容量範囲) セクションで、以下のものを選択します。

   a. [Minimum ACUs] (最小 ACU) の値を入力します。コンソールには、許容される値の範囲が表示されます。最小容量は 0.5 ~ 128 ACU から選択できます。最大容量は 1 ~ 128 ACU から選択できます。容量値は 0.5 ACU 単位で調整できます。

   b. [Maximum ACUs] (最大 ACU) の値を入力します。この値は、[Minimum ACUs] (最小 ACU) 以上にする必要があります。コンソールには、許容される値の範囲が表示されます。以下の図は、その選択肢を示しています。



6. Continue (続行) をクリックします。[変更の概要] ページが表示されます。
7. [Apply immediately] (すぐに適用) を選択します。

   容量の変更は、すぐに適用するよう選択したか、次回の定期メンテナンス期間中に適用するよう選択したかに関係なく、すぐに反映されます。
8. [クラスターの変更] をクリックして、変更の概要の内容を受け入れます。[戻る] をクリックして設定を修正したり、[キャンセル] をクリックして変更を破棄することもできます。

AWS CLI

AWS CLI を使用して Aurora Serverless v2 DB インスタンスを使用する予定のクラスターの容量を設定するには、[modify-db-cluster] AWS CLI コマンドを実行します。`--serverless-v2-scaling-configuration` オプションを指定します。クラスターには、既に 1 つまたは複数の Aurora Serverless v2 DB インスタンスが含まれている場合があります。または DB インスタンスを

後で追加することもできます。MinCapacity および MaxCapacity フィールドに有効な値は以下のとおりです。

- 0.5、1、1.5、2 など、0.5 のステップで、最大 128 までです。

この例では、sample-cluster という名前の Aurora DB クラスターの ACU 範囲を、最小 48.5、最大 64 に設定します。

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster \
--serverless-v2-scaling-configuration MinCapacity=48.5,MaxCapacity=64
```

容量の変更は、すぐに適用するよう選択したか、次回の定期メンテナンス期間中に適用するよう選択したかに関係なく、すぐに反映されます。

設定後は、Aurora Serverless v2 DB インスタンスをクラスターに追加でき、それぞれの新しい DB インスタンスは 48.5 から 64 ACU の間でスケーリングできます。また、新しい容量範囲は、クラスター内に既に存在していた Aurora Serverless v2 DB インスタンスに適用されます。DB インスタンスは、新しい容量範囲内に収まるように、必要に応じてスケールアップまたはスケールダウンします。

CLI を使用して容量範囲を設定するその他の例については、「Aurora クラスターの Aurora Serverless v2 での容量範囲の選択」を参照してください。

Aurora Serverless を使用して AWS CLI DB クラスターのスケーリング設定を変更するには、AWS CLI の modify-db-cluster コマンドを実行します。--serverless-v2-scaling-configuration オプションを指定すると、最小容量および最大容量を設定できます。有効な容量値には次のようなものがあります。

- Aurora MySQL: 0.5、1、1.5、2 など、0.5 ACU 単位で最大 128 までです。
- Aurora PostgreSQL: 0.5、1、1.5、2 など、0.5 ACU 単位で最大 128 までです。

次の例では、sample-cluster という名前のクラスターの一部である sample-instance という名前の Aurora Serverless v2 DB インスタンスのスケーリングの設定を変更します。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster \
--serverless-v2-scaling-configuration MinCapacity=8,MaxCapacity=64
```

Windows の場合:

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster ^
--serverless-v2-scaling-configuration MinCapacity=8,MaxCapacity=64
```

RDS API

ModifyDBCluster API オペレーションを使用して、Aurora DB インスタンスの容量を設定できます。`ServerlessV2ScalingConfiguration` パラメータを指定します。`MinCapacity` および `MaxCapacity` フィールドに有効な値は以下のとおりです。

- `0.5`、`1`、`1.5`、`2` など、0.5 のステップで、最大 128 までです。

ModifyDBCluster API オペレーションを使用して Aurora Serverless v2 DB インスタンスを含むクラスターのスケーリング設定を変更できます。`ServerlessV2ScalingConfiguration` パラメータを指定すると、最小容量および最大容量を設定できます。有効な容量値には次のようなものがあります。

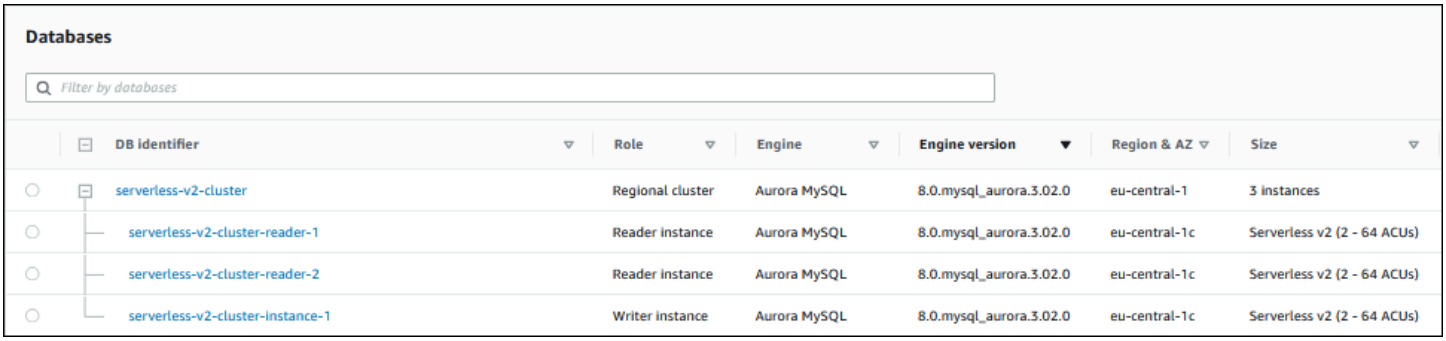
- Aurora MySQL: `0.5`、`1`、`1.5`、`2` など、0.5 ACU 単位で最大 128 までです。
- Aurora PostgreSQL: `0.5`、`1`、`1.5`、`2` など、0.5 ACU 単位で最大 128 までです。

容量の変更は、すぐに適用するよう選択したか、次回の定期メンテナンス期間中に適用するよう選択したかに関係なく、すぐに反映されます。

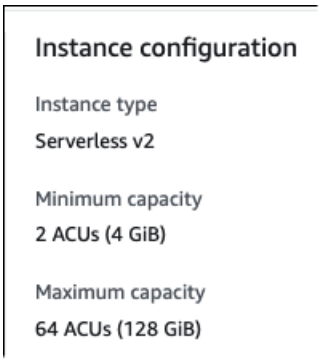
Aurora Serverless v2 の容量範囲の確認

Aurora Serverless v2 クラスターの容量範囲を確認する手順では、最初に容量範囲を設定する必要があります。まだ完了していない場合は、「Aurora Serverless v2 クラスターの容量設定」の手順に従ってください。

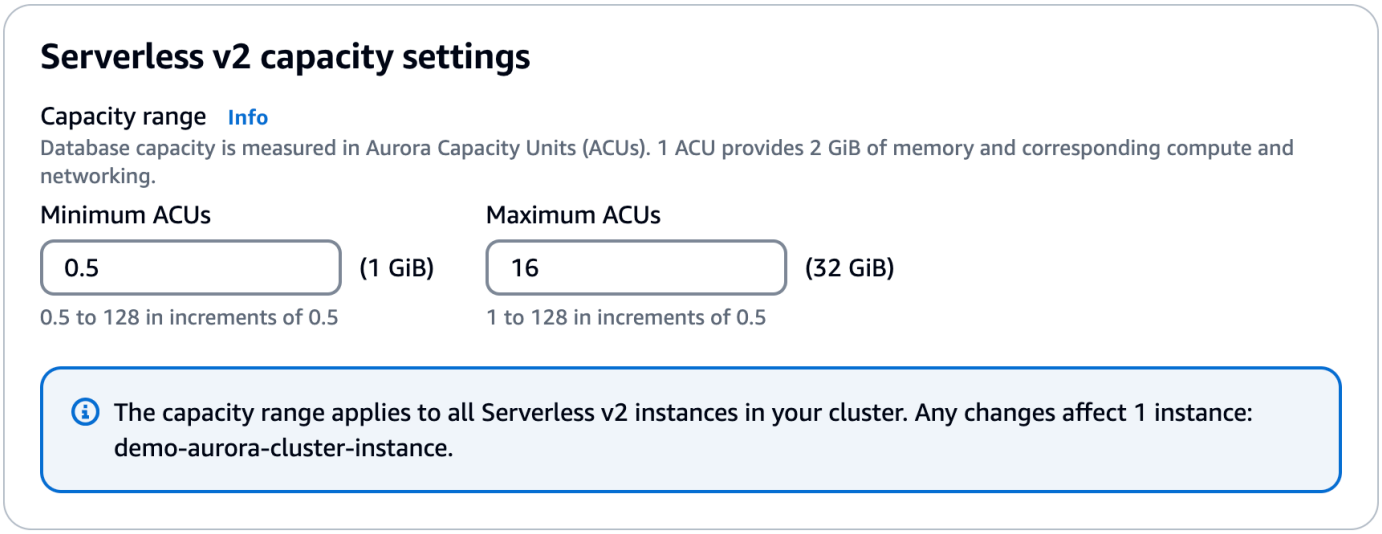
クラスターレベルで設定した容量範囲は、クラスター内のすべての Aurora Serverless v2 DB インスタンスに適用されます。以下のイメージは、複数の Aurora Serverless v2 DB インスタンスによるクラスターを示しています。それぞれ、同じ容量範囲を持っています。



クラスター内の Aurora Serverless v2 DB インスタンスの詳細ページを表示することもできます。DB インスタンスの容量範囲は、[Configuration] (設定) タブに表示されます。



また、クラスターの現在の容量範囲は、クラスターの [Modify] (変更) ページで確認できます。以下の図では、その方法を表しています。この時点で、容量範囲を変更できます。容量範囲の設定または変更が可能なすべての方法については、「Aurora Serverless v2 クラスターの容量設定」を参照してください。



Aurora クラスターの現在の容量範囲を確認する

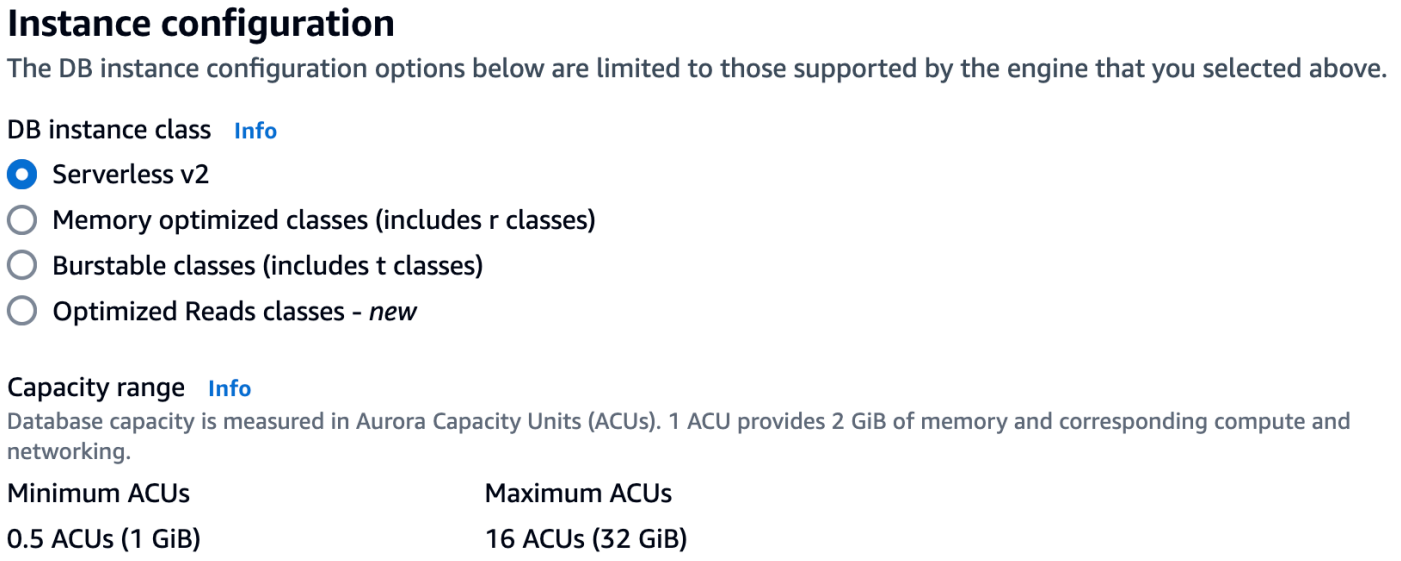
クラスターの `ServerlessV2ScalingConfiguration` 属性を調べることで、クラスター内の Aurora Serverless v2 DB インスタンスに設定されている容量範囲を確認できます。以下の AWS CLI の例では、最小容量が 0.5 Aurora 容量単位 (ACU)、最大容量 16 ACU のクラスターを示しています。

```
$ aws rds describe-db-clusters --db-cluster-identifier serverless-v2-64-acu-cluster \
  --query 'DBClusters[*].[ServerlessV2ScalingConfiguration]'
[
    [
        {
            "MinCapacity": 0.5,
            "MaxCapacity": 16.0
        }
    ]
]
```

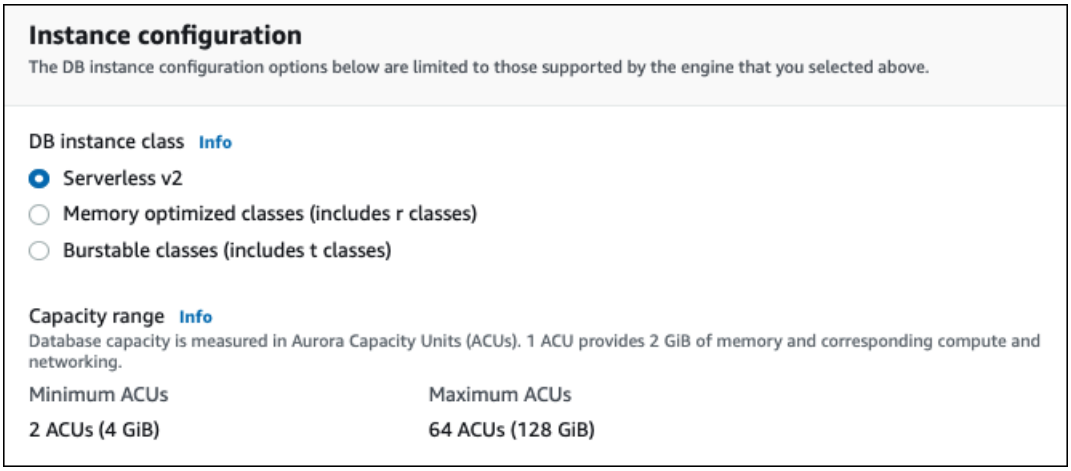
Aurora Serverless v2 リーダーの追加

Aurora Serverless v2 リーダー DB インスタンスをクラスターに追加するには、「DB クラスターに Aurora レプリカを追加する」と同じ手順に従います。新しい DB インスタンスに Serverless v2 インスタンスクラスを選択する

クラスター内のリーダー DB インスタンスが最初の Aurora Serverless v2 DB インスタンスの場合は、容量範囲も選択します。以下のイメージは、最小および最大の Aurora 容量単位 (ACU) を指定するために使用するコントロールを示しています。この設定は、このリーダー DB インスタンスと、クラスターに追加する他の Aurora Serverless v2 DB インスタンスに適用されます。各 Aurora Serverless v2 DB インスタンスは、最小から最大の ACU 値の間でスケーリングできます。



クラスターに既にいずれかの Aurora Serverless v2 DB インスタンスを追加している場合、別の Aurora Serverless v2 リーダー DB インスタンスには、現在の容量範囲が表示されます。以下のイメージは、これらの読み取り専用のコントロールを示しています。



クラスターの容量範囲を変更する場合は、「Aurora Serverless v2 クラスターの容量設定」の手順に従ってください。

複数のリーダー DB インスタンスを含むクラスターの場合は、各 Aurora Serverless v2 リーダー DB インスタンスのフェイルオーバーの優先度によって、その DB インスタンスのスケールアップとスケールダウンに重要な役割を果たします。最初にクラスターを作成するときに、優先度を指定することはできません。2 番目以降のリーダー DB インスタンスをクラスターに追加する場合は、この特性に注意してください。詳細については、「Aurora Serverless v2 リーダーの昇格階層の選択」を参照してください。

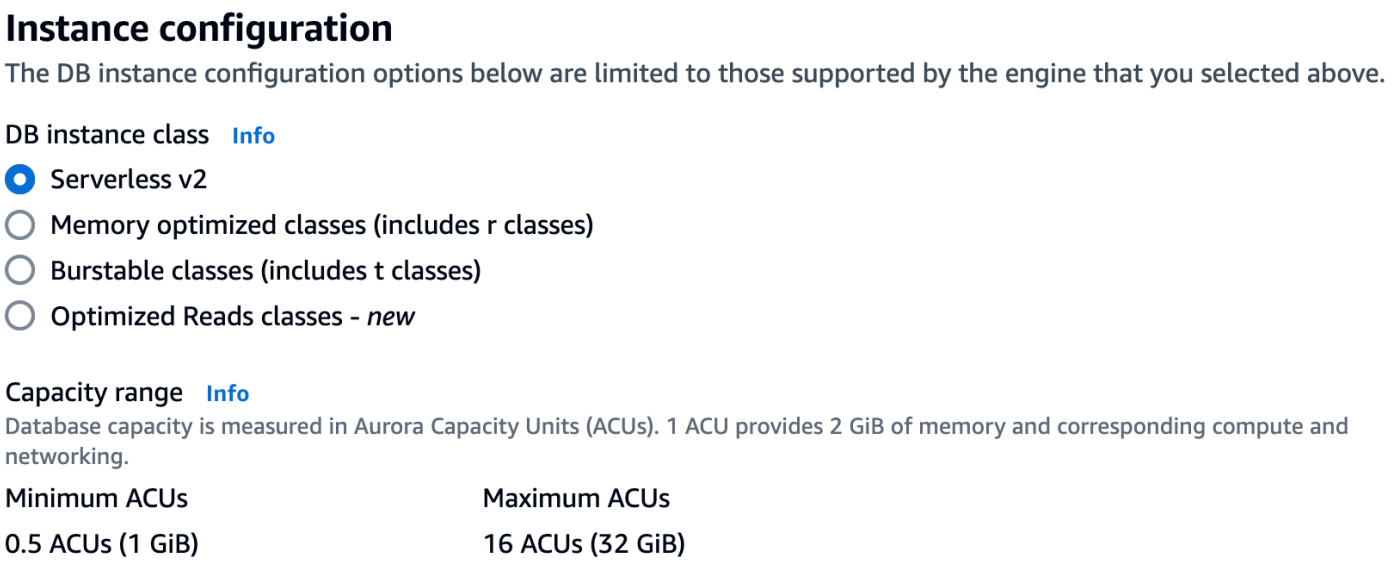
クラスターの現在の容量範囲を確認できるその他の方法については、「Aurora Serverless v2 の容量範囲の確認」を参照してください。

プロビジョン済みライターまたはリーダーを Aurora Serverless v2 に変換する

プロビジョン済み DB インスタンスを変換することで Aurora Serverless v2 を使用できます。これを行うには、「DB クラスター内の DB インスタンスの変更」の手順に従います。このクラスターは、「Aurora Serverless v2 の要件と制限」の要件を満たす必要があります。例えば Aurora Serverless v2 DB インスタンスでは、クラスターが特定の最小エンジンバージョンを実行している必要があります。

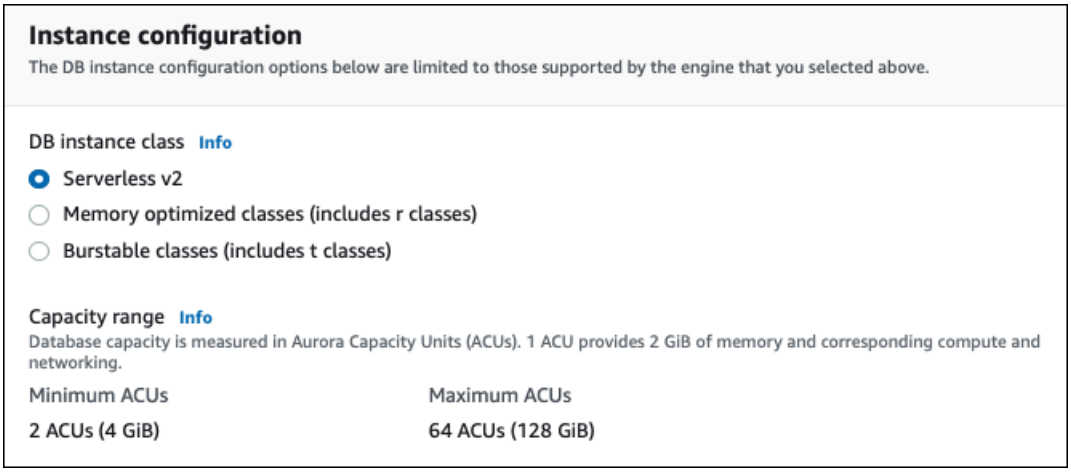
実行中のプロビジョン済みクラスターを変換して、Aurora Serverless v2 を利用するとします。その場合、切り替え処理の最初のステップとして、DB インスタンスを Aurora Serverless v2 に変換することで、ダウンタウンを最小限に抑えることができます。完全な手順については、「プロビジョニングされたクラスターから Aurora Serverless v2 への切り替え」を参照してください。

クラスター内で変換する DB インスタンスが最初の Aurora Serverless v2 DB インスタンスである場合、変更オペレーションの一部として、クラスターの容量範囲を選択します。この容量範囲は、クラスターに追加する各 Aurora Serverless v2 DB インスタンスに適用されます。以下のイメージは、最小および最大の Aurora 容量単位 (ACU) を指定するページを示しています。



容量範囲の重要性についての詳細は、「Aurora Serverless v2 の容量」を参照してください。

クラスターに既に 1 つまたは複数の Aurora Serverless v2 DB インスタンスが含まれている場合、変更オペレーションの際に既存の容量範囲が表示されます。以下のイメージは、その情報パネルの例を示しています。



Aurora Serverless v2 DB インスタンスを追加した後でクラスターの容量範囲を変更する場合は、「Aurora Serverless v2 クラスターの容量設定」の手順に従ってください。

Aurora Serverless v2 ライターまたはリーダーをプロビジョン済みに変換する

Aurora Serverless v2 DB インスタンスをプロビジョン済み DB インスタンスに変換できます。これを行うには、「DB クラスター内の DB インスタンスの変更」の手順に従います。[Serverless] (サーバーレス) 以外の DB インスタンスクラスを選択します。

Aurora Serverless v2 DB インスタンスが Aurora Serverless v2 DB インスタンスの最大 ACU (Aurora 容量単位) より大きな容量が必要な場合、プロビジョン済みに変換する場合があります。例えば、最大の db.r5 および db.r6g DB インスタンスクラスは、Aurora Serverless v2 DB インスタンスがスケーラブルな容量よりも大きなメモリ容量を持っています。

> **Tip**
>
> db.r3 や db.t2 などの古い DB インスタンスクラスには、Aurora Serverless v2 で使用する Aurora バージョンでは使用できないものもあります。Aurora Serverless v2 DB インスタンスをプロビジョン済みに変換する際に使用可能な DB インスタンスは、「DB インスタンスクラスでサポートされている DB エンジン」を参照してください。

クラスターのライター DB インスタンスを Aurora Serverless v2 から プロビジョン済みに変換するには、逆順で「プロビジョニングされたクラスターから Aurora Serverless v2 への切り替え」の手順に従います。クラスター内のリーダー DB インスタンスの 1 つを Aurora Serverless v2 からプロビジョン済みに切り替えます。次に、フェイルオーバーを実行して、プロビジョン済み DB インスタンスをライターにします。

すべての Aurora Serverless v2 DB インスタンスをクラスターから削除した場合でも、以前にクラスターに指定した容量範囲はそのまま維持されます。容量範囲を変更する場合は、「Aurora Serverless v2 クラスターの容量設定」の説明に従って、クラスターを変更できます。

Aurora Serverless v2 リーダーの昇格階層の選択

複数の Aurora Serverless v2 DB インスタンスを含むクラスター、またはプロビジョン済み Aurora Serverless v2 DB インスタンスが混在するクラスターでは、各 Aurora Serverless v2 DB インスタンスの昇格階層の設定に注意してください。この設定は、プロビジョン済み Aurora Serverless v2 DB インスタンスと比較して、DB インスタンスに対してより多くの動作をコントロールします。

AWS Management Console の [Create database] (データベースの作成)、[Modify instance] (インスタンスの変更)、[Add reader] (リーダーの追加) ページの [Additional configuration] (追加設定) で、[Failover priority] (フェイルオーバーの優先度) の選択肢を使用して、この設定を指定します。既存の DB インスタンスのこのプロパティについては、[Databases] (データベース) ページのオプションの [Priority tier] (優先階層) 列に表示されます。このプロパティは、DB クラスターまたは DB インスタンスの詳細ページにも表示されます。

プロビジョン済み DB インスタンスの場合、階層 0 ~ 15 の選択によって、フェイルオーバーオペレーション中に Aurora がどのリーダー DB インスタンスをライターに昇格させるかを選択する順序のみを決定します。Aurora Serverless v2 リーダー DB インスタンスの場合、ライター DB インスタンスの容量に合わせてスケールアップするか、インスタンスそのもののワークロードに応じて独自にスケールアップするかどうかは、階層番号で決まります。階層 0 または 1 の Aurora Serverless v2 リーダー DB インスタンスは、最低でもライター DB インスタンスと同じ容量に維持されます。これにより、フェイルオーバー発生時にライター DB インスタンスから引き継ぐ準備が整います。ライター DB インスタンスがプロビジョン済み DB インスタンスの場合、Aurora では同等の Aurora Serverless v2 容量を推定します。この推定値を Aurora Serverless v2リーダー DB インスタンスの最小容量として使用します。

階層 2 ~ 15 の Aurora Serverless v2 リーダー DB インスタンスは、最小容量に対する同じ制約はありません。アイドル状態の場合、クラスターの容量範囲で指定された Aurora 容量単位 (ACU) の最小値までスケールダウンできます。

以下の Linux AWS CLI の例は、クラスター内のすべての DB インスタンスの昇格階層を調べる方法を示しています。最後のフィールドには、ライター DB インスタンスの場合は `True`、すべてのリーダー DB インスタンスの場合は `False` の値が含まれています。

```
$ aws rds describe-db-clusters --db-cluster-identifier promotion-tier-demo \
  --query 'DBClusters[*].DBClusterMembers[*].
[PromotionTier,DBInstanceIdentifier,IsClusterWriter]' \
  --output text

1   instance-192  True
1   tier-01-4840  False
10  tier-10-7425  False
15  tier-15-6694  False
```

以下の Linux AWS CLI の例は、クラスター内の特定の DB インスタンスの昇格階層を変更する方法を示しています。このコマンドでは、まず新しい昇格階層で DB インスタンスを変更します。その後、DB インスタンスが再び利用可能になるのを待って、DB インスタンスの新しい昇格階層を確認します。

```
$ aws rds modify-db-instance --db-instance-identifier instance-192 --promotion-tier 0
$ aws rds wait db-instance-available --db-instance-identifier instance-192
$ aws rds describe-db-instances --db-instance-identifier instance-192 \
  --query '*[].[PromotionTier]' --output text
0
```

さまざまなユースケースで昇格階層を指定する方法の詳細については、「Aurora Serverless v2 でのスケーリング」を参照してください。

Aurora Serverless v2 での TLS/SSL の使用

Aurora Serverless v2 では、クライアントと Aurora Serverless v2 DB インスタンス間の通信に対し、Transport Layer Security/Secure Sockets Layer (TLS/SSL) プロトコルを使用して暗号化することができます。サポートされる TLS/SSL バージョンは、1.0、1.1、および 1.2 です。Aurora での TLS/SSL の使用に関する一般的な情報については、「Aurora MySQL DB クラスターでの TLS の使用」を参照してください。

MySQL クライアントを使用しての Aurora MySQL データベースへの接続の詳細については、「MySQL データベースエンジンを実行している DB インスタンスへの接続」を参照してください。

Aurora Serverless v2 では、MySQL クライアント (`mysql`) および PostgreSQL クライアント (`psql`) で使用できるすべての TLS/SSL モードがサポートされます。これらには、以下の表に示すものも含まれます。

TLS/SSL モードの説明	mysql	psql
TLS/SSL を使用せずに接続します。	DISABLED	無効化
初期に TLS/SSL を使用して接続を試みますが、必要に応じて非 SSL にフォールバックします。	PREFERRED	優先 (デフォルト)
強制的に TLS/SSL を使用します。	REQUIRED	require
TLS/SSL を義務化し、認証機関 (CA) を検証します。	VERIFY_CA	verify-ca
TLS/SSL を強制的に使用し CA を確認し、また CA ホスト名を確認します。	VERIFY_IDENTITY	verify-full

Aurora Serverless v2 は、ワイルドカード証明書を使用します。TLS/SSL を使用するときに "verify CA" または "verify CA and CA hostname" オプションを指定した場合は、まず Amazon ルート CA 1 信頼ストアを Amazon Trust Services からダウンロードしてください。ダウンロードした、この PEM 形式のファイルは、クライアントコマンドにより識別できます。これを PostgreSQL クライアントを使用して行うには、以下のように実行します。

Linux、macOS、Unix の場合:

```
psql 'host=endpoint user=user sslmode=require sslrootcert=amazon-root-CA-1.pem
 dbname=db-name'
```

Postgres クライアントを使用した Aurora PostgreSQL データベースの動作の詳細については、「PostgreSQL データベースエンジンを実行する DB インスタンスへの接続」を参照してください。

Aurora DB クラスターへの接続全般の詳細については、「Amazon Aurora DB クラスターへの接続」を参照してください。

Aurora Serverless v2 DB クラスターへの接続用にサポートされている暗号スイート

設定可能な暗号スイートを使用すると、データベース接続のセキュリティをより詳細に制御できます。データベースへのクライアント TLS/SSL 接続を保護するために許可する暗号スイートのリストを指定できます。設定可能な暗号スイートを使用すると、データベースサーバーが受け入れる接続暗号化を制御できます。これにより、安全でない暗号や使用されなくなった暗号の使用を防ぐことができます。

Aurora MySQL に基づく Aurora Serverless v2 DB クラスターは、Aurora MySQL プロビジョン済み DB クラスターと同じ暗号スイートをサポートします。これらの暗号スイートについては、「Aurora PostgreSQL DB クラスターへの接続用暗号スイートを設定する」を参照してください。

Aurora PostgreSQL に基づく DB クラスターは、Aurora PostgreSQL プロビジョンド DB クラスターと同じ暗号スイートをサポートします。これらの暗号スイートについては、「Aurora PostgreSQL DB クラスターへの接続用暗号スイートを設定する」を参照してください。

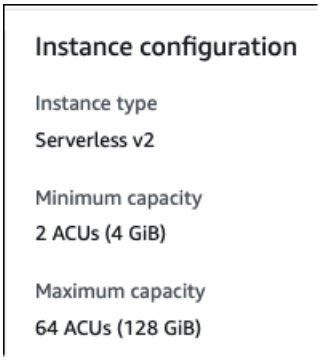
Aurora Serverless v2 ライターとライターの表示

プロビジョン済み DB インスタンスと同じ方法で、Aurora Serverless v2 DB インスタンスの詳細を表示できます。これを行うには、「Amazon Aurora DB クラスターの表示」の一般的な手順に従います。クラスターには、すべてまたは一部の Aurora Serverless v2 DB インスタンス、プロビジョン済み DB インスタンスが含まれる場合があります。

1 つまたは複数の Aurora Serverless v2 DB インスタンスを作成した場合、どの DB インスタンスがサーバーレスタイプで、どの DB インスタンスがインスタンスタイプであるかを確認できます。また、Aurora Serverless v2 DB インスタンスが使用可能な最小および最大の Aurora 容量単位 (ACU) を表示できます。各 ACU は、処理 (CPU) 容量とメモリ (RAM) 容量の組み合わせです。この容量範囲は、クラスター内の各 Aurora Serverless v2 DB インスタンスに適用されます。クラスターまたはクラスター内の任意の Aurora Serverless v2 DB インスタンスの容量範囲を確認する手順については、「Aurora Serverless v2 の容量範囲の確認」を参照してください。

AWS Management Console で、[Databases] (データベース) ページの [Size] (サイズ) 列に Aurora Serverless v2 DB インスタンスが表示されます。プロビジョン済み DB インスタンスには、r6g.xlarge などの DB インスタンスクラスの名前が表示されます。Aurora Serverless DB インスタンスでは、DB インスタンスクラスの場合は [Serverless] (サーバーレス)が、DB インスタンスの最小容量と最大容量と合わせて表示されます。例えば、Serverless v2 (4 ~ 64 ACU) または Serverless v2 (1 ~ 40 ACU) のように表示されます。

コンソールの各 Aurora Serverless v2 DB インスタンスの [Configuration] (設定) タブにも同じ情報が表示されます。例えば、以下のような [Instance type] (インスタンスタイプ) セクションが表示される場合があります。ここで、[Instance type] (インスタンスタイプ) の値を Serverless v2 とすると、最小容量の値は 2 ACU (4 GiB) で、最大容量の値は 64 ACU (128 GiB) です。



各 Aurora Serverless v2 DB インスタンスの容量は時系列でモニタリングできます。これにより、各 DB インスタンスが消費する ACU の最小、最大、平均を確認できます。また、DB インスタンスの最小容量または最大容量にどれだけ近づいたかを確認できます。これを AWS Management Console で見るには、DB インスタンスの [Monitoring] (モニタリング) タブで、Amazon CloudWatch メトリクスのグラフを確認します。注目すべきメトリクスと、それを解釈する方法の詳細については、「Aurora Serverless v2 の重要な Amazon CloudWatch メトリクス」を参照してください。

Aurora Serverless v2 のログ記録

データベースのログをオンにするには、カスタムパラメータグループのコンフィギュレーションパラメータを使用して、有効にするログを指定します。

Aurora MySQL では、以下のログを有効にすることができます。

Aurora MySQL	説明
`general_log`	一般ログを作成します。オンにするには、1 に設定します。デフォルトはオフ (0) です。
`log_queries_not_using_indexes`	インデックスを使用しないスロークエリログにクエリを記録します。デフォルトはオフ (0) です。このログを有効にするには、1 に設定します。
`long_query_time`	高速実行クエリがスロークエリログに記録されないようにします。0 から 31536000 までの

Aurora MySQL	説明
	浮動小数点数に設定できます。デフォルトは 0 (アクティブではない) です。
server_audit_events	ログにキャプチャするイベントのリスト。サポートされている値は CONNECT、QUERY、QUERY_DCL 、QUERY_DDL 、QUERY_DML 、TABLE です。
server_audit_logging	この値を 1 に設定すると、サーバー監査ログ記録がオンになります。これをオンにしている場合は、CloudWatch に送信する監査イベントを、server_audit_events パラメータにリストアップすることで指定できます。
slow_query_log	スロークエリログを作成します。スロークエリログをオンにするには、1 に設定します。デフォルトはオフ (0) です。

詳細については、「Amazon Aurora MySQL DB クラスターでのアドバンストな監査の使用」を参照してください。

Aurora PostgreSQL では、Aurora Serverless v2 DB インスタンスで以下のログを有効にできます。

Aurora PostgreSQL	説明
log_connections	成功した各接続をログに記録します。
log_disconnections	セッションの終了をログに記録します (セッションの実行時間も含まれます)。
log_lock_waits	デフォルトは 0 (オフ) です。ロック待機をログに記録するには、1 に設定します。
log_min_duration_statement	ステートメントがログに記録されるまでに実行される最小時間 (ミリ秒単位)。

Aurora PostgreSQL	説明
log_min_messages	ログに記録するメッセージレベルを設定します。サポートされている値は debug5、debug4、debug3、debug2、debug1、inf です。パフォーマンスデータを postgres ログに記録するには、値を debug1 に設定します。
log_temp_files	指定されたキロバイト (KB) を超えるテンポラリファイルの使用を記録します。
log_statement	ログに記録される特定の SQL ステートメントを制御します。サポートされている値は none、ddl、mod、all です。デフォルトは none です。

トピック

- Amazon CloudWatch でのログ記録
- Amazon CloudWatch の Aurora Serverless v2 ログの表示
- Amazon CloudWatch でキャパシティーを監視する

Amazon CloudWatch でのログ記録

Aurora Serverless v2 のログ記録 の手順で、オンにするデータベースログを選択した後、Amazon CloudWatch にアップロード (公開) するログを選択できます。

Amazon CloudWatch を使用すると、ログデータの分析や、アラームの作成、メトリクスの表示を行うことができます。デフォルトでは、Aurora Serverless v2 のエラーログが有効になっており、CloudWatch に自動的にアップロードされます。また、その他の Aurora Serverless v2 DB インスタンスのログを CloudWatch アップロードできます。

次に、CloudWatch にアップロードするログを AWS Management Console の[Log exports] (ログのエクスポート) 設定または AWS CLI の --enable-cloudwatch-logs-exports オプションで選択します。

CloudWatch にアップロードする Aurora Serverless v2 ログを選択できます。詳細については、「Amazon Aurora MySQL DB クラスターでのアドバンストな監査の使用」を参照してください。

Aurora DB クラスターのタイプと同様に、デフォルトの DB クラスターパラメータグループを変更することはできません。代わりに、DB クラスターとエンジンタイプのデフォルトパラメータをベースに、独自の DB クラスターパラメータグループが作成できます。Aurora Serverless v2 DB クラスターを作成する前に、カスタムの DB クラスターパラメータグループを作成しておくことをお勧めします。これにより、コンソールでデータベースを作成するときに、パラメータグループを選択できるようになります。

> **Note**
>
> Aurora Serverless v2では、DB クラスターと DB パラメータグループの両方を作成できます。これはAurora Serverless v1、DB クラスターのパラメータグループしか作成できないのとは対照的です。

Amazon CloudWatch の Aurora Serverless v2 ログの表示

「Amazon CloudWatch でのログ記録」の手順を使用してオンにするデータベースログを選択すると、ログの内容を表示できます。

CloudWatch、Aurora MySQL、Aurora PostgreSQL ログ使用の詳細については、「Amazon CloudWatch でログイベントをモニタリングする」および「Amazon CloudWatch Logs への Aurora PostgreSQL ログの発行」を参照してください。

Aurora Serverless v2 DB クラスターのログを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. AWS リージョン を選択します。
3. [ロググループ] を選択します。
4. リストから Aurora Serverless v2 DB クラスターのログを選択します。ログの命名パターンは、次に従います。

```
/aws/rds/cluster/cluster-name/log_type
```

> **Note**
>
> Aurora MySQL 互換 Aurora Serverless v2 DB クラスターの場合、エラーがない場合でも、エラーログにバッファプールのスケーリングイベントが含まれます。

Amazon CloudWatch でキャパシティーを監視する

Aurora Serverless v2では、CloudWatch を使用して、クラスター内のすべての Aurora Serverless v2 DB インスタンスの容量と使用率をモニタリングできます。インスタンスレベルのメトリクスを表示して、Aurora Serverless v2 DB インスタンスがスケールアップやスケールダウンした場合の容量を確認できます。また、容量に関連したメトリクスを他のメトリクスを比較することで、ワークロードの変化がリソース消費にどのように影響しているかを見ることもできます。例えば、`ServerlessDatabaseCapacity` と `DatabaseUsedMemory`、`DatabaseConnections`、`DMLThroughput` を比較すると、DB クラスターのオペレーション中の応答を評価できます。Aurora Serverless v2 に適用される容量に関連したメトリクスの詳細については、「Aurora Serverless v2 の重要な Amazon CloudWatch メトリクス」を参照してください。

Aurora Serverless v2 DB クラスターのキャパシティーを監視するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. [メトリクス] をクリックします。使用可能なすべてのメトリクスが、サービス名ごとにグループ化されたカードとして、コンソールに表示されます。
3. [RDS] を選択します。
4. (オプション) [Search] (検索) ボックスを使用して、Aurora Serverless v2 に特に重要なメトリクスである `ServerlessDatabaseCapacity`、`ACUUtilization`、`CPUUtilization`、`FreeableMemory` を検索します。

CloudWatch ダッシュボードをセットアップし、容量に関連したメトリクスを使用して Aurora Serverless v2 DB クラスターの容量をモニタリングすることをお勧めします。詳細については、「CloudWatch を使用したダッシュボードのビルド」を参照してください。

Amazon Aurora での Amazon CloudWatch の使用の詳細については、「Amazon CloudWatch Logs への Amazon Aurora MySQL ログの発行」を参照してください。

Aurora Serverless v2 でのパフォーマンスとスケーリング

以下の手順と例は、Aurora Serverless v2 クラスターとそれに関連する DB インスタンスの容量範囲を設定する方法を示しています。以下の手順を使用して、DB インスタンスのビジー状態をモニタリングすることもできます。その結果によって容量範囲を増減させる必要があるかどうかを判断できます。

これらの手順を使用する前に、Aurora Serverless v2 のスケーリングの仕組みを把握しておいてください。スケーリングのメカニズムは、Aurora Serverless v1 とは異なります。詳細については、「Aurora Serverless v2 でのスケーリング」を参照してください。

目次

Aurora クラスターの Aurora Serverless v2 での容量範囲の選択

Aurora Serverless v2 DB インスタンスでは、最初の Aurora Serverless v2 DB インスタンスを DB クラスターに追加するのと同時に、DB クラスター内のすべての DB インスタンスに適用する容量範囲

を設定します。その手順については、「Aurora Serverless v2 クラスターの容量設定」を参照してください。

また、既存のクラスターの容量範囲を変更することもできます。以下のセクションでは、適切な最小値と最大値の選択方法と、容量範囲を変更した場合の動作について詳しく説明します。例えば、容量範囲を変更すると、一部の設定パラメータのデフォルト値が変更されることがあります。パラメータの変更をすべて適用するには、各 Aurora Serverless v2 DB インスタンスの再起動が必要になることがあります。

トピック

- クラスターに Aurora Serverless v2 の最小容量設定を選択する
- クラスターに Aurora Serverless v2 の最大容量設定を選択する
- 例: Aurora MySQL クラスターの Aurora Serverless v2 容量範囲の変更
- 例: Aurora PostgreSQL クラスターの Aurora Serverless v2 容量範囲の変更

クラスターに Aurora Serverless v2 の最小容量設定を選択する

Aurora Serverless v2 の最小容量設定には、常に 0.5 を選択しようとします。この値にすることで、DB インスタンスが完全にアイドル状態になったときに最大限にスケールダウンできます。ただし、そのクラスターの使用方法やその他の設定によっては、最も効果的なのは別の値の場合もあります。最小容量設定を選択する場合は、以下の要素を考慮してください。

- Aurora Serverless v2 DB インスタンスのスケーリングレートは、そのインスタンスの現在の容量によって異なります。現在の容量が大きいほど、スケールアップが速くなります。DB インスタンスを非常に大きな容量にすばやくスケールアップする必要があるときは、スケーリングレートの要件を満たす値に最小容量を設定することを検討してください。
- 通常、ワークロードが高いか低いかを見越して DB インスタンスの DB インスタンスクラスを変更している場合は、その経験を活かして同等の Aurora Serverless v2 容量範囲を概算で見積もることができます。トラフィックが少ない場合に使用するメモリサイズを決定するには、「Aurora 用の DB インスタンスクラスのハードウェア仕様」を参照してください。

  例えば、クラスターのワークロードが低い場合に db.r6g.xlarge DB インスタンスクラスを使用するとします。その DB インスタンスクラスのメモリは 32 GiB です。したがって、Aurora 容量単位 (ACU) の最小設定を 16 に指定すると、ほぼ同じ容量にスケールダウンできる Aurora Serverless v2 DB インスタンスを設定できます。これは、各 ACU が約 2 GiB のメモリに対応するためです。db.r6g.xlarge DB インスタンスの使用率が低い場合に、DB インスタンスをさらにスケールダウンさせるため、やや小さい値を指定することがあります。

- DB インスタンスのバッファキャッシュに一定量のデータがあるときにアプリケーションが最も効率的に動作する場合は、頻繁にアクセスされるデータを保持するのに十分なメモリ容量を持つ最小の ACU 設定を指定することを検討してください。それ以外の場合、Aurora Serverless v2 DB インスタンスがさらに小さいメモリサイズにスケールダウンした場合、一部のデータがバックキャッシュから削除されます。その後、DB インスタンスのスケールアップ時に、その情報が経時的に読み込まれてバッファキャッシュに戻ります。データをバッファキャッシュに戻すための I/O 量が大きい場合は、最小 ACU 値を大きくする方が効果的な場合があります。
- ほとんどの時間、Aurora Serverless v2 DB インスタンスが特定の容量で実行されている場合、ベースラインよりも小さく、ただし小さすぎない最小容量の設定を検討してください。Aurora Serverless v2DB インスタンスが現在の容量が必要容量より極端に小さくない場合、スケールアップする規模と速度を最も効果的に見積もることができます。
- プロビジョン済みワークロードのメモリ要件が T3 や T4g–などの小さな DB インスタンスクラスに対して大きすぎる場合は、R5 や R6g DB インスタンスに相当するメモリを提供する最小 ACU 設定を選択します。

  特に、指定された機能で使用するには、以下の最小容量をお勧めします (これらの推奨値は変更される場合があります)。

  - パフォーマンスインサイト – 2 ACU。
  - Aurora グローバルデータベース — 8 ACU (プライマリ AWS リージョン のみに適用)
- 場合によっては、クラスターに Aurora Serverless v2 ライターとは独立してスケーリングするリーダー DB インスタンスが含まれることがあります。その場合は、ライター DB インスタンスで書き込み集中型のワークロードでビジー状態のときに、リーダー DB インスタンスがライターからの変更を適用できるように十分な大きさの最小容量設定を選択します。昇格階層 2 ~ 15 のリーダーでレプリカのラグが発生した場合は、クラスターの最小容量設定を増やすことを検討してください。リーダー DB インスタンスのスケールをライターと一緒にスケーリングするか、個別にスケーリングするかを選択する方法の詳細については、「Aurora Serverless v2 リーダーの昇格階層の選択」を参照してください。
- Aurora Serverless v2 リーダー DB インスタンスを持つ DB クラスターがある場合、リーダーの昇格階層が 0 または 1 でない場合、リーダーはライター DB インスタンスと一緒にスケーリングされません。この場合、最小容量を小さく設定すると、レプリケーションの遅延が大きくなる場合があります。これは、データベースがビジー状態のときに、ライターからの変更を適用するのに十分な容量がリーダーにない可能性があるためです。最小容量は、ライター DB インスタンスと同程度のメモリと CPU 量を表す値に設定することをお勧めします。

- Aurora Serverless v2 DB インスタンスの `max_connections` パラメータの値は、最大 ACU から得られるメモリサイズに基づきます。ただし、PostgreSQL 互換 DB インスタンスで 0.5 ACU の最小容量を指定すると、`max_connections` の最大値は 2,000 に制限されます。

  Aurora PostgreSQL クラスターを重要な接続ワークロードに使用する場合は、最小 ACU 設定を 1 以上にすることを検討してください。Aurora Serverless v2 が `max_connections` 設定パラメータをどのように処理するかについての詳細は、「Aurora Serverless v2 の最大接続数」を参照してください。

- Aurora Serverless v2 DB インスタンスを最小容量から最大容量までスケーリングするのにかかる時間は、ACU の最小値と最大値の差によって異なります。現在の DB インスタンスの容量が大きい場合、Aurora Serverless v2 では、小さな容量から開始する場合よりも大きな増分で DB インスタンスをスケールアップします。したがって、比較的大きい最大容量を指定し、ほとんどの時間、DB インスタンスがその容量付近で使用されている場合は、最小 ACU の設定を引き上げることを検討してください。そうすれば、アイドル状態の DB インスタンスを、より迅速に最大容量にスケールアップできます。

クラスターに Aurora Serverless v2 の最大容量設定を選択する

Aurora Serverless v2 の最大容量設定には、常にある程度大きい値を選択しようとします。最大容量が大きいと、集中的なワークロードを実行している場合に、DB インスタンスは最もスケールアップできます。値を小さくすると、予期せぬ料金が発生する可能性を回避できます。そのクラスターの使用方法およびその他の設定によっては、最も効果的な値が当初検討していたより大きくなったり、小さくなったりすることがあります。最大容量設定を選択する場合は、以下の要素を考慮してください。

- 最大容量は、最小容量より大きくなければなりません。最小容量と最大容量を同一に設定することができます。ただし、その場合は容量がスケールアップまたはスケールダウンすることはありません。したがって、テスト以外では、最小容量と最大容量に同じ値を使用することは適切ではありません。
- 最大容量は 0.5 ACU より大きくなければなりません。ほとんどの場合、最小容量と最大容量は同じに設定できます。ただし、最小値と最大値の両方に 0.5 を指定することはできません。最大容量には 1 以上の値を使用します。
- 通常、ワークロードが高いか低いかを見越して DB インスタンスの DB インスタンスクラスを変更している場合は、その経験を活用して同等の Aurora Serverless v2 容量範囲を見積もることができます。トラフィックが多い場合に使用するメモリサイズを決定するには、「Aurora 用の DB インスタンスクラスのハードウェア仕様」を参照してください。

例えば、クラスターのワークロードが高い場合に db.r6g.4xlarge DB インスタンスクラスを使用するとします。その DB インスタンスクラスのメモリは 128 GiB です。したがって、ACU の最大設定を 64 に指定すると、ほぼ同じ容量にスケールアップできる Aurora Serverless v2 DB インスタンスを設定できます。これは、各 ACU が約 2 GiB のメモリに対応するためです。db.r6g.4xlarge DB インスタンスにワークロードを効果的に処理するのに十分な容量がないことがあり、DB インスタンスをよりスケールアップさせるために多少大きい値を指定することができます。

- データベースの使用に予算の上限がある場合は、すべての Aurora Serverless v2 DB インスタンスを常に最大容量で実行しても、予算の上限に収まるような値を選択してください。クラスターに n の Aurora Serverless v2 DB インスタンスがある場合、クラスターが常に消費できる Aurora Serverless v2 の理論上の最大容量は、クラスターの最大 ACU 設定の n 倍であることに注意してください。(例えば、一部のリーダーがライターから独立してスケーリングする場合など、実際の消費量は少なくなる場合があります)。
- Aurora Serverless v2 リーダー DB インスタンスを利用してライター DB インスタンスから一部の読み取り専用ワークロードをオフロードするには、最大容量設定を小さく選択できることがあります。これは、各リーダー DB インスタンスが、クラスターに単一の DB インスタンスしか含まれていない場合ほど大きくスケーリングする必要がないことを反映するためです。
- データベースパラメータの設定間違いやアプリケーション内の非効率的なクエリによる過度の使用から保護したいとします。その場合、設定可能な理論的な最大値よりも最大容量設定を小さく選択することで、誤って過剰に使用することを回避できます。
- 実際のユーザーアクティビティによるスパイクがまれしか発生しない場合は、最大容量設定を選択する際にその機会を考慮できます。アプリケーションが完全なパフォーマンスとスケーラビリティで動作し続けることを優先する場合は、通常の使用状況よりも大きい最大容量設定を指定できます。アクティビティの非常に極端なスパイク中、アプリケーションのスループットが低下しても問題ない場合は、最大容量を少し小さめに設定できます。アプリケーションの実行を維持するのに十分なメモリと CPU リソースがある設定を選択してください。
- クラスターで 各 DB インスタンスのメモリ使用量を増やす設定をオンにする場合は、最大 ACU 値を決定する際にそのメモリを考慮に入れてください。このような設定には、パフォーマンスインサイト、Aurora MySQL 並列クエリ、Aurora MySQL パフォーマンススキーマ、Aurora MySQL バイナリログレプリケーションの設定が含まれます。それらの機能が使用されているときに、Aurora Serverless v2 DB インスタンスがワークロードを処理するのに十分なスケールアップができる 最大 ACU 値になっていることを確認します。最大 ACU の設定が小さいことと、メモリのオーバーヘッドが発生する Aurora 機能が組み合わされることによって発生する問題のトラブルシューティングについては、「メモリ不足エラーを回避する」を参照してください。

例: Aurora MySQL クラスターの Aurora Serverless v2 容量範囲の変更

以下の AWS CLI の例では、既存の Aurora MySQL クラスター内の Aurora Serverless v2 DB インスタンスの ACU 範囲を更新する方法を示しています。最初は、クラスターの容量範囲は 8 ~ 32 ACU です。

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 8.0,
    "MaxCapacity": 32.0
}
```

DB インスタンスはアイドル状態で、8 ACU にスケールダウンされます。この時点で DB インスタンスには、以下の容量に関連した設定が適用されます。バッファプールのサイズを読みやすい単位で表すため、2 の 30 乗 で割り算して、ギビバイト (GiB) 単位の測定値で表示します。これは、Aurora のメモリ関連の測定値では 10 の累乗ではなく 2 の累乗に基づく単位を使用しているためです。

```
mysql> select @@max_connections;
+-------------------+
| @@max_connections |
+-------------------+
|              3000 |
+-------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|                9294577664 |
+---------------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+-----------+
| gibibytes |
+-----------+
|   8.65625 |
+-----------+
1 row in set (0.00 sec)
```

次に、クラスターの容量範囲を変更します。変更後、modify-db-cluster コマンドの実行が完了すると、クラスターの ACU 範囲は 12.5 ~ 80 になります。

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration MinCapacity=12.5,MaxCapacity=80

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 12.5,
    "MaxCapacity": 80.0
}
```

容量範囲を変更したことで、一部の設定パラメータのデフォルト値が変更されました。Aurora では、これらの新しいデフォルトの一部をすぐに適用できます。ただし、一部のパラメータの変更は、再起動後に有効になります。この pending-reboot status は、一部のパラメータの変更を適用するために再起動が必要であることを示しています。

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "pending-reboot"
        }
    ]
}
```

この時点では、クラスターはアイドル状態で、DB インスタンス serverless-v2-instance-1 では 12.5 ACU を消費しています。innodb_buffer_pool_size パラメータは、DB インスタンスの現在の容量に基づいて、既に調整されています。max_connections パラメータは、以前の最大容量の値をそのまま反映しています。この値をリセットするには、DB インスタンスを再起動する必要があります。

> **Note**
>
> カスタム DB パラメータグループで max_connections パラメータを直接設定する場合、再起動は必要ありません。

```
mysql> select @@max_connections;
+-------------------+
| @@max_connections |
+-------------------+
|              3000 |
+-------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|               15572402176 |
+---------------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+---------------+
| gibibytes     |
+---------------+
| 14.5029296875 |
+---------------+
1 row in set (0.00 sec)
```

ここで、DB インスタンスを再起動して、再び利用可能になるまで待機します。

```
aws rds reboot-db-instance --db-instance-identifier serverless-v2-instance-1
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBInstanceStatus": "rebooting"
}

aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
```

pending-reboot ステータスがクリアされます。in-sync の値によって、Aurora が保留中のパラメータの変更をすべて適用したことを確認します。

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "in-sync"
        }
    ]
}
```

innodb_buffer_pool_size パラメータは、アイドル状態の DB インスタンスの最終サイズに増加しました。ACU の最大値から計算した値を反映するようにmax_connections パラメータが増加しました。Aurora が max_connections に使用する計算式によると、メモリサイズが 2 倍になると 1,000 増加します。

```
mysql> select @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|               16139681792 |
+---------------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+-----------+
| gibibytes |
+-----------+
|  15.03125 |
+-----------+
1 row in set (0.00 sec)

mysql> select @@max_connections;
+-------------------+
| @@max_connections |
+-------------------+
|              4000 |
```

```
+--------------------+
1 row in set (0.00 sec)
```

容量範囲を 0.5 ~ 128 ACU に設定し、DB インスタンスを再起動します。ここで、アイドル状態の DB インスタンスのバッファキャッシュサイズは 1 GiB 未満なので、メビバイト (MiB) 単位で測定します。5,000 という max_connections 値は、最大容量設定のメモリサイズから算出しています。

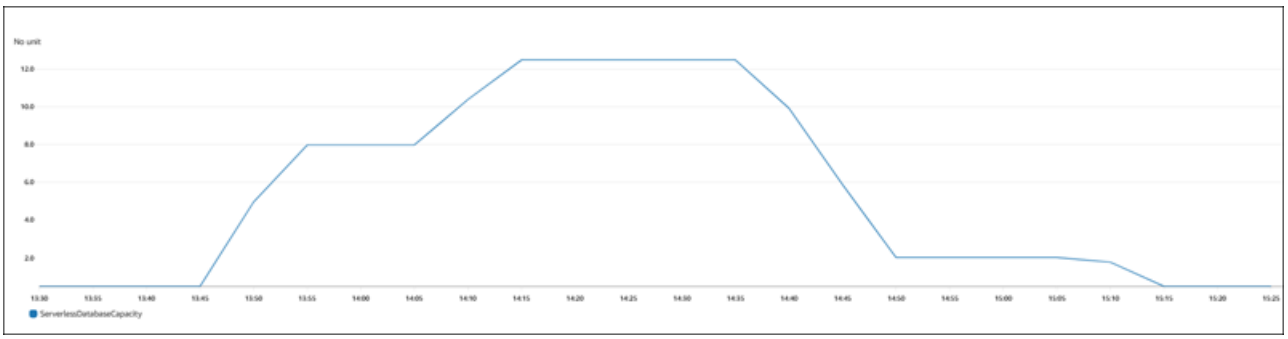
```
mysql> select @@innodb_buffer_pool_size / pow(2,20) as mebibytes, @@max_connections;
+-----------+--------------------+
| mebibytes | @@max_connections |
+-----------+--------------------+
|       672 |               5000 |
+-----------+--------------------+
1 row in set (0.00 sec)
```

例: Aurora PostgreSQL クラスターの Aurora Serverless v2 容量範囲の変更

以下の CLI の例では、既存の Aurora PostgreSQL クラスター内の Aurora Serverless v2 DB インスタンスの ACU 範囲を更新する方法を示しています。

1. クラスターの容量範囲は 0.5 ~ 1 ACU から始まります。
2. 容量範囲を 8 ~ 32 ACU に変更します。
3. 容量範囲を 12.5 ~ 80 ACU に変更します。
4. 容量範囲を 0.5 ~ 128 ACU に変更します。
5. 容量を初期値の 0.5 ~ 1 ACU の範囲に戻します。

次の図は、Amazon CloudWatch の容量の変化を示しています。



DB インスタンスはアイドル状態で、0.5 ACU にスケールダウンされます。この時点で DB インスタンスには、以下の容量に関連した設定が適用されます。

```
postgres=> show max_connections;
 max_connections
-----------------
 189
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 16384
(1 row)
```

次に、クラスターの容量範囲を変更します。変更後、modify-db-cluster コマンドの実行が完了すると、クラスターの ACU 範囲は 8.0 ~ 32 になります。

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 8.0,
    "MaxCapacity": 32.0
}
```

容量範囲を変更することで、一部の設定パラメータのデフォルト値が変更されます。Aurora では、これらの新しいデフォルトの一部をすぐに適用できます。ただし、一部のパラメータの変更は、再起動後に有効になります。この pending-reboot status は、一部のパラメータの変更を適用するために再起動が必要であることを示しています。

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "pending-reboot"
        }
    ]
}
```

この時点では、クラスターはアイドル状態で、DB インスタンス serverless-v2-instance-1 では 8.0 ACU を消費しています。shared_buffers パラメータは、DB インスタンスの現在の容量に基づいて、既に調整されています。max_connections パラメータは、以前の最大容量の値をそのまま反映しています。この値をリセットするには、DB インスタンスを再起動する必要があります。

> **Note**
>
> カスタム DB パラメータグループで max_connections パラメータを直接設定する場合、再起動は必要ありません。

```
postgres=> show max_connections;
 max_connections
-----------------
 189
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 1425408
(1 row)
```

DB インスタンスを再起動して、再び利用可能になるまで待機します。

```
aws rds reboot-db-instance --db-instance-identifier serverless-v2-instance-1
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBInstanceStatus": "rebooting"
}

aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
```

DB インスタンスを再起動したことで、これで pending-reboot ステータスがクリアされました。in-sync の値によって、Aurora が保留中のパラメータの変更をすべて適用したことを確認します。

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
```

```
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "in-sync"
        }
    ]
}
```

再起動後、max_connections は新しい最大容量が反映された値を示します。

```
postgres=> show max_connections;
 max_connections
-----------------
 5000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 1425408
(1 row)
```

次に、クラスターの容量範囲を 12.5 ~ 80 ACU に変更します。

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration MinCapacity=12.5,MaxCapacity=80

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 12.5,
    "MaxCapacity": 80.0
}
```

この時点では、クラスターはアイドル状態で、DB インスタンス serverless-v2-instance-1 では 12.5 ACU を消費しています。shared_buffers パラメータは、DB インスタンスの現在の容量に基づいて、既に調整されています。max_connections の値は 5000 のままです。

```
postgres=> show max_connections;
 max_connections
-----------------
 5000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 2211840
(1 row)
```

再起動しますが、パラメータ値は変わりません。これは、Aurora PostgreSQL を実行している Aurora Serverless v2 DB クラスターの `max_connections` の最大値が 5000 であるためです。

```
postgres=> show max_connections;
 max_connections
-----------------
 5000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 2211840
(1 row)
```

ここで、容量範囲を 0.5 ~ 128 ACU に設定します。DB クラスターは 10 ACU、2 ACU の順にスケールダウンします。DB インスタンスを再起動します。

```
postgres=> show max_connections;
 max_connections
-----------------
 2000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 16384
(1 row)
```

Aurora Serverless v2 DB インスタンスの max_connections の値は、最大 ACU から得られるメモリサイズに基づきます。ただし、PostgreSQL 互換 DB インスタンスで 0.5 ACU の最小容量を指定すると、max_connections の最大値は 2,000 に制限されます。

ここで、容量を初期範囲の 0.5 ~ 1 ACU に戻し、DB インスタンスを再起動します。max_connections パラメータは元の値に戻されました。

```
postgres=> show max_connections;
 max_connections
-----------------
 189
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 16384
(1 row)
```

Aurora Serverless v2 のパラメータグループを使用する

Aurora Serverless v2 DB クラスターの作成時に、特定の Aurora DB エンジンと、それに関連する DB クラスターパラメータグループを選択します。Aurora で、パラメータグループを使用してクラスター間で一貫した設定を適用する方法に精通していない場合は、「「パラメータグループを使用する」」を参照してください。パラメータグループの作成、修正、適用などのアクションに関する手順は、すべて Aurora Serverless v2 に適用されます。

パラメータグループ機能は、プロビジョン済みクラスターと、Aurora Serverless v2 DB インスタンスを含むクラスターの間でほぼ同じように動作します。

- クラスター内のすべての DB インスタンスのデフォルトパラメータ値は、クラスターパラメータグループで定義されます。
- これらの DB インスタンスのカスタム DB パラメータグループを指定することで、特定の DB インスタンスの一部のパラメータを上書きできます。これは、特定の DB インスタンスのデバッグまたはパフォーマンスのチューニング中に行うことができます。例えば、ある Aurora Serverless v2 DB インスタンスとプロビジョン済み DB インスタンスを含むクラスターがあるとします。この場合、カスタム DB パラメータグループを使用して、プロビジョン済み DB インスタンスに複数の異なるパラメータを指定できます。

- Aurora Serverless v2 の場合、provisioned の値を持つすべてのパラメータをパラメータグループ内の SupportedEngineModes 属性に使用できます。Aurora Serverless v1 では、SupportedEngineModes 属性には serverless を持つパラメータのサブセットに限り使用できます。

トピック

- デフォルトパラメータ値
- Aurora Serverless v2 の最大接続数
- Aurora Serverless v2 のスケールアップとスケールダウンの際に Aurora が調整するパラメータ
- Aurora Serverless v2 の最大容量に基づいて Aurora が計算するパラメータ

デフォルトパラメータ値

プロビジョン済み Aurora Serverless v2 DB インスタンスと DB インスタンスの決定的な相違点は、DB インスタンスの容量に関連する特定のパラメータのカスタムパラメータ値を Aurora が上書きするということです。カスタムパラメータ値は、クラスター内のプロビジョン済み DB インスタンスにも適用されます。Aurora Serverless v2 DB インスタンスが Aurora パラメータグループのパラメータを解釈する方法についての詳細は、「Aurora クラスターの設定パラメータ」を参照してください。Aurora Serverless v2 が上書きする具体的なパラメータは、「Aurora Serverless v2 のスケールアップとスケールダウンの際に Aurora が調整するパラメータ」および「Aurora Serverless v2 の最大容量に基づいて Aurora が計算するパラメータ」を参照してください。

CLI コマンドの describe-db-cluster-parameters を使用し AWS リージョン に対しクエリすることで、さまざまな Aurora DB エンジンのデフォルトパラメータグループのデフォルト値リストを取得できます。Aurora Serverless v2 と互換性のあるエンジンバージョンの --db-parameter-group-family と -db-parameter-group-name オプションに使用できる値は以下のとおりです。

データベースエンジンとバージョン	パラメータグループファミリー	デフォルトパラメータグループ名
Aurora MySQL バージョン 3	aurora-mysql8.0	default.aurora-mysql8.0
Aurora PostgreSQL バージョン 13.x	aurora-postgresql13	default.aurora-postgresql13

データベースエンジンとバージョン	パラメータグループファミリー	デフォルトパラメータグループ名
Aurora PostgreSQL バージョン 14.x	aurora-postgresql14	default.aurora-postgresql14
Aurora PostgreSQL バージョン 15.x	aurora-postgresql15	default.aurora-postgresql15
Aurora PostgreSQL バージョン 16.x	aurora-postgresql16	default.aurora-postgresql16

以下の例では、Aurora MySQL 3 と Aurora PostgreSQL 13 のデフォルトの DB クラスターグループからパラメータのリストを取得します。これらは、Aurora Serverless v2 で使用する Aurora MySQL と Aurora PostgreSQL のバージョンです

Linux、macOS、Unix の場合:

```
aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name default.aurora-mysql8.0 \
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' \
  --output text

aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name default.aurora-postgresql13 \
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' \
  --output text
```

Windows の場合:

```
aws rds describe-db-cluster-parameters ^
  --db-cluster-parameter-group-name default.aurora-mysql8.0 ^
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' ^
  --output text
```

```
aws rds describe-db-cluster-parameters ^
  --db-cluster-parameter-group-name default.aurora-postgresql13 ^
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' ^
  --output text
```

Aurora Serverless v2 の最大接続数

Aurora MySQL と Aurora PostgreSQL の両方に対して、Aurora Serverless v2 DB インスタンスでは、max_connections パラメータを一定に維持し、DB インスタンスのスケールダウン時に接続が切断されないようにします。このパラメータのデフォルト値は、DB インスタンスのメモリサイズに基づいた数式から算出されます。プロビジョン済み DB インスタンスクラスの計算式とデフォルト値の詳細については、「Aurora MySQL DB インスタンスへの最大接続数」および「Aurora PostgreSQL DB インスタンスへの最大接続数」を参照してください。

Aurora Serverless v2 が計算式を評価する場合、現在の ACU 値ではなく、DB インスタンスの最大 Aurora 容量単位 (ACU) に基づいたメモリサイズを使用します。デフォルト値を変更する場合は、定数値を指定する代わりに、計算式のバリエーションを使用することをお勧めします。このように、Aurora Serverless v2 では、最大容量に基づいて適切な設定を使用できます。

Aurora Serverless v2 DB クラスターの最大容量を変更する場合、Aurora Serverless v2 DB インスタンスを再起動して max_connections 値を更新する必要があります。これは、Aurora Serverless v2 の場合、max_connections が静的パラメータであるためです。

次の表は、最大 ACU 値に基づいた Aurora Serverless v2 の max_connections のデフォルト値を示しています。

最大 ACU	Aurora MySQL のデフォルトの最大接続値	Aurora PostgreSQL のデフォルトの最大接続値
1	90	189
4	135	823
8	1,000	1,669
16	2,000	3,360

最大 ACU	Aurora MySQL のデフォルトの最大接続値	Aurora PostgreSQL のデフォルトの最大接続値
32	3,000	5,000
64	4,000	5,000
128	5,000	5,000

> **Note**
>
> Aurora Serverless v2 DB インスタンスの `max_connections` の値は、最大 ACU から得られるメモリサイズに基づきます。ただし、PostgreSQL 互換 DB インスタンスで 0.5 ACU の最小容量を指定すると、`max_connections` の最大値は 2,000 に制限されます。

ACU の最大値によって `max_connections` がどのように変化するかを示す具体例については、「例: Aurora MySQL クラスターの Aurora Serverless v2 容量範囲の変更 」と「例: Aurora PostgreSQL クラスターの Aurora Serverless v2 容量範囲の変更」を参照してください。

Aurora Serverless v2 のスケールアップとスケールダウンの際に Aurora が調整するパラメータ

オートスケーリング中、Aurora Serverless v2 は、容量の増減に対して各 DB インスタンスが最適に機能するように、パラメータを変更できる必要があります。したがって、容量に関連する一部のパラメータを上書きすることはできません。上書きできる一部のパラメータでは、固定値のハードコートはしないでください。容量に関連するこれらの設定には、以下の考慮事項が適用されます。

Aurora MySQL の場合、Aurora Serverless v2 では、スケーリング中に一部のパラメータのサイズを動的に変更します。Aurora Serverless v2 では、以下のパラメータには指定したカスタムパラメータ値を使用しません。

- `innodb_buffer_pool_size`
- `innodb_purge_threads`
- `table_definition_cache`
- `table_open_cache`

Aurora PostgreSQL の場合、Aurora Serverless v2 では、スケーリング中に以下のパラメータのパラメータサイズを動的に変更します。Aurora Serverless v2 では、以下のパラメータには指定したカスタムパラメータ値を使用しません。

- `shared_buffers`

ここに示されている以外のパラメータについては、Aurora Serverless v2 DB インスタンスでは、プロビジョン済み DB インスタンスと同じように動作します。デフォルトのパラメータ値は、クラスターパラメータグループから継承されます。カスタムクラスターのパラメータグループを使用して、カスタムクラスター全体のデフォルトを変更できます。または、カスタム DB パラメータグループを使用して、特定の DB インスタンスのデフォルトを変更できます。動的パラメータはすぐに更新されます。静的パラメータの変更は、DB インスタンスの再起動後に有効になります。

Aurora Serverless v2 の最大容量に基づいて Aurora が計算するパラメータ

以下のパラメータについては、Aurora PostgreSQL では、`max_connections` と同様に 最大 ACU 設定に基づくメモリサイズから算出したデフォルト値を使用します。

- `autovacuum_max_workers`
- `autovacuum_vacuum_cost_limit`
- `autovacuum_work_mem`
- `effective_cache_size`
- `maintenance_work_mem`

メモリ不足エラーを回避する

Aurora Serverless v2 DB インスタンスの 1 つが常に最大容量の制限に達している場合、Aurora ではこの状態を DB インスタンスのステータスを `incompatible-parameters` に設定することで表示します。DB インスタンスが `incompatible-parameters` のステータスの間、一部のオペレーションはブロックされます。例えば、エンジンバージョンをアップグレードすることはできません。

通常、DB インスタンスでは、メモリ不足エラーが原因で頻繁に再起動した場合、このステータスになります。Aurora では、このタイプの再起動が発生したときにイベントを記録します。イベントを表示するには、「Amazon RDS イベントの表示」の手順に従います。パフォーマンスインサイトや IAM 認証などの設定をオンにすることによるオーバーヘッドが原因で、メモリ使用量が異常に大きくなる場合があります。また、DB インスタンスのワークロードが高い場合や、多数のスキーマオブジェクトに関連するメタデータの管理から発生する場合があります。

DB インスタンスが頻繁に最大容量に達しないように、メモリの負荷が低くなると、Aurora では DB インスタンスのステータスを自動的に `available` に戻します。

この状態から回復させるには、以下アクションの一部またはすべてを実行できます。

- クラスターの Aurora 容量単位 (ACU) の最小値を変更して、Aurora Serverless v2 DB インスタンスの容量の下限を引き上げます。これを行うことで、アイドル状態のデータベースが、クラスターでオンになっている機能に必要なメモリよりも少ない容量にスケールダウンする問題を回避できます。クラスターの ACU 設定を変更した後、Aurora Serverless v2 DB インスタンスを再起動します。そうすることで、Aurora がステータスを `available` に戻してリセットできるかが評価されます
- クラスターの ACU の最大値を変更して、Aurora Serverless v2 DB インスタンスの容量の上限を引き上げます。そうすることで、クラスターでオンになっている機能やデータベースワークロードに、ビジー状態のデータベースが十分なメモリがある容量にスケールアップできない問題を回避できます。クラスターの ACU 設定を変更した後、Aurora Serverless v2 DB インスタンスを再起動します。そうすることで、Aurora がステータスを `available` に戻してリセットできるかが評価されます
- メモリオーバーヘッドが必要な設定をオフにします。例えば、AWS Identity and Access Management (IAM)、パフォーマンスインサイト、Aurora MySQL バイナリログレプリケーションをオンにしているが、使用していないとします。その場合は、これらをオフにできます。または、クラスターの最小容量と最大容量値を調整することで、これらの機能で使用されるメモリを考慮することもできます。最小および最大容量設定の選択に関するガイドラインについては、「Aurora クラスターの Aurora Serverless v2 での容量範囲の選択」を参照してください。
- DB インスタンスのワークロードを削減します。例えば、クラスターにリーダー DB インスタンスを追加して、読み取り専用クエリからのロードを他の DB インスタンスに分散させることができます。
- アプリケーションで使用される SQL コードを調整して、使用されるリソースを削減します。例えば、クエリプランの確認、遅いクエリログのチェック、テーブルのインデックスの調整ができます。その他、従来の SQL チューニングを実行することもできます。

Aurora Serverless v2 の重要な Amazon CloudWatch メトリクス

Aurora Serverless v2 DB インスタンスで Amazon CloudWatch の使用を開始するには、「Amazon CloudWatch の Aurora Serverless v2 ログの表示」を参照してください。CloudWatch を使用して Aurora DB クラスターをモニタリングする方法の詳細については、「Amazon CloudWatch でログイベントをモニタリングする」を参照してください。

CloudWatch で Aurora Serverless v2 DB インスタンスを表示すると、ServerlessDatabaseCapacity メトリクスで各 DB インスタンスが消費する容量をモニタリングできます。また、DatabaseConnections や Queries などの Aurora CloudWatch のスタンダードのメトリクスをすべてモニタリングできます。Aurora でモニタリング可能な CloudWatch メトリクスのすべてのリストは、「Amazon Aurora の Amazon CloudWatch メトリクス」を参照してください。メトリクスは、Amazon Aurora のクラスターレベルのメトリクス および Amazon Aurora のインスタンスレベルのメトリクス で、クラスターレベルとインスタンスレベルのメトリクスに分けることができます。

以下の CloudWatch インスタンスレベルのメトリクスは、Aurora Serverless v2 DB インスタンスはスケールアップとスケールダウンを理解するうえで重要なモニタリングです。これらすべてのメトリクスは 1 秒ごとに計算されます。そうすれば、Aurora Serverless v2 DB インスタンスの現在のステータスをモニタリングできます。Aurora Serverless v2 DB インスタンスが容量に関連するメトリクスのしきい値に近づいた場合に通知するアラームを設定できます。最小容量と最大容量設定は適切か、調整が必要かを判断できます。データベースの効率を最適化するため、どこに注力すべきかを判断できます。

- ServerlessDatabaseCapacity。インスタンスレベルのメトリクスとして、現在の DB インスタンスの容量で表される ACU 値を報告します。クラスターレベルのメトリクスとして、クラスター内のすべての Aurora Serverless v2 DB インスタンスの ServerlessDatabaseCapacity 値の平均を表しています。このメトリクスは、Aurora Serverless v1 ではクラスターレベルのメトリクスに限られます。Aurora Serverless v2 では、DB インスタンスレベルとクラスターレベルで利用できます。
- ACUUtilization。これは Aurora Serverless v2 での新しいメトリクスです。この値は割合 (%) で表されます。これは、ServerlessDatabaseCapacity メトリクスの値を DB クラスターの最大 ACU 値で割った値です。このメトリクスを解釈してアクションを実行するには、以下のガイドラインを考慮してください。
  - このメトリクスが 100.0 値に近づいた場合、DB インスタンスは限りなく大きくスケールアップしたことになります。クラスターの最大 ACU 設定を引き上げることを検討してください。これにより、ライターとリーダーの両方の DB インスタンスを、より大きな容量にスケーリングできます。
  - 読み取り専用のワークロードによって、リーダー DB インスタンスが 100.0 のACUUtilization に近づき、一方でライター DB インスタンスは最大容量に近づいていないとします。その場合は、クラスターにリーダー DB インスタンスを追加することを検討してください。これにより、ワークロードの読み取り専用部分のワークロードをより多くの DB インスタンスに分散することで、各リーダー DB インスタンスのロードを軽減できます。

  - パフォーマンスとスケーラビリティが主な考慮事項である本番アプリケーションを実行しているとします。その場合、クラスターの最大 ACU 値を大きい数値に設定できます。目標は、ACUUtilization のメトリクスが常に 100.0 未満であることです。ACU の最大値を大きくすると、データベースのアクティビティに予期しないスパイクが発生した場合でも十分な余裕があり、安心につながります。実際に消費されたデータベース容量に対してのみ課金されます。

- CPUUtilization。このメトリクスは Aurora Serverless v2 において、プロビジョン済みの DB インスタンスとは異なる解釈がされます。Aurora Serverless v2 の場合、この値は、現在の CPU の使用量を DB クラスターの最大 ACU 値で使用可能な CPU 容量で割った割合です。Aurora はこの値を自動的にモニタリングし、DB インスタンスが CPU 容量 を使用している割合が常に大きい場合、Aurora Serverless v2 DB インスタンスをスケールアップします。

  このメトリクスが 100.0 値に近づいた場合、DB インスタンスは最大 CPU 容量に達しています。クラスターの最大 ACU 設定を引き上げることを検討してください。このメトリクスがリーダー DB インスタンスで 100.0 値に近づいた場合、クラスターにリーダー DB インスタンスを追加することを検討してください。これにより、ワークロードの読み取り専用部分のワークロードをより多くの DB インスタンスに分散することで、各リーダー DB インスタンスのロードを軽減できます。

- FreeableMemory。この値は、Aurora Serverless v2 DB インスタンスを最大容量にスケーリングしたときに利用できる未使用のメモリ量を表します。現在の容量が最大容量を下回る 各 ACU では、この値は約 2 GiB 増加します。したがって、DB インスタンスが限りなく大きくスケールアップされるまで、このメトリクスはゼロに近づきません。

  このメトリクスが 0 値に近づいた場合、DB インスタンスは可能な限りスケールアップし、使用可能なメモリの上限に近づいています。クラスターの最大 ACU 設定を引き上げることを検討してください。このメトリクスがリーダー DB インスタンスで 0 値に近づいた場合、クラスターにリーダー DB インスタンスを追加することを検討してください。これにより、ワークロードの読み取り専用部分のワークロードをより多くの DB インスタンスに分散することで、各リーダー DB インスタンスのメモリ使用量を軽減できます。

- TempStorageIops。DB インスタンスにアタッチされたローカルストレージで実行された IOPS の数です。これには、読み取りと書き込みの両方の IOPS が含まれます。このメトリクスはカウントを表し、1 秒に 1 回測定されます。これは、Aurora Serverless v2 の新しいメトリクスです。詳細については、「Amazon Aurora のインスタンスレベルのメトリクス」を参照してください。

- TempStorageThroughput。DB インスタンスに関連するローカルストレージとの間で転送されるデータの量です。このメトリクスはバイトを表し、1 秒に 1 回測定されます。これは、Aurora Serverless v2 の新しいメトリクスです。詳細については、「Amazon Aurora のインスタンスレベルのメトリクス」を参照してください。

通常、Aurora Serverless v2 DB インスタンスのスケールアップの大部分は、メモリ使用率と CPU アクティビティに起因しています。TempStorageIops および TempStorageThroughput のメトリクスは、DB インスタンスとローカルストレージデバイス間の転送のためのネットワークアクティビティが、予期しない容量増加の原因となるまれなケースを診断するのに役立ちます。他のネットワークアクティビティを監視するには、以下の既存のメトリクスを使用できます。

- NetworkReceiveThroughput
- NetworkThroughput
- NetworkTransmitThroughput
- StorageNetworkReceiveThroughput
- StorageNetworkThroughput
- StorageNetworkTransmitThroughput

Aurora で、一部またはすべてのデータベースログを CloudWatch に発行することができます。公開するログを選択するには、Aurora Serverless v2 DB インスタンスを含むクラスターに関連する DB クラスターパラメータグループの general_log や slow_query_log などの設定パラメータをオンにします。ログ設定パラメータをオフにすると、CloudWatch へのログの公開が停止します。不要になったログは CloudWatch で削除することもできます。

Aurora Serverless v2 メトリクスを AWS の請求に適用する方法

AWS の請求書に記載されている Aurora Serverless v2 の料金は、お客様がモニタリング可能な ServerlessDatabaseCapacity メトリクスと同じものに基づいて計算されています。請求メカニズムでは、Aurora Serverless v2 の容量を 1 時間の一部しか使用していない場合、このメトリクスで計算された CloudWatch の平均とは異なる場合があります。また、システムの問題で、短時間 CloudWatch メトリクスが利用できない場合にも異なる場合があります。したがって、お客様が ServerlessDatabaseCapacity の平均値から計算したものと、請求書に記載される ACU 時間の値が若干異なる場合があります。

Aurora Serverless v2 メトリクス用の CloudWatch コマンドの例

以下の AWS CLI の例では、Aurora Serverless v2 に関連する最も重要な CloudWatch メトリクスをモニタリングする方法を示しています。いずれの場合も、--dimensions パラメータの Value= 文字列は、お客様の Aurora Serverless v2 インスタンスの ID に置き換えてください。

以下の Linux の例では、DB インスタンスの最小、最大、平均の容量値を 1 時間で 10 分ごとに測定して表示しています。Linux の date コマンドでは、現在の日付と時刻を基準にして開始時刻と終了

時刻を指定します。--query パラメータの sort_by 関数は、Timestamp のフィールドに基づいて結果を時系列でソートします。

```
aws cloudwatch get-metric-statistics --metric-name "ServerlessDatabaseCapacity" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

以下の Linux の例では、クラスター内の各 DB インスタンスの容量のモニタリングをデモンストレーションしています。各 DB インスタンスの最小、最大、平均の容量使用率を測定しています。測定は、1 時間に 1 回、3 時間にわたって行います。これらの例では、ACU の固定数を表すの ServerlessDatabaseCapacity の代わりに、ACU の上限に対する割合を表す ACUUtilization メトリクスを使用しています。そうすれば、容量範囲の最小と最大の ACU 値の実際の数値を知る必要はありません。割合は 0 から 100 までの範囲で表示できます。

```
aws cloudwatch get-metric-statistics --metric-name "ACUUtilization" \
  --start-time "$(date -d '3 hours ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_writer_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table

aws cloudwatch get-metric-statistics --metric-name "ACUUtilization" \
  --start-time "$(date -d '3 hours ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_reader_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

以下の Linux の例では、前のものと同様の測定を実行します。この場合は、CPUUtilization のメトリクスのための測定になります。測定は、1 時間で 10 分ごとに行われます。この数値は、DB インスタンスの最大容量設定に利用可能な CPU リソースに基づき、利用可能な CPU リソースを表します。

```
aws cloudwatch get-metric-statistics --metric-name "CPUUtilization" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_instance \
```

```
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

以下の Linux の例では、前のものと同様の測定を実行します。この場合は、FreeableMemory のメトリクスのための測定になります。測定は、1 時間で 10 分ごとに行われます。

```
aws cloudwatch get-metric-statistics --metric-name "FreeableMemory" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

パフォーマンスインサイトで Aurora Serverless v2 のパフォーマンスをモニタリングする

パフォーマンスインサイトを使用して、Aurora Serverless v2 DB インスタンスのパフォーマンスをモニタリングできます。パフォーマンスインサイトの手順については、「Amazon Aurora での Performance Insights を使用したDB 負荷のモニタリング」を参照してください。

以下の新しいパフォーマンスインサイトカウンターが Aurora Serverless v2 DB インスタンスに適用されます。

- os.general.serverlessDatabaseCapacity - ACU 内の DB インスタンスの現在の容量。この値は、ServerlessDatabaseCapacity DB インスタンスの CloudWatch メトリクスに対応します。
- os.general.acuUtilization — 設定された最大容量のうち、現在の容量の割合。この値は、ACUUtilization DB インスタンスの CloudWatch メトリクスに対応します。
- os.general.maxConfiguredAcu — この Aurora Serverless v2 DB インスタンスのために設定した最大容量。これは、ACU で測定されます。
- os.general.minConfiguredAcu — この Aurora Serverless v2 DB インスタンスのために設定した最小容量。これは、ACU で測定されます

パフォーマンスインサイトカウンターのすべてのリストは、「Performance Insights カウンターメトリクス」を参照してください。

パフォーマンスインサイトで Aurora Serverless v2 DB インスタンスの vCPU 値が表示される場合、その値は、DB インスタンスの ACU 値に基づいた推定値を表します。デフォルトの 1 分間隔

では、vCPU 値の小数分は整数に切り上げられます。それ以上の間隔の場合、表示される vCPU 値は、1 分ごとの vCPU 値の整数の平均になります。

Aurora Serverless v2 の容量の問題のトラブルシューティング

場合によっては、データベースに負荷がかからない状態でも、Aurora Serverless v2 が最小容量にスケールダウンしない場合があります。これは、次のような理由で発生します。

- 特定の機能により、リソースの使用量が増加し、データベースが最小容量までスケールダウンできない可能性があります。主な機能は以下のとおりです。
  - Aurora Global Database
  - CloudWatch Logs のエクスポート
  - Aurora PostgreSQL 互換 DB クラスターでの `pg_audit` の有効化
  - 拡張モニタリング
  - Performance Insights

  詳細については、「クラスターに Aurora Serverless v2 の最小容量設定を選択する」を参照してください。
- リーダーインスタンスが最小容量までスケールダウンせず、ライターインスタンスと同じかそれ以上の容量にとどまっている場合は、リーダーインスタンスの優先度の階層を確認します。階層 0 または 1 の Aurora Serverless v2 リーダー DB インスタンスは、ライター DB インスタンスと少なくとも同程度の最小容量に保たれます。リーダーの優先度の階層を 2 以上に変更して、ライターとは無関係にスケールアップおよびスケールダウンされるようにします。詳細については、「Aurora Serverless v2 リーダーの昇格階層の選択」を参照してください。
- 共有メモリのサイズに影響するデータベースパラメータをデフォルト値に設定します。デフォルト値より大きく設定すると、共有メモリ要件が増加し、データベースが最小容量までスケールダウンできなくなります。例は、`max_connections` および `max_locks_per_transaction` です。

  > **Note**
  >
  > 共有メモリパラメータを更新するには、データベースを再起動して変更を有効にする必要があります。

- データベースワークロードが重いと、リソースの使用量が増える可能性があります。
- データベースのボリュームが大きいと、リソースの使用量が増える可能性があります。

Amazon Aurora は DB クラスターの管理にメモリと CPU リソースを使用します。Aurora は、データベースボリュームが大きい DB クラスターを管理するために、より多くの CPU とメモリを必要とします。クラスターの最小容量がクラスター管理に必要な最小容量よりも少ない場合、クラスターは最小容量までスケールダウンされません。

- 消去などのバックグラウンド処理によっても、リソースの使用量が増加する場合があります。

それでもデータベースが設定された最小容量までスケールダウンしない場合は、データベースを停止して再起動し、時間の経過とともに蓄積した可能性があるメモリフラグメントを再利用します。データベースを停止して起動するとダウンタイムが発生するため、これを実行するかどうかは慎重に判断することをお勧めします。

Aurora Serverless v2 への移行

既存の DB クラスターを変換して Aurora Serverless v2 を使用する手順は、次のとおりです。

- プロビジョニングされた Aurora DB クラスターからアップグレードします。
- Aurora Serverless v1 クラスターからアップグレードします。
- オンプレミスのデータベースから Aurora Serverless v2 クラスターに移行します。

アップグレードされたクラスターが Aurora Serverless v2 の要件と制限 に記載されている適切なエンジンバージョンを実行している場合、Aurora Serverless v2 DBインスタンスの追加を始めることができます。アップグレードされたクラスターに追加する最初の DB インスタンスは、プロビジョニングされた DB インスタンスである必要があります。次に、書き込みワークロード、読み取りワークロード、またはその両方の処理を、Aurora Serverless v2 DBインスタンスに切り替えることができます。

目次

  - Aurora Serverless v2 と Aurora Serverless v1 機能サポートの比較
  - Aurora Serverless v1 ユースケースの Aurora Serverless v2 への適応
- Aurora Serverless v1 クラスターから Aurora Serverless v2 クラスターへのアップグレード
  - Aurora MySQL 互換 DB クラスター
  - Aurora PostgreSQL 互換 DB クラスター
- オンプレミスデータベースから Aurora Serverless v2 に移行する

> **Note**
>
> これらのトピックでは、既存の DB クラスターを変換する方法について説明します。新しい Aurora Serverless v2 DB クラスターの作成については、「Aurora Serverless v2 を使用する DB クラスターの作成」を参照してください。

既存のクラスターをアップグレードまたは切り替えて Aurora Serverless v2 を使用する

プロビジョニングされたクラスターに、Aurora Serverless v2 をサポートするエンジンバージョンがある場合、Aurora Serverless v2 に切り替えると、アップグレードは必要ありません。その場合は、元のクラスターに Aurora Serverless v2 DB インスタンスを追加できます。クラスターを切り替えて、すべての Aurora Serverless v2 DB インスタンスを使用することができます。また、同じ DB クラスターで Aurora Serverless v2 とプロビジョニングされた DB インスタンスを組み合わせて使用することもできます。Aurora Serverless v2 をサポートする Aurora エンジンのバージョンについては、「Aurora Serverless v2 でサポートされているリージョンと Aurora DB エンジン」を参照してください。

Aurora Serverless v2 をサポートしていない下位のエンジンバージョンを使用している場合、以下の一般的な手順を実行します。

1. クラスターをアップグレードします。
2. アップグレードされたクラスターのプロビジョニングされた書き込み DB インスタンスを作成します。
3. クラスターを変更して Aurora Serverless v2 DB インスタンスを使用する

> **Important**
>
> スナップショットの復元やクローンを使用して Aurora Serverless v2 互換バージョンへのメジャーバージョンアップグレードを実行する場合、新しいクラスターに追加する最初の DB インスタンスはプロビジョニングされた DB インスタンスである必要があります。この追加により、アップグレードプロセスの最終段階が開始されます。
> 最終段階が発生するまで、クラスターには Aurora Serverless v2 サポートに必要なインフラストラクチャがありません。したがって、これらのアップグレードされたクラスターは、常にプロビジョニングされた書き込み DB インスタンスで始まります。その後、プロビジョニングされた DB インスタンスを Aurora Serverless v2 インスタンスに変換したり、フェイルオーバーさせることができます。

Aurora Serverless v1 から Aurora Serverless v2 へのアップグレードでは、中間ステップとしてプロビジョニングされたクラスターを作成する必要があります。次に、プロビジョニングされたクラスターで開始したときと同じアップグレードステップを実行します。

MySQL 互換クラスターが Aurora Serverless v2 を使用するためのアップグレードパス

元のクラスターで Aurora MySQL を実行している場合は、クラスターのエンジンのバージョンとエンジンモードに応じて適切な手順を選択します。

元の Aurora MySQL クラスターがこれである場合	Aurora Serverless v2 に切り替えるには、この操作を行います
Aurora MySQL バージョン 3 を実行するプロビジョニングされたクラスターは MYSQL 8.0 との互換性があります。	これは、既存の Aurora MySQL クラスターからのすべての変換の最終段階です。 必要に応じて、バージョン 3.02.0 以降へのマイナーバージョンアップグレードを実行します。書き込み DB インスタンスにプロビジョニングされた DB インスタンスを使用します。Aurora Serverless v2 書き込み DB インスタンスを 1 つ追加します。フェイルオーバーを実行して、それを書き込み DB インスタンスにします。

元の Aurora MySQL クラスターがこれである場合	Aurora Serverless v2 に切り替えるには、この操作を行います
	(オプション) クラスター内の他のプロビジョニングされた DB インスタンスを Aurora Serverless v2 に変換します。または、新しい Aurora Serverless v2 DB インスタンスを追加して、プロビジョニングされた DB インスタンスを削除します。 完全な手順と例については、「プロビジョニングされたクラスターから Aurora Serverless v2 への切り替え」を参照してください。
Aurora MySQL バージョン 2 を実行するプロビジョニングされたクラスターは MySQL 5.7 との互換性があります。	Aurora MySQL バージョン 3.02.0 以降へのメジャーバージョンのアップグレードを実行します。次に、Aurora MySQL バージョン 3 の手順に従って、Aurora Serverless v2 DB インスタンスを使用するようにクラスターを切り替えます。
Aurora MySQL バージョン 2 を実行する Aurora Serverless v1 クラスターは MySQL 5.7 と互換性があります。	Aurora Serverless v1 からの変換を計画するには、まず Aurora Serverless v2 と Aurora Serverless v1 の比較 を参照してください。 その後「Aurora Serverless v1 クラスターから Aurora Serverless v2 クラスターへのアップグレード」の手順に従います。

PostgreSQL 互換クラスターが Aurora Serverless v2 を使用するためのアップグレードパス。

元のクラスターで Aurora PostgreSQL を実行している場合は、クラスターのエンジンのバージョンとエンジンモードに応じて適切な手順を選択します。

元の Aurora PostgreSQL クラスターがこれである場合	Aurora Serverless v2 に切り替えるには、この操作を行います
Aurora PostgreSQL バージョン 13 を実行しているプロビジョニングされたクラスター	これは、既存の Aurora PostgreSQL クラスターからのすべての変換の最終段階です。 必要に応じて、バージョン 13.6 以降へのマイナーバージョンアップグレードを実行します。書き込み DB インスタンスにプロビジョニングされた DB インスタンスを追加します。Aurora Serverless v2 書き込み DB インスタンスを 1 つ追加します。その Aurora Serverless v2 インスタンスをライター DB インスタンスにするためにフェイルオーバーを実行します。 (オプション) クラスター内の他のプロビジョニングされた DB インスタンスを Aurora Serverless v2 に変換します。または、新しい Aurora Serverless v2 DB インスタンスを追加して、プロビジョニングされた DB インスタンスを削除します。 完全な手順と例については、「プロビジョニングされたクラスターから Aurora Serverless v2 への切り替え」を参照してください。
Aurora PostgreSQL バージョン 11 または 12 を実行しているプロビジョニングされたクラスター	Aurora PostgreSQL バージョン 13.6 以降へのメジャーバージョンアップグレードを実行します。次に、Aurora PostgreSQL バージョン 13 の手順に従って、Aurora Serverless v2 DB インスタンスを使用するようにクラスターを切り替えます。
Aurora PostgreSQL バージョン 11 または 13 を実行している Aurora Serverless v1 クラスター	Aurora Serverless v1 からの変換を計画するには、まず Aurora Serverless v2 と Aurora Serverless v1 の比較 を参照してください。

元の Aurora PostgreSQL クラスターがこれである場合	Aurora Serverless v2 に切り替えるには、この操作を行います
	その後「Aurora Serverless v1 クラスターから Aurora Serverless v2 クラスターへのアップグレード」の手順に従います。

プロビジョニングされたクラスターから Aurora Serverless v2 への切り替え

プロビジョニングされたクラスターを Aurora Serverless v2 を使用できるように切り替えるには、以下の手順に従います。

1. Aurora Serverless v2 DBインスタンスで使用するためにプロビジョニングされたクラスターのアップグレードが必要かどうかを確認します。Aurora Serverless v2 と互換性がある Aurora バージョンについては、「Aurora Serverless v2 の要件と制限」を参照してください。

   プロビジョニングされたクラスターが Aurora Serverless v2 で使用できないエンジンバージョンを実行している場合、クラスターのエンジンバージョンをアップグレードします。

   - MySQL 5.7 互換のプロビジョニングされたクラスターを使用している場合は、Aurora MySQL バージョン 3 へのアップグレード手順に従ってください。インプレースアップグレードの実行手順 にある手順を実行します。
   - PostgreSQL バージョン 11 または 12 を実行している PostgreSQL 互換のプロビジョニングされたクラスターを使用している場合は、Aurora PostgreSQL バージョン 13 のアップグレード手順に従ってください。メジャーバージョンのアップグレードを実施する方法 にある手順を実行します。
2. その他のクラスタープロパティは、Aurora Serverless v2 の要件と制限 からの Aurora Serverless v2 要件と一致するように設定します。
3. クラスターのスケーリング設定を設定します。「Aurora Serverless v2 クラスターの容量設定」の手順に従います。
4. レイヤーに 1 つ以上の Aurora Serverless v2 DB インスタンスを追加します。DB クラスターに Aurora レプリカを追加する の一般的な手順に従います。新しい DB インスタンスごとに、AWS Management Console では サーバーレス、AWS CLI では `db.serverless` という特別な DB インスタンスクラス名、または Amazon RDS API を指定します。

場合によっては、クラスター内に 1 つ以上のプロビジョニングされた読み取り DB インスタンスが既に存在することがあります。その場合、新しい DB インスタンスを作成する代わりに、リーダーの 1 つを Aurora Serverless v2 DB インスタンスに変換できます。これを行うには、「プロビジョン済みライターまたはリーダーを Aurora Serverless v2 に変換する」の手順に従います。

5. Aurora Serverless v2 DB インスタンスの 1 つをクラスターの書き込み DB インスタンスとするフェイルオーバーオペレーションを行います。
6. (オプション) プロビジョニングされた DB インスタンスを Aurora Serverless v2 に変換するか、クラスターから削除します。プロビジョン済みライターまたはリーダーを Aurora Serverless v2 に変換する または Aurora DB クラスターからの DB インスタンスの削除 の一般的な手順に従います。

> **Tip**
>
> プロビジョニングされた DB インスタンスの削除は必須ではありません。Aurora Serverless v2 とプロビジョニングされた DB インスタンスの両方を含むクラスターを設定できます。ただし、Aurora Serverless v2 DB インスタンスのパフォーマンスとスケーリング特性に精通するまでは、すべて同じタイプの DB インスタンスでクラスターを設定することをお勧めします。

次の AWS CLI の例では、Aurora MySQL バージョン 3.02.0 を実行しているプロビジョニングされたクラスターを使用して、切り替え処理を示しています。クラスターには `mysql-80` という名前が付けられています。このクラスターは、`provisioned-instance-1` と `provisioned-instance-2`という 2 つのプロビジョニングされた DB インスタンス、ライターとリーダーから始まります。両者とも `db.r6g.large` DB インスタンスクラスを使用しています。

```
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 \
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' --output text
mysql-80
provisioned-instance-2     False
provisioned-instance-1     True

$ aws rds describe-db-instances --db-instance-identifier provisioned-instance-1 \
  --output text --query '*[].[DBInstanceIdentifier,DBInstanceClass]'
provisioned-instance-1     db.r6g.large

$ aws rds describe-db-instances --db-instance-identifier provisioned-instance-2 \
```

```
  --output text --query '*[].[DBInstanceIdentifier,DBInstanceClass]'
provisioned-instance-2     db.r6g.large
```

いくつかのデータを含むテーブルを作成します。これにより、切り替えの前後のクラスターのデータとオペレーションが同じであることを確認できます。

```
mysql> create database serverless_v2_demo;
mysql> create table serverless_v2_demo.demo (s varchar(128));
mysql> insert into serverless_v2_demo.demo values ('This cluster started with a
 provisioned writer.');
Query OK, 1 row affected (0.02 sec)
```

まず、クラスターに容量範囲を追加します。そうしないと、Aurora Serverless v2 DB インスタンスをクラスターに追加する際にエラーが発生します。この手順で AWS Management Console を使用すると、最初の Aurora Serverless v2 DB インスタンスを追加するときに、そのステップが自動的に行われます。

```
$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-1 \
  --db-cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-
mysql

An error occurred (InvalidDBClusterStateFault) when calling the CreateDBInstance
 operation:
Set the Serverless v2 scaling configuration on the parent DB cluster before creating a
 Serverless v2 DB instance.

$ # The blank ServerlessV2ScalingConfiguration attribute confirms that the cluster
 doesn't have a capacity range set yet.
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 --query
 'DBClusters[*].ServerlessV2ScalingConfiguration'
[]

$ aws rds modify-db-cluster --db-cluster-identifier mysql-80 \
  --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=16
{
  "DBClusterIdentifier": "mysql-80",
  "ServerlessV2ScalingConfiguration": {
    "MinCapacity": 0.5,
    "MaxCapacity": 16
  }
}
```

元の DB インスタンスの代わりに 2Aurora Serverless v2 つのリーダーを作成します。そのためには、新しいdb.serverless DB インスタンスに DB インスタンスクラスを指定する

```
$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-1 --db-
cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-mysql
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBClusterIdentifier": "mysql-80",
  "DBInstanceClass": "db.serverless",
  "DBInstanceStatus": "creating"
}

$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-2 \
  --db-cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-
mysql
{
  "DBInstanceIdentifier": "serverless-v2-instance-2",
  "DBClusterIdentifier": "mysql-80",
  "DBInstanceClass": "db.serverless",
  "DBInstanceStatus": "creating"
}

$ # Wait for both DB instances to finish being created before proceeding.
$ aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
 && \
  aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-2
```

フェイルオーバーを実行して、Aurora Serverless v2 DB インスタンスの 1 つをクラスターの新しいライターにします。

```
$ aws rds failover-db-cluster --db-cluster-identifier mysql-80 \
  --target-db-instance-identifier serverless-v2-instance-1
{
  "DBClusterIdentifier": "mysql-80",
  "DBClusterMembers": [
    {
      "DBInstanceIdentifier": "serverless-v2-instance-1",
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "serverless-v2-instance-2",
```

```
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "provisioned-instance-2",
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "provisioned-instance-1",
      "IsClusterWriter": true,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    }
  ],
  "Status": "available"
}
```

その変更が有効になるまで、数秒かかります。その時点で、Aurora Serverless v2 ライターと Aurora Serverless v2 リーダーがあることになります。したがって、元のプロビジョニングされた DB インスタンスも必要ありません。

```
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 \
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' \
  --output text
mysql-80
serverless-v2-instance-1        True
serverless-v2-instance-2        False
provisioned-instance-2          False
provisioned-instance-1          False
```

切り替え手順の最後のステップは、プロビジョニングされた両方の DB インスタンスを削除することです。

```
$ aws rds delete-db-instance --db-instance-identifier provisioned-instance-2 --skip-
final-snapshot
{
  "DBInstanceIdentifier": "provisioned-instance-2",
  "DBInstanceStatus": "deleting",
```

```
  "Engine": "aurora-mysql",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "DBInstanceClass": "db.r6g.large"
}

$ aws rds delete-db-instance --db-instance-identifier provisioned-instance-1 --skip-
final-snapshot
{
  "DBInstanceIdentifier": "provisioned-instance-1",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "DBInstanceClass": "db.r6g.large"
}
```

最終チェックとして、Aurora Serverless v2 ライター DB インスタンスから元のテーブルにアクセスでき、書き込みが可能であることを確認します。

```
mysql> select * from serverless_v2_demo.demo;
+--------------------------------------------------+
| s                                                |
+--------------------------------------------------+
| This cluster started with a provisioned writer.  |
+--------------------------------------------------+
1 row in set (0.00 sec)

mysql> insert into serverless_v2_demo.demo values ('And it finished with a Serverless
 v2 writer.');
Query OK, 1 row affected (0.01 sec)

mysql> select * from serverless_v2_demo.demo;
+--------------------------------------------------+
| s                                                |
+--------------------------------------------------+
| This cluster started with a provisioned writer.  |
| And it finished with a Serverless v2 writer.     |
+--------------------------------------------------+
2 rows in set (0.01 sec)
```

また、Aurora Serverless v2 読み取り DB インスタンスに接続し、新たに書き込まれたデータがそこでも利用できることを確認します。

```
mysql> select * from serverless_v2_demo.demo;
```

```
+-------------------------------------------------+
| s                                               |
+-------------------------------------------------+
| This cluster started with a provisioned writer. |
| And it finished with a Serverless v2 writer.    |
+-------------------------------------------------+
2 rows in set (0.01 sec)
```

Aurora Serverless v2 と Aurora Serverless v1 の比較

既に Aurora Serverless v1 を使用している場合、Aurora Serverless v1 と Aurora Serverless v2 の大きな違いを知ることができます。読み取り DB インスタンスのサポートなどのアーキテクチャの違いにより、新しいタイプのユースケースが生まれます。

次の表を使用して、Aurora Serverless v2 と Aurora Serverless v1 の最も重要な違いを理解できます。

トピック

- Aurora Serverless v2 と Aurora Serverless v1 要件の比較
- Aurora Serverless v2 と Aurora Serverless v1 のスケーリングと可用性の比較
- Aurora Serverless v2 と Aurora Serverless v1 機能サポートの比較
- Aurora Serverless v1 ユースケースの Aurora Serverless v2 への適応

Aurora Serverless v2 と Aurora Serverless v1 要件の比較

Aurora Serverless v2 または Aurora Serverless v1 を使用して、データベースを実行するためのさまざまな要件の概要を次の表に示します。Aurora Serverless v2 では、Aurora Serverless v1 よりも上位バージョンの Aurora MySQL および Aurora PostgreSQL DB エンジンが提供されています。

機能	Aurora Serverless v2 の要件	Aurora Serverless v1 の要件
DB エンジン	Aurora MySQL、Aurora PostgreSQL	Aurora MySQL、Aurora PostgreSQL
サポートされている Aurora MySQL バージョン	「」を参照してくださいAurora MySQL での Aurora Serverless v2	「」を参照してくださいAurora MySQL での Aurora Serverless v1

機能	Aurora Serverless v2 の要件	Aurora Serverless v1 の要件
サポートされている Aurora PostgreSQL バージョン	「」を参照してくださいAurora PostgreSQL での Aurora Serverless v2	「」を参照してくださいAurora PostgreSQL での Aurora Serverless v1
DB クラスターのアップグレード	プロビジョニングされた DB クラスターと同様に、Aurora による DB クラスターのアップグレードを待つことなく手動でアップグレードを実行できます。詳細については、「Amazon Aurora DB クラスターの変更」を参照してください。 **Note** Aurora PostgreSQL 互換 DB クラスターの 13.x から 14.x または 15. x へのメジャーバージョンアップグレードを実行するには、クラスターの最大容量が 2 ACU 以上である必要があります。	マイナーバージョンアップグレードは、利用可能になると自動的に適用されます。詳細については、「Aurora Serverless v1 とAurora データベースエンジンのバージョン」を参照してください。 メジャーバージョンアップグレードは、手動で実行できます。詳細については、「Aurora Serverless v1 DB クラスターの変更」を参照してください。

機能	Aurora Serverless v2 の要件	Aurora Serverless v1 の要件
プロビジョニングされた DB クラスターからの変換	次の方法を使用します。 • 1 つまたは複数の Aurora Serverless v2 読み取り DB インスタンスを既存のプロビジョニングされたクラスターに追加します。ライターに Aurora Serverless v2 を使用する場合は、Aurora Serverless v2 DB インスタンスのいずれかにフェイルオーバーを実行します。クラスター全体で Aurora Serverless v2 DB インスタンスを使用する場合、Aurora Serverless v2 DB インスタンスをライターに昇格させた後、プロビジョニングされた書き込み DB インスタンスを削除してください。 • 適切な DB エンジンとエンジンバージョンの新しいクラスターを作成します。標準的な方法のいずれかを使用します。例えば、クラスタースナップショットを復元したり、既存のクラスターのクローンを作成できます。新しいクラスターの一部またはすべての DB インスタンスで Aurora Serverless v2 を選択します。	プロビジョニングされたクラスターのスナップショットをリストアして、新しい Aurora Serverless v1 クラスターを作成します。

機能	Aurora Serverless v2 の要件	Aurora Serverless v1 の要件
	クローン作成を使用して新しいクラスターを作成する場合、エンジンのバージョンを同時にアップグレードすることはできません。元のクラスターが、Aurora Serverless v2 と互換性があるエンジンバージョンを既に実行していることを確認します。	
Aurora Serverless v1 クラスターからの変換	「Aurora Serverless v1 クラスターから Aurora Serverless v2 クラスターへのアップグレード」の手順に従います。	該当しない
利用可能な DB インスタンスクラス	特別な DB インスタンスクラス `db.serverless` 。AWS Management Console では、サーバーレスと表記されています。	該当しない。Aurora Serverless v1 では `serverless` エンジンモードを使用します。
[ポート]	MySQL または PostgreSQL と互換性のあるポート	デフォルトの MySQL または PostgreSQL ポートのみ
パブリック IP アドレスは許可されていますか。	はい	いいえ
Virtual private Cloud (VPC) が必要ですか。	はい	はい。各 Aurora Serverless v1 クラスターは、VPC に割り当てられた 2 つのインターフェイスとゲートウェイロードバランサーのエンドポイントを消費します。

Aurora Serverless v2 と Aurora Serverless v1 のスケーリングと可用性の比較

Aurora Serverless v2 と Aurora Serverless v1 のスケーラビリティおよび可用性の違いを以下の表にまとめました。

Aurora Serverless v2 スケーリングは、Aurora Serverless v1 のスケーリングよりも応答性が高く、きめ細かく、破壊性が低くなります。Aurora Serverless v2 は、DB インスタンスのサイズを変更することでも、DB クラスターに DB インスタンスを追加することでも拡張できます。また、Aurora グローバルデータベースに他の AWS リージョン のクラスターを追加することでスケールすることもできます。対照的に、Aurora Serverless v1 はライターの容量を増減させるだけでスケールできます。Aurora Serverless v1 クラスターのすべてのコンピューティングは、1 つのアベイラビリティーゾーンと 1 つの AWS リージョン で実行されます。

スケーリングと高可用性の機能	Aurora Serverless v2 の動作	Aurora Serverless v1 の動作
最小 Aurora 容量ユニット (ACU) (Aurora MySQL)	0.5	クラスターが実行中の場合は 1、クラスターが一時停止しているときは 0。
最小 ACU (Aurora PostgreSQL)	0.5	クラスターが実行中の場合は 2、クラスターが一時停止しているときは 0。
最大 ACU (Aurora MySQL)	128	256
最 大ACU (Aurora PostgreSQL)	128	384
クラスターの停止	プロビジョニングされたクラスターと同じクラスター停止および開始機能を使用して、手動でクラスターを停止および開始できます。	クラスターは、タイムアウト後に自動的に一時停止します。アクティビティが再開すると、利用可能になるまでに少し時間がかかります。
DB インスタンスのスケーリング	0.5 ACU の最小増分でスケールアップおよびスケールダウン	ACU を倍増または半分にすることでスケールアップとスケールダウン

スケーリングと高可用性の機能	Aurora Serverless v2 の動作	Aurora Serverless v1 の動作
DB インスタンス数	プロビジョニングされたクラスターと同じ: 1 つの書き込み DB インスタンス、最大 15 の読み取り DB インスタンス。	読み取りと書き込みの両方を処理する 1 つの DB インスタンス。
SQL ステートメントの実行中にスケーリングが発生する可能性がありますか。	はい。Aurora Serverless v2 はクワイエットポイントを待つ必要はありません。	いいえ。例えば、スケーリングは、長時間実行されるトランザクション、一時テーブル、およびテーブルロックの完了を待ちます。
読み取り DB インスタンスはライターとともにスケーリングされます	オプション。	該当しません。
最大ストレージ	128 TiB	データベースエンジンとバージョンに応じて、128 TiB または 64 TiB。
スケーリング時にバッファキャッシュが保持されます	はい。バッファキャッシュは動的にサイズ変更されます。	いいえ。バッファキャッシュは、スケーリング後にリウォームされます。
フェイルオーバー	はい、プロビジョニングされたクラスターの場合と同じです。	ベストエフォートのみ。キャパシティーの可用性によります。Aurora Serverless v2 中より遅い。

スケーリングと高可用性の機能	Aurora Serverless v2 の動作	Aurora Serverless v1 の動作
マルチ AZ 機能	はい、プロビジョニングされた場合と同じです。マルチ AZ クラスターには、第 2 アベイラビリティーゾーン (AZ) 内の読み取り DB インスタンスが必要です。マルチ AZ クラスターの場合、Aurora は AZ に障害が発生した場合にマルチ AZ フェイルオーバーを実行します。	Aurora Serverless v1 クラスターは、すべてのコンピューティングを 1 つの AZ で実行します。AZ に障害が発生した場合の復旧はベストエフォートのみであり、キャパシティーの可用性によります。
Aurora Global Database	はい	いいえ
メモリプレッシャーに基づくスケーリング	はい	いいえ
CPU 負荷に基づくスケーリング	はい	はい
ネットワークトラフィックに基づくスケーリング	はい。ネットワークトラフィックのメモリと CPU オーバーヘッドに基づきます。`max_connections` パラメータは一定のままであり、スケールダウン時に接続がドロップされないようにします。	はい、接続の数に基づいています。
スケーリングイベントのタイムアウトアクション	いいえ	はい

スケーリングと高可用性の機能	Aurora Serverless v2 の動作	Aurora Serverless v1 の動作
AWS Auto Scaling で新しい DB インスタンスをクラスターに追加する	該当しません。Aurora Serverless v2 読み取り DB インスタンスを昇格階層 2 ~ 15 で作成し、低容量にスケールダウンしておくことができます。	いいえ。読み取り DB インスタンスは利用できません。

Aurora Serverless v2 と Aurora Serverless v1 機能サポートの比較

次の表に以下の内容がまとめてあります。

- Aurora Serverless v2 では使用できるが、Aurora Serverless v1 では使用できない機能
- Aurora Serverless v1 と Aurora Serverless v2 で異なる働きをする機能
- Aurora Serverless v2 では現在使用できない機能

Aurora Serverless v2 には、Aurora Serverless v1 では提供されていないプロビジョニングされたクラスターの機能が多く含まれています。

機能	Aurora Serverless v2 サポート	Aurora Serverless v1 サポート
クラスタートポロジ	Aurora Serverless v2 は個々の DB インスタンスのプロパティです。クラスターには複数の Aurora Serverless v2 DB インスタンス、または Aurora Serverless v2 とプロビジョニングされた DB インスタンスの組み合わせを含めることができます。	Aurora Serverless v1 クラスターは、DB インスタンスの概念を使用しません。クラスターの作成後は、Aurora Serverless v1 プロパティを変更できません。
設定パラメータ	プロビジョニングされたクラスターの場合とほぼ同じパラメータを変更できます	パラメータのサブセットのみを変更できます。

機能	Aurora Serverless v2 サポート	Aurora Serverless v1 サポート
	。詳細については、「Aurora Serverless v2 のパラメータグループを使用する」を参照してください。	
パラメータグループ	クラスターパラメータグループと DB パラメータグループ `SupportedEngineModes` 属性に `provisioned` 値を持つパラメータが使用できます。それは Aurora Serverless v1 よりもはるかに多くのパラメータです。	クラスターパラメータグループのみ。`SupportedEngineModes` 属性に `serverless` 値を持つパラメータが使用できます。
クラスターボリュームの暗号化	オプションです。	必須。Amazon Aurora の暗号化された DB クラスターの制限事項 での制限事項は、すべての Aurora Serverless v1 クラスターに適用されます。
クロスリージョンスナップショット	はい	スナップショットは、独自の AWS Key Management Service (AWS KMS) キーで暗号化する必要があります。
DB クラスターの削除後に保持される自動バックアップ	はい	いいえ
TLS/SSL	はい。サポートは、プロビジョニングされたクラスターの場合と同じです。使用に関する情報については、「Aurora Serverless v2 での TLS/SSL の使用」を参照してください。	はい。プロビジョニングされたクラスターの TLS サポートとはいくつかの違いがあります。使用に関する情報については、「Aurora Serverless v1 での TLS/SSL の使用」を参照してください。

機能	Aurora Serverless v2 サポート	Aurora Serverless v1 サポート
クローン作成	Aurora Serverless v2 と互換性がある DB エンジンバージョン間のみ。クローン作成を使用して、Aurora Serverless v1 またはプロビジョニングされたクラスターの以前のバージョンからアップグレードすることはできません。	Aurora Serverless v1 と互換性がある DB エンジンバージョン間のみ。
Amazon S3 との統合	はい	はい
AWS Secrets Manager との統合	いいえ	いいえ
S3 への DB クラススナップショットのエクスポート	はい	いいえ
IAM ロールの関連付け	はい	いいえ
Amazon CloudWatch への ログのアップロード	オプション。どのログをオンにするかと、CloudWatch にアップロードするログを選択します。	オンになっているすべてのログは CloudWatch に自動的にアップロードされます。
Data API が利用可能	はい	はい
クエリエディタが使用可能	はい	はい
Performance Insights	はい	いいえ
Amazon RDS Proxy が使用可能	はい	いいえ

機能	Aurora Serverless v2 サポート	Aurora Serverless v1 サポート
Babelfish for Aurora PostgreSQL が使用可能	はい。Babelfish と Aurora Serverless v2 の両方と互換性のある Aurora PostgreSQL バージョンでサポートされています。	いいえ

Aurora Serverless v1 ユースケースの Aurora Serverless v2 への適応

Aurora Serverless v1 のユースケースに応じて、次のように Aurora Serverless v2 機能を利用するようにそのアプローチを適応させることができます。

負荷の軽い Aurora Serverless v1 クラスターがあり、コストを最小限に抑えながら継続的な可用性を維持することを優先するとします。Aurora Serverless v2 では、Aurora Serverless v1 の最小 ACU が 1 であるのに対し、0.5 と小さく設定することが可能です。マルチ AZ 設定を作成することで、可用性を高めることができます。読み取り DB インスタンスの最小値は 0.5 ACU です。

開発およびテストシナリオで使用する Aurora Serverless v1 クラスターがあるとします。この場合、コストも高い優先度になりますが、クラスターを常時利用可能にする必要はありません。現在、Aurora Serverless v2 はクラスターが完全にアイドル状態になったときに自動的に一時停止することはありません。代わりに、不要になったときにクラスターを手動で停止し、次のテストまたは開発サイクルの時間になったときにクラスターを開始できます。

重いワークロードを持つ Aurora Serverless v1 クラスターがあるとします。Aurora Serverless v2 を使用する同等のクラスターは、より詳細にスケールすることができます。例えば、Aurora Serverless v1 は、容量を 2 倍にすることでスケールします (例えば 64 ACU から 128 ACU)。反対に、Aurora Serverless v2 DB インスタンスは、これらの数字の中間の値にスケールできます。

ワークロードが、Aurora Serverless v1 で使用可能な容量より大きな総容量を必要とするとします。複数の Aurora Serverless v2 読み取り DB インスタンスを使用して、書き込み DB インスタンスからワークロードの読み取り負荷の高い部分をオフロードすることができます。読み取り負荷の高いワークロードを複数の読み取り DB インスタンスに分割することもできます。

書き込みが多いワークロードの場合、大規模なプロビジョニングされた DB インスタンスをライターとして、クラスターを設定することができます。1 つまたは複数の Aurora Serverless v2 リーダー DB インスタンスと一緒に行うことができます。

Aurora Serverless v1 クラスターから Aurora Serverless v2 クラスターへのアップグレード

DB クラスターを Aurora Serverless v1 から Aurora Serverless v2 にアップグレードするプロセスには、複数のステップが必要です。Aurora Serverless v1 から Aurora Serverless v2 に直接変換できないためです。Aurora Serverless v1 DB クラスターをプロビジョニングされたクラスターに変換するには、常に中間ステップが伴います。

Aurora MySQL 互換 DB クラスター

Aurora Serverless v1 DB クラスターをプロビジョニングされた DB クラスターに変換し、ブルー/グリーンデプロイを使用してアップグレードして、Aurora Serverless v2 DB クラスターに変換できます。本番環境では、この手順をお勧めします。詳細については、「データベース更新のために Amazon RDS ブルー/グリーンデプロイを使用する」を参照してください。

ブルー/グリーンデプロイを使用して Aurora MySQL バージョン 2 を実行する Aurora Serverless v1 クラスター (MySQL 5.7 と互換性あり) にアップグレードするには

1. Aurora Serverless v1 DB クラスターをプロビジョニングされた Aurora MySQL バージョン 2 クラスターに変換します。「Aurora Serverless v1 からプロビジョン済みに変換する」の手順に従います。
2. ブルー/グリーンデプロイを作成する。「ブルー/グリーンデプロイの作成」の手順に従います。
3. 3.04.1 など、Aurora Serverless v2 と互換性のあるグリーンクラスターの Aurora MySQL バージョンを選択します。

   互換性のあるバージョンについては、「Aurora MySQL での Aurora Serverless v2」を参照してください。
4. グリーンクラスターのライター DB インスタンスを変更し、サーバーレス v2 (db.serverless) DB インスタンスクラスを使用するようにします。

   詳細については、「プロビジョン済みライターまたはリーダーを Aurora Serverless v2 に変換する」を参照してください。
5. アップグレードした Aurora Serverless v2 DB クラスターが使用可能になったら、ブルークラスターからグリーンクラスターに切り替えます。

Aurora PostgreSQL 互換 DB クラスター

Aurora Serverless v1 DB クラスターをプロビジョニングされた DB クラスターに変換し、ブルー/グリーンデプロイを使用してアップグレードして、Aurora Serverless v2 DB クラスターに変換できます。本番環境では、この手順をお勧めします。詳細については、「データベース更新のために Amazon RDS ブルー/グリーンデプロイを使用する」を参照してください。

ブルー/グリーンデプロイを使用するには、Aurora PostgreSQL バージョン 11 を実行している Aurora Serverless v1 クラスターをアップグレードするには

1. Aurora Serverless v1 DB クラスターをプロビジョニングされた Aurora PostgreSQL クラスターに変換します。「Aurora Serverless v1 からプロビジョン済みに変換する」の手順に従います。
2. ブルー/グリーンデプロイを作成する。「ブルー/グリーンデプロイの作成」の手順に従います。
3. グリーンクラスターを Aurora Serverless v2 と互換性がある Aurora PostgreSQL バージョン (例えば 15.3) にアップグレードします。

   互換性のあるバージョンについては、「Aurora PostgreSQL での Aurora Serverless v2」を参照してください。
4. グリーンクラスターのライター DB インスタンスを変更し、サーバーレス v2 (db.serverless) DB インスタンスクラスを使用するようにします。

   詳細については、「プロビジョン済みライターまたはリーダーを Aurora Serverless v2 に変換する」を参照してください。
5. アップグレードした Aurora Serverless v2 DB クラスターが使用可能になったら、ブルークラスターからグリーンクラスターに切り替えます。

また、Aurora Serverless v1 DB クラスターを Aurora PostgreSQL バージョン 11 からバージョン 13 に直接アップグレードし、プロビジョニングされた DB クラスターに変換してから、プロビジョニングされたクラスターを Aurora Serverless v2 DB クラスターに変換することもできます。

アップグレードしてから、Aurora PostgreSQL バージョン 11 を実行している Aurora Serverless v1 クラスターに変換するには

1. Aurora Serverless v1 クラスターを Aurora Serverless v2 と互換性がある Aurora PostgreSQL バージョン 13 (例えば 13.12) にアップグレードします。「メジャーバージョンをアップグレードする」の手順に従います。

互換性のあるバージョンについては、「Aurora PostgreSQL での Aurora Serverless v2」を参照してください。

2. Aurora Serverless v1 DB クラスターをプロビジョニングされた Aurora PostgreSQL クラスターに変換します。「Aurora Serverless v1 からプロビジョン済みに変換する」の手順に従います。
3. Aurora Serverless v2 リーダー DB インスタンスを DB クラスターに追加します。詳細については、「Aurora Serverless v2 リーダーの追加」を参照してください。
4. Aurora Serverless v2 DB インスタンスにフェイルオーバーします。
    a. DB クラスターのライター DB インスタンスを選択します。
    b. [アクション] で、[フェイルオーバー] を選択します。
    c. 確認ページで、[フェイルオーバー] を選択します。

Aurora PostgreSQL バージョン 13 を実行している Aurora Serverless v1 DB クラスターの場合、Aurora Serverless v1 クラスターをプロビジョニングされた DB クラスターに変換してから、プロビジョニングされたクラスターを Aurora Serverless v2 DB クラスターに変換します。

Aurora PostgreSQL バージョン 13 を実行している Aurora Serverless v1 クラスターをアップグレードするには

1. Aurora Serverless v1 DB クラスターをプロビジョニングされた Aurora PostgreSQL クラスターに変換します。「Aurora Serverless v1 からプロビジョン済みに変換する」の手順に従います。
2. Aurora Serverless v2 リーダー DB インスタンスを DB クラスターに追加します。詳細については、「Aurora Serverless v2 リーダーの追加」を参照してください。
3. Aurora Serverless v2 DB インスタンスにフェイルオーバーします。
    a. DB クラスターのライター DB インスタンスを選択します。
    b. [アクション] で、[フェイルオーバー] を選択します。
    c. 確認ページで、[フェイルオーバー] を選択します。

オンプレミスデータベースから Aurora Serverless v2 に移行する

プロビジョニングされた Aurora MySQL や Aurora PostgreSQL と同様に、オンプレミスのデータベースを Aurora Serverless v2 に移行できます。

- MySQL データベースでは、`mysqldump` コマンドを使用できます。詳細については、「ダウンタイムを抑えて MySQL または MariaDB DB インスタンスにデータをインポートする」を参照してください。
- PostgreSQL データベースでは、`pg_dump` および `pg_restore` コマンドを使用できます。詳細については、ブログ記事「Best practices for migrating PostgreSQL databases to Amazon RDS and Amazon Aurora」(PostgreSQL データベースを Amazon RDS と Amazon Aurora に移行するためのベストプラクティス) を参照してください。

Amazon Aurora Serverless v1 の使用

Amazon Aurora Serverless v1 (Amazon Aurora サーバーレスバージョン 1) は、Amazon Aurora 用のオンデマンドオートスケーリング構成です。Aurora Serverless v1 DB クラスターでは、クラスターのコンピューティング容量を、アプリケーションのニーズに対応して増減させられます。これは、ユーザーにより容量が手動で管理される、Aurora のプロビジョニングされた DB クラスターとは対照的です。Aurora Serverless v1 では、頻度が低く、断続的、または予測が困難なワークロードを処理するための、比較的シンプルかつコスト効率の高いオプションを提供しています。このコスト効率が良いのは、アプリケーションの使用状況に合わせて自動的に起動してコンピューティング性能をスケールし、不使用時にはシャットダウンするためです。

料金の詳細については、「Amazon Aurora pricing」ページの「MySQL 互換エディション」または「PostgreSQL 互換エディション」の「サーバーレスの料金」を参照してください。

Aurora Serverless v1 クラスターには、プロビジョンド DB クラスターで使用されているのと同じタイプの、大容量かつ分散型で可用性の高いストレージボリュームが備わっています。

Aurora Serverless v2 クラスターの場合、クラスターボリュームを暗号化するかどうかを選択できます。

Aurora Serverless v1 クラスターでは、クラスターボリュームは常に暗号化されます。暗号化キーは選択できますが、暗号化を無効にすることはできません。つまり、暗号化されたスナップショットに対して実行できる操作と同じ操作を、Aurora Serverless v1 でも実行できるということです。詳細については、「Aurora Serverless v1 とスナップショット」を参照してください。

トピック

- Aurora Serverless v1 DB クラスターの変更
- Aurora Serverless v1 DB クラスターの容量を手動でスケーリングする
- Aurora Serverless v1 DB クラスターの表示
- Aurora Serverless v1 DB クラスターの削除
- Aurora Serverless v1 とAurora データベースエンジンのバージョン

> **Important**
>
> Aurora には、Aurora Serverless v2 および Aurora Serverless v1 の 2 世代のサーバーレステクノロジーがあります。MySQL 8.0 または PostgreSQL 13 上でアプリケーションを実行できる場合は、Aurora Serverless v2 を使用することをお勧めします。Aurora Serverless v2 は、より速く、よりきめ細かい方法でスケーリングします。また Aurora Serverless v2 は読み取り DB インスタンスなど、他の Aurora の機能との互換性がより高くなっています。したがって、既に Aurora に慣れている場合は、Aurora Serverless v1 の場合ほど Aurora Serverless v2 を使用するために、多くの新しい手順や制限を学ぶ必要はありません。Aurora Serverless v2 を使用する では Aurora Serverless v2 について学ぶことができます。

リージョンとバージョンの可用性

利用できる機能とそのサポートは、各 Aurora データベースエンジンの特定のバージョン、および AWS リージョン によって異なります。Aurora と Aurora Serverless v1 でのバージョンとリージョン可用性については、「Aurora Serverless v1 でサポートされているリージョンと Aurora DB エンジン」を参照してください。

Aurora Serverless v1 の利点

Aurora Serverless v1 では、次のような利点が得られます。

- プロビジョニングよりもシンプル - Aurora Serverless v1 では、DB インスタンスや容量を管理する上での複雑さが、大幅に軽減されます。
- スケーラブル - Aurora Serverless v1 は、必要に応じてシームレスにコンピューティング性能とメモリ容量をスケールします。これに伴うクライアント接続の中断はありません。
- 高いコスト効率 - Aurora Serverless v1 の使用料金は、データベースリソースの秒単位の消費分に対してのみ発生します。

- 可用性の高いストレージ - Aurora Serverless v1 では、Aurora と同じ 6 ウェイレプリケーションを使用した耐障害性の高い分散型ストレージシステムを使用することで、データを損失から守っています。

Aurora Serverless v1 のユースケース

Aurora Serverless v1 は、以下のユースケース用に設計されています。

- 利用頻度の低いアプリケーション - 1 日または週に数回、それぞれ数分のみ使用される (低ボリュームのブログサイトなどの) アプリケーション。Aurora Serverless v1 では、消費したデータベースリソース分のみの料金を、秒単位でお支払いいただきます。
- 新しいアプリケーション - 現在デプロイ中で、必要とされるインスタンスサイズが明確でない、新しいアプリケーション。Aurora Serverless v1 を使用することで、データベースのエンドポイントが作成できます。データベースの容量は、アプリケーションの要件に応じて自動的にスケーリングされます。
- 可変ワークロード - 使用時間のピークが 30 分 ~ 数時間ほどで、それが 1 日もしくは 1 年に数回発生するような、負荷が重くないアプリケーション。人事管理、予算作成、運営報告用のアプリケーションなどが該当します。Aurora Serverless v1 を導入することで、ピーク容量や平均容量に合わせてプロビジョニングする必要はなくなります。
- 予測が困難なワークロード - 毎日実行され、突然、想定し得ないアクティビティの増加が発生するようなワークロード。雨が降り出したときにアクティビティが急増 (サージ) するトラフィックサイトなどが該当します。Aurora Serverless v1 では、アプリケーションのピーク時の負荷要件に合わせて、データベースの容量がオートスケーリングされ、アクティビティのサージが終了した時点でスケールバックされます。
- 開発およびテスト中のデータベース - 勤務時間中にデベロッパーにより使用されるものの、夜間や週末には必要とされないデータベース。Aurora Serverless v1 では、使用されていないデータベースは、自動的にシャットダウンされます。
- マルチテナントアプリケーション - Aurora Serverless v1 を使用することで、フリート内のアプリケーションごとのデータベース容量を、ユーザーが個別に管理する必要はなくなります。個々のデータベース容量は、Aurora Serverless v1 により自動的に管理されます。

Aurora Serverless v1 の制約事項

Aurora Serverless v1 には以下の制限が適用されます。

- Aurora Serverless v1 では、以下の機能はサポートされていません。
  - Aurora Global Database
  - Aurora レプリカ
  - AWS Identity and Access Management (IAM) データベース認証
  - Aurora でのバックトラック
  - データベースアクティビティストリーミング
  - Kerberos 認証
  - Performance Insights
  - RDS Proxy
  - AWS Management Console でログを表示する
- 1 日以上開かれたままにされた Aurora Serverless v1 DB クラスターへの接続は、自動的に閉じられます。
- すべての Aurora Serverless v1 DB クラスターには、次の制限があります。
  - Aurora Serverless v1 スナップショットを Amazon S3 バケットにエクスポートすることはできません。
  - AWS Database Migration Service DB クラスターでは、Aurora Serverless v1 および変更データキャプチャ (CDC) を使用することはできません。プロビジョニングされた Aurora DB クラスターのみがソースとしての AWS DMS と CDC をサポートします。
  - データを Amazon S3 に置かれているテキストファイルに保存したり、データを S3 から Aurora Serverless v1 に置かれているテキストファイルに読み込んだりすることはできません。
  - IAM ロールを Aurora Serverless v1 DB クラスターにアタッチすることはできません。ただし、Aurora Serverless v1 関数で `aws_s3` パラメータを指定しながら `aws_s3.table_import_from_s3` エクステンションを使用すれば、`credentials` から Amazon S3 へのデータのロードが可能です。詳細については、「Amazon S3 から Aurora PostgreSQL DB クラスターにデータをインポートする」を参照してください。
  - クエリエディターを使用すると、データベースにアクセスするための DB 認証情報用に Secrets Manager シークレットが作成されます。クエリエディターから認証情報を削除すると、関連するシークレットも Secrets Manager から削除されます。シークレットは削除後に回復することはできません。
- Aurora Serverless v1 が実行されている Aurora MySQL ベースの DB クラスターでは、以下はサポートされません。
  - Aurora MySQL DB クラスター内からの AWS Lambda 関数の呼び出し。ただし、AWS Lambda 関数は Aurora Serverless v1 DB クラスターを呼び出すことができます。

  - Aurora MySQL もしくは RDS for MySQL ではない DB インスタンスからのスナップショットの復元。
  - バイナリログ (binlog) に基づくレプリケーションを使用したデータのレプリケート。この制限は、Aurora MySQL ベースの DB クラスター (Aurora Serverless v1) が、レプリケーションのソースであるかターゲットであるかに関係なく適用されます。Amazon EC2 で実行されているインスタンスなど、Aurora の外部にある MySQL DB インスタンスから、データを Aurora Serverless v1 DB クラスターにレプリケートする場合は、AWS Database Migration Service が使用できます。詳細については、AWS Database Migration Service ユーザーガイドをご参照ください。
  - ホストベースのアクセスによるユーザーの作成 (`'username'@'IP_address'`)。これにより、Aurora Serverless v1 ではクライアントとデータベースホスト間のルーターフリートを使用し、シームレスなスケーリングを実現します。Aurora Serverless DB クラスターが認識する IP アドレスは、クライアントのものではなくルーターホストのものです。詳細については、「Aurora Serverless v1 アーキテクチャ」を参照してください。

    代わりに、ワイルドカードを使用します (`'username'@'%'`)。
- Aurora Serverless v1 を実行している Aurora PostgreSQL ベースの DBクラスターには、次の制限があります。
  - Aurora PostgreSQL クエリプラン管理 (`apg_plan_management` エクステンション) はサポートされていません。
  - Amazon RDS PostgreSQL と Aurora PostgreSQL で使用可能な論理的なレプリケーション機能はサポートされていません。
  - Amazon RDS for PostgreSQL エクステンションで有効になっているタイプのアウトバウンド通信はサポートされていません。例えば、`postgres_fdw/dblink` エクステンションを使用して外部データにアクセスすることはできません。RDS PostgreSQL エクステンションの詳細については、RDS ユーザーガイドの、「Amazon RDS 上の PostgreSQL」を参照してください。
  - 現在、一部の SQL クエリとコマンドの使用が推奨されていません。これには、セッションレベルのアドバイザリロック、テンポラリリレーション、非同期通知 (`LISTEN`)、および WITH HOLD (`DECLARE name ... CURSOR WITH HOLD FOR query`) が指定されたカーソルが含まれます。また、`NOTIFY` コマンドはスケーリングを阻害するため、推奨されません。

    詳細については、「Aurora Serverless v1 のオートスケーリング」を参照してください。
- Aurora Serverless v1 DB クラスターの優先自動バックアップ期間を設定することはできません。
- Aurora Serverless v1 DB クラスターのメンテナンス期間を設定できます。詳細については、「DB クラスターの適切なメンテナンスウィンドウの調整」を参照してください。

Aurora Serverless v1 での設定の要件

Aurora Serverless v1 DB クラスターを作成する際には、次の要件に注意してください。

- DB エンジンごとに、次に挙げる特定のポート番号を使用します。
  - Aurora MySQL - 3306
  - Aurora PostgreSQL - 5432
- Virtual Private Cloud (VPC) に置かれる Aurora Serverless v1 DB クラスターは、Amazon VPC サービスをベースに作成します。VPC で Aurora Serverless v1 DB クラスターを作成すると、50 個のインターフェイスエンドポイントとゲートウェイロードバランサーエンドポイントの 2 つを VPC に割り当て、その 2 つを使用します。これらのエンドポイントは自動的に作成されます。クォータを増やすには、AWS Support にご連絡ください。詳細については、Amazon VPC クォータを参照してください。
- Aurora Serverless v1 DB クラスターにパブリック IP アドレスを割り当てることはできません。VPC 内からのみ、Aurora Serverless v1DB クラスターへのアクセスが可能です。
- Aurora Serverless v1 DB クラスターで使用する DB サブネットグループでは、異なるアベイラビリティーゾーンごとにサブネットを作成します。つまり、同じアベイラビリティーゾーンに複数のサブネットを持つことはできません。
- Aurora Serverless v1 DB クラスターで使用されるサブネットグループに対する変更は、クラスターには適用されません。
- Aurora Serverless v1 DB クラスターには AWS Lambda からアクセスできます。このアクセスのためには、Aurora Serverless v1 DB クラスターと同じ VPC で実行されるように、Lambda 関数を設定する必要があります。AWS Lambda の使用の詳細については、AWS Lambda デベロッパーガイドの「Amazon VPC 内のリソースにアクセスするための Lambda 関数の設定」を参照してください。

Aurora Serverless v1 での TLS/SSL の使用

Aurora Serverless v1 では、クライアントと Aurora Serverless v1 DB クラスター間の通信に対し、Transport Layer Security/Secure Sockets Layer (TLS/SSL) プロトコルを使用した暗号化が、デフォルトで使用されます。サポートされる TLS/SSL バージョンは、1.0、1.1、および 1.2 です。TLS/SSL を使用するために、Aurora Serverless v1 DB クラスターを設定する必要はありません。

ただし、次のような制限があります。

- Aurora Serverless v1 DB クラスターに対する TLS/SSL サポートは、現在、中国 (北京) の AWS リージョン ではご利用いただけません。
- Aurora MySQL ベースの Aurora Serverless v1 DB クラスターでデータベースユーザーを作成する際は、SSL アクセス許可に REQUIRE 句を使用しないでください。これを使用すると、ユーザーが Aurora DB インスタンスに接続できなくなります。
- MySQL クライアントと PostgreSQL クライアントのユーティリティの両方に関して (クライアントと Aurora Serverless v1 の間で TLS/SSL を使用している場合に)、他の環境では使用できるセッション可変の中で機能しないものが存在します。
- 現在、MySQL クライアントが TLS/SSL の VERIFY_IDENTITY モードを使用して接続する場合は、MySQL 8.0 互換の mysql コマンドを使用する必要があります。詳細については、「MySQL データベースエンジンを実行している DB インスタンスへの接続」を参照してください。

Aurora Serverless v1 DB クラスターに接続するクライアントによっては、暗号化された接続を確立するために、TLS/SSL を指定する必要がない場合があります。例えば、PostgreSQL クライアントを使用して、Aurora PostgreSQL 互換エディションを実行中の Aurora Serverless v1 DB クラスターに接続する場合は、通常どおりの接続が可能です。

```
psql -h endpoint -U user
```

パスワードを入力すると、PostgreSQL クライアントにより、TLS/SSL のバージョンと暗号を含む接続の詳細が表示されます。

```
psql (12.5 (Ubuntu 12.5-0ubuntu0.20.04.1), server 10.12)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256,
 compression: off)
Type "help" for help.
```

Important

Aurora Serverless v1 は、クライアントアプリケーションによって SSL/TLS が無効になっていない限り、デフォルトで Transport Layer Security/Secure Sockets Layer (TLS/SSL) プロトコルを使用して接続を暗号化します。TLS/SSL 接続はルーターフリートで終了します。ルーターフリートと Aurora Serverless v1 DB クラスター間の通信は、サービスの内部ネットワーク境界内で行われます。
クライアント接続のステータスを確認して、Aurora Serverless v1 への接続が TLS/SSL 暗号化されているかどうかを調べることができます。PostgreSQL pg_stat_ssl と

pg_stat_activity テーブル、およびその ssl_is_used 関数は、クライアントアプリケーションと Aurora Serverless v1 間の通信の TLS/SSL 状態を表示しません。同様に、TLS/SSL 状態は MySQL status ステートメントから派生できません。PostgreSQL の Aurora クラスターパラメータ force_ssl および MySQL の require_secure_transport は Aurora Serverless v1 では正式にサポートされていませんでした。これらのパラメータは、Aurora Serverless v1 に使用可能になっています。Aurora Serverless v1 でサポートされているパラメータの完全なリストについては、DescribeEngineDefaultClusterParameters API オペレーションを呼び出してください。パラメータグループと Aurora Serverless v1 の詳細については、Aurora Serverless v1 のパラメータグループ を参照してください。

MySQL クライアントを使用して Aurora MySQL 互換エディションを実行中の Aurora Serverless v1 DB クラスターに接続するには、リクエスト内で TLS/SSL を指定します。次の例には、Amazon Trust Services からダウンロードされた Amazon ルート CA 1 信頼ストアが含まれています。この信頼ストアは、この接続を正常に実行するために必要です。

```
mysql -h endpoint -P 3306 -u user -p --ssl-ca=amazon-root-CA-1.pem --ssl-mode=REQUIRED
```

プロンプトが表示されたら、パスワードを入力します。すぐに MySQL モニタが開きます。status コマンドを使用すると、セッションが暗号化されていることを確認できます。

```
mysql> status
--------------
mysql  Ver 14.14 Distrib 5.5.62, for Linux (x86_64) using readline 5.1
Connection id:          19
Current database:
Current user:           ***@*******
SSL:                    Cipher in use is ECDHE-RSA-AES256-SHA
...
```

MySQL クライアントを使用しての Aurora MySQL データベースへの接続の詳細については、「MySQL データベースエンジンを実行している DB インスタンスへの接続」を参照してください。

Aurora Serverless v1 では、MySQL クライアント (mysql) および PostgreSQL クライアント (psql) で使用できるすべての TLS/SSL モードがサポートされます。これらには、次の表に示すものも含まれます。

TLS/SSL モードの説明	mysql	psql
TLS/SSL を使用せずに接続します。	DISABLED	無効化
初期に TLS/SSL を使用して接続を試みますが、必要に応じて非 SSL にフォールバックします。	PREFERRED	優先 (デフォルト)
強制的に TLS/SSL を使用します。	REQUIRED	require
TLS/SSL を強制的に使用し CA を確認します。	VERIFY_CA	verify-ca
TLS/SSL を強制的に使用し CA を確認し、また CA ホスト名を確認します。	VERIFY_IDENTITY	verify-full

Aurora Serverless v1 は、ワイルドカード証明書を使用します。TLS/SSL を使用するときに "verify CA" または "verify CA and CA hostname" オプションを指定した場合は、まず Amazon ルート CA 1 信頼ストアを Amazon Trust Services からダウンロードしてください。ダウンロードした、この PEM 形式のファイルは、クライアントコマンドにより識別できます。PostgreSQL クライアントを使用してこれを行う場合:

Linux、macOS、Unix の場合:

```
psql 'host=endpoint user=user sslmode=require sslrootcert=amazon-root-CA-1.pem
 dbname=db-name'
```

Postgres クライアントを使用した Aurora PostgreSQL データベースの操作の詳細については、「PostgreSQL データベースエンジンを実行する DB インスタンスへの接続」を参照してください。

Aurora DB クラスターへの接続全般の詳細については、「Amazon Aurora DB クラスターへの接続」を参照してください。

Aurora Serverless v1 DB クラスターへの接続用にサポートされている暗号スイート

設定可能な暗号スイートを使用すると、データベース接続のセキュリティをより詳細に制御できます。データベースへのクライアント TLS/SSL 接続を保護するために許可する暗号スイートのリストを指定できます。設定可能な暗号スイートを使用すると、データベースサーバーが受け入れる接続暗号化を制御できます。これにより、安全でない暗号や使用されなくなった暗号の使用を防ぐことができます。

Aurora MySQL に基づく Aurora Serverless v1 DB クラスターは、Aurora MySQL プロビジョン済み DB クラスターと同じ暗号スイートをサポートします。これらの暗号スイートについては、「Aurora PostgreSQL DB クラスターへの接続用暗号スイートを設定する」を参照してください。

Aurora PostgreSQL に基づく Aurora Serverless v1 DB クラスターは、暗号スイートをサポートしていません。

Aurora Serverless v1 の働き

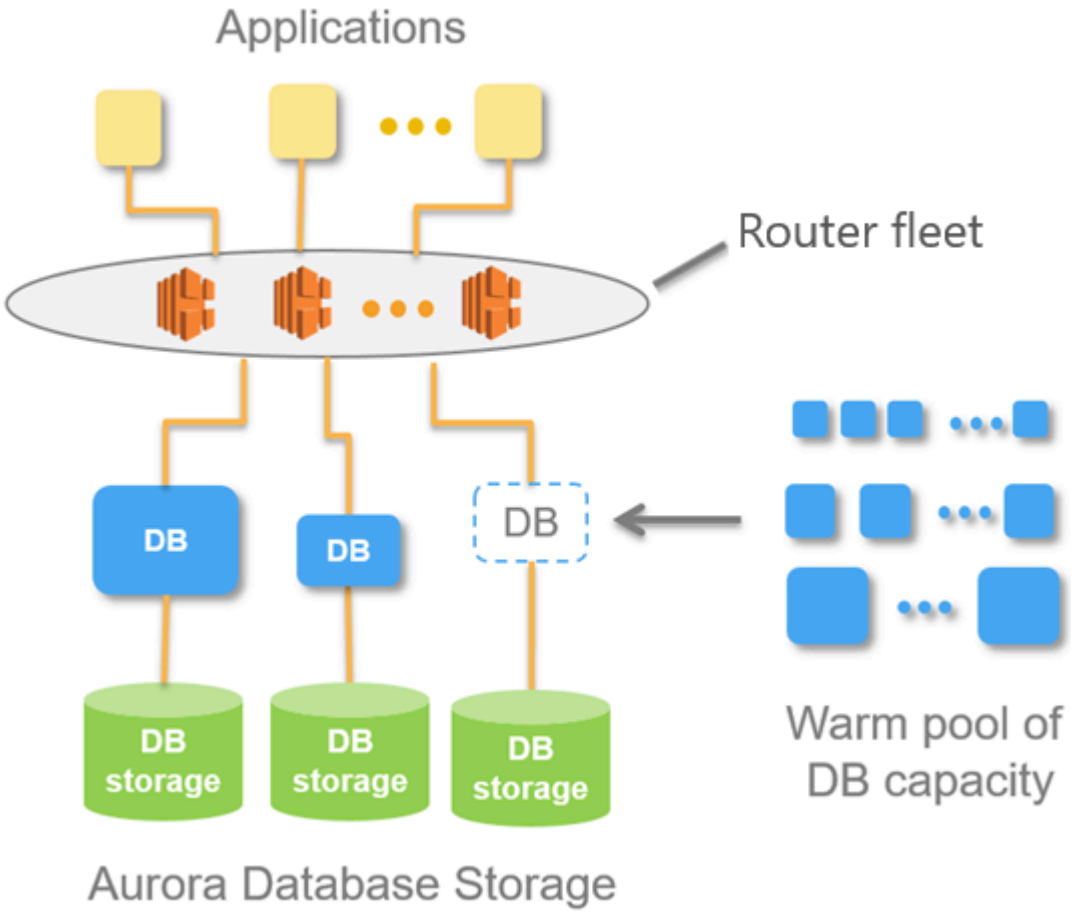
Aurora Serverless v1 がどのように機能しているのかがわかります。

トピック

- Aurora Serverless v1 アーキテクチャ
- Aurora Serverless v1 のオートスケーリング
- 容量の変更のタイムアウトアクション
- Aurora Serverless v1 の一時停止と再開
- Aurora Serverless v1 のデータベース接続の最大数を確認する
- Aurora Serverless v1 のパラメータグループ
- Aurora Serverless v1 のログ記録
- Aurora Serverless v1 とメンテナンス
- Aurora Serverless v1 とフェイルオーバー
- Aurora Serverless v1 とスナップショット

Aurora Serverless v1 アーキテクチャ

次のイメージは、Aurora Serverless v1 アーキテクチャの概要を示しています。



データベースサーバーをプロビジョニングして管理する代わりに、Aurora 容量ユニット (ACU) を指定します。各 ACU では、約 2 GB (GB) のメモリ、対応する CPU、およびネットワーキングが組み合わせられています。データベースストレージは、自動的に 10 ギビバイト (GiB) から 128 tebibytes (TiB) にスケールします (スタンダード Aurora DB クラスターのストレージと同じです)。

最小と最大の ACU を指定できます。Aurora の最小容量ユニットは、DB クラスターをスケールダウンできる最小の ACU です。Aurora の最大容量ユニットは、DB クラスターをスケールアップできる最大の ACU です。設定に応じて、CPU 使用率、接続、および使用可能メモリの各しきい値に関するスケーリングルールが、Aurora Serverless v1 により自動的に作成されます。

Aurora Serverless v1 は、AWS リージョン のリソースのウォームプールを管理し、スケーリング時間を最小限に短縮します。Aurora Serverless v1 は、Aurora DB クラスターに新しいリソースを追加する際、ルーターフリートを使用してアクティブなクライアント接続を新しいリソースに切り替えます。いつの時点においても、Aurora DB クラスターで実際に使用している ACU 分のみ課金されます。

Aurora Serverless v1 のオートスケーリング

Aurora Serverless v1 DB クラスターに割り当てられる容量は、クライアントアプリケーションによって生成された負荷に対応して、シームレスに拡大および縮小されます。ここでは、負荷は CPU 使用率と接続数です。これらのいずれかによって容量が制約されると、Aurora Serverless v1 がスケールアップします。 また、スケールアップすることで解決できるパフォーマンスの問題を検出すると、Aurora Serverless v1 もスケールアップします。

Aurora Serverless v1 で AWS Management Console クラスターのスケーリングイベントを表示できます。オートスケーリングの実行中に、Aurora Serverless v1 により、`EngineUptime` メトリクスがリセットされます。リセットメトリクス値の値は、シームレススケーリングに問題があったことを意味するものでも、Aurora Serverless v1 が接続を切断したことを意味するものでもありません。これは、新しい容量での稼働時間のスタート点に過ぎません。詳細については、「Amazon Aurora クラスターでのメトリクスのモニタリング」を参照してください。

Aurora Serverless v1 DB クラスターにアクティブな接続がない場合、容量ゼロ (0 ACU) にスケールダウンできます。詳細については、「Aurora Serverless v1 の一時停止と再開」を参照してください。

スケーリング操作を実行する必要がある場合は、Aurora Serverless v1 は初期にスケーリングポイントを特定しようとします。これは、クエリが処理されていない瞬間です。Aurora Serverless v1 は、次の理由により、スケーリングポイントを見つけることができない場合があります。

- クエリの実行時間が長い
- トランザクションが進行中である
- テンポラリテーブルまたはテーブルがロックされている

スケーリングポイントを見つけるときに Aurora Serverless v1 DB クラスターの成功率を上げるには、長時間実行されるクエリや長時間実行されるトランザクションを回避することをお勧めします。スケーリングをブロックするオペレーションとそれらを回避する方法の詳細については、「Aurora Serverless v1 を使用するためのベストプラクティス」を参照してください。

デフォルトでは、Aurora Serverless v1 は、スケーリングポイントを 5 分 (300 秒) で見つけようとします。クラスターの作成時または変更時に、別のタイムアウト期間を指定できます。タイムアウト期間は、60 秒 ~ 10 分 (600 秒) の間で指定できます。Aurora Serverless v1 が、指定した期間以内にスケーリングポイントが見つからない場合、オートスケーリング操作がタイムアウトします。

デフォルトでは、オートスケーリングがタイムアウトする前にスケーリングポイントを見つけられない場合、Aurora Serverless v1 はクラスターを現在の容量で維持します。[Force the capacity change]

(容量の変更を強制する) オプションを選択して Aurora Serverless v1 DB クラスターを作成または変更するときに、このデフォルトの動作を変更できます。詳細については、「容量の変更のタイムアウトアクション」を参照してください。

容量の変更のタイムアウトアクション

スケーリングポイントが見つからずにオートスケーリングがタイムアウトした場合、デフォルトでは Aurora は現在の容量を維持します。[Force the capacity change] (容量の変更を強制する) オプションを有効にすることで、Aurora で変更を強制することを選択できます。このオプションは、クラスターを作成するときに、[Create database] (データベースの作成) ページの [Autoscaling timeout and action] (オートスケーリングのタイムアウトとアクション) セクションで使用できます。

デフォルトでは、[Force the capacity change] (容量を強制的に変更する) オプションは選択解除されています。スケーリングポイントが見つからずにスケーリング操作がタイムアウトした場合に Aurora Serverless v1 DB クラスターの容量を変更しないようにするには、このオプションをオフのままにします。

このオプションを選択すると、Aurora Serverless v1 DB クラスターを使用して、スケーリングポイントがない場合でも、容量変更を強制します。このオプションを選択する前に、次の選択の結果に注意してください。

- 処理中のトランザクションはすべて中断され、次のエラーメッセージが表示されます。

  Aurora MySQL バージョン 2 — ERROR 1105 (HY000): 最後のトランザクションがシームレススケーリングのため中止されました。もう一度試してください。

  このトランザクションは、Aurora Serverless v1 DB クラスターが利用できるようになりしだい、再送信できます。
- テンポラリテーブルとロックへの接続は削除されます。

  アプリケーションが切断された接続または不完全なトランザクションから回復できる場合にのみ、[Force the capacity change] (容量を強制的に変更する) オプションを選択することをお勧めします。

Aurora Serverless v1 DB クラスターを作成するときに AWS Management Console で行った選択は、`SecondsBeforeTimeout` および `TimeoutAction`プロパティの `ScalingConfigurationInfo` オブジェクトに格納されます。クラスターを作成すると、`TimeoutAction` プロパティの値は次のいずれかの値に設定されます。

- `RollbackCapacityChange` - この値は、[Roll back the capacity change] (容量の変更をロールバックする) オプションを選択したときに設定されます。これがデフォルトの動作です。
- `ForceApplyCapacityChange` - この値は、[Force the capacity change] (容量の変更を強制する) オプションを選択したときに設定されます。

次に示すように、describe-db-clusters Aurora Serverless v1 コマンドを使用して、既存の AWS CLI DB クラスターでこのプロパティの値を取得できます。

Linux、macOS、Unix の場合:

```
aws rds describe-db-clusters --region region \
  --db-cluster-identifier your-cluster-name \
  --query '*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}'
```

Windows の場合:

```
aws rds describe-db-clusters --region region ^
  --db-cluster-identifier your-cluster-name ^
  --query "*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}"
```

例として、米国西部 (北カリフォルニア) リージョンにある、`west-coast-sles` という名前の Aurora Serverless v1 DB クラスターに対するクエリとその応答を次に示します。

```
$ aws rds describe-db-clusters --region us-west-1 --db-cluster-identifier west-coast-
sles
--query '*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}'

[
    {
        "ScalingConfigurationInfo": {
            "MinCapacity": 1,
            "MaxCapacity": 64,
            "AutoPause": false,
            "SecondsBeforeTimeout": 300,
            "SecondsUntilAutoPause": 300,
            "TimeoutAction": "RollbackCapacityChange"
        }
    }
]
```

応答が示すように、この Aurora Serverless v1 DB クラスターはデフォルト設定を使用しています。

詳細については、「Aurora Serverless v1 DB クラスターの作成」を参照してください。Aurora Serverless v1 を作成した後、タイムアウトアクションやその他の容量設定はいつでも変更できます。この方法については、「Aurora Serverless v1 DB クラスターの変更」を参照してください。

Aurora Serverless v1 の一時停止と再開

Aurora Serverless v1 DB クラスターでは、アクティビティなしの状態が一定時間継続した場合に、一時停止させるような設定が可能です。DB クラスターを一時停止するまでのアクティビティなしの時間を指定します。このオプションを選択すると、デフォルトの非アクティブ時間は 5 分ですが、この値は変更できます。これは任意の設定です。

DB クラスターを一時停止すると、コンピューティングまたはメモリのアクティビティが完全に停止し、ストレージに対してのみ課金されます。Aurora Serverless v1 DB クラスターの一時停止中にデータベース接続がリクエストされると、DB クラスターは自動的に再開して接続リクエストに対応します。

DB クラスターがアクティビティを再開すると、Aurora がクラスターを一時停止したときと同じ容量になります。ACU の数は、クラスターを一時停止する前に Aurora がスケールアップまたはスケールダウンした程度によって異なります。

> **Note**
>
> DB クラスターを一時停止してから 7 日間を超えると、DB クラスターはスナップショットからバックアップされる場合があります。この場合、接続要求がある場合、Aurora は DB クラスターをスナップショットから復元します。

Aurora Serverless v1 のデータベース接続の最大数を確認する

以下は、MySQL 5.7 との互換性がある Aurora Serverless v1 DB クラスターの例です。アクセスを設定している場合は、MySQL クライアントまたはクエリエディタを使用できます。詳細については、「クエリエディタでのクエリの実行」を参照してください。

許可されるデータベース接続の最大数を確認するため、

1. AWS CLI を使用する Aurora Serverless v1 DB クラスターの容量範囲を確認します。

```
aws rds describe-db-clusters \
```

```
    --db-cluster-identifier my-serverless-57-cluster \
    --query 'DBClusters[*].ScalingConfigurationInfo|[0]'
```

その結果、容量範囲が 1 ~ 4 ACU であることがわかりました。

```
{
    "MinCapacity": 1,
    "AutoPause": true,
    "MaxCapacity": 4,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

2. 次の SQL クエリを実行して、接続の最大数を調べます。

```
select @@max_connections;
```

表示される結果は、クラスターの最小容量、つまり 1 ACU についてです。

```
@@max_connections
90
```

3. クラスターを 8 ~ 32 ACU にスケールします。

スケーリングの詳細については、「Aurora Serverless v1 DB クラスターの変更」を参照してください。

4. 容量範囲を確認します。

```
{
    "MinCapacity": 8,
    "AutoPause": true,
    "MaxCapacity": 32,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

5. 接続の最大数を確認します。

```
select @@max_connections;
```

表示される結果は、クラスターの最小容量、つまり 8 ACU についてです。

```
@@max_connections
1000
```

6. クラスターを可能な限り最大、つまり 256 ~ 256 ACU にスケールします。
7. 容量範囲を確認します。

```
{
    "MinCapacity": 256,
    "AutoPause": true,
    "MaxCapacity": 256,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

8. 接続の最大数を確認します。

```
select @@max_connections;
```

表示された結果は 256 ACU についてです。

```
@@max_connections
6000
```

> **Note**
>
> max_connections 値は ACU の数に合わせて直線的にスケールされません。

9. クラスターを 1 ~ 4 ACU にスケールダウンします。

```
{
    "MinCapacity": 1,
    "AutoPause": true,
    "MaxCapacity": 4,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

この場合、max_connections 値は 4 ACU です。

```
@@max_connections
270
```

10. クラスターを 2 つの ACU にスケールダウンさせます。

```
@@max_connections
180
```

一定時間アイドル状態になった後、クラスターを一時停止するように設定した場合は、0 ACU にスケールダウンします。ただし、max_connections が 1 ACU の値を下回ることはありません。

```
@@max_connections
90
```

Aurora Serverless v1 のパラメータグループ

Aurora Serverless v1 DB クラスターの作成時に、特定の Aurora DB エンジンと、それに関連する DB クラスターパラメータグループを選択します。プロビジョニングされた Aurora DB クラスターとは異なり、Aurora Serverless v1 DB クラスターには単一の読み取り/書き込み DB インスタンスがあり、DB クラスターパラメータグループで構成されているだけで、— 個別の DB パラメータグループはありません。オートスケーリング中、Aurora Serverless v1 は、容量の増減に対してクラスターが最適に機能するように、パラメータを変更できる必要があります。したがって、Aurora Serverless v1 DBクラスターでは、特定のDBエンジンタイプのパラメータに加える可能性のある変更の一部が適用されないことがあります。

例えば、Aurora PostgreSQL-ベースAurora Serverless v1の DB クラスターは、クエリプラン管理のためにプロビジョンド Aurora PostgreSQL DB クラスターで使用される可能性のある apg_plan_mgmt.capture_plan_baselines とその他のパラメータを使用できません。

CLI コマンドの describe-engine-default-cluster-parameters を使用し AWS リージョン に対しクエリすることで、さまざまな Aurora DB エンジンのデフォルトパラメータグループのデフォルト値のリストを取得できます。--db-parameter-group-family オプションに使用できる値は次のとおりです。

Aurora MySQL バージョン 2	aurora-mysql5.7

Aurora PostgreSQL バージョン 11	`aurora-postgresql11`
Aurora PostgreSQL バージョン 13	`aurora-postgresql13`

AWS アクセスキー ID と AWS シークレットアクセスキーを使用して AWS CLI を設定し、AWS CLI コマンドを使用する前に AWS リージョン を設定することをお勧めします。CLI 設定にリージョンを指定すると、コマンドの実行時に `--region` パラメータを入力する手間が省けます。AWS CLI の設定の詳細については、AWS Command Line Interface ユーザーガイドの「設定の基本」を参照してください。

次の例では、Aurora MySQL バージョン 2 のデフォルトの DB クラスターグループからパラメータのリストを取得します。

Linux、macOS、Unix の場合:

```
aws rds describe-engine-default-cluster-parameters \
  --db-parameter-group-family aurora-mysql5.7 --query \
  'EngineDefaults.Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} | [?
contains(SupportedEngineModes, `serverless`) == `true`] | [*].{param:ParameterName}' \
  --output text
```

Windows の場合:

```
aws rds describe-engine-default-cluster-parameters ^
   --db-parameter-group-family aurora-mysql5.7 --query ^
   "EngineDefaults.Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} | [?
contains(SupportedEngineModes, 'serverless') == `true`] | [*].{param:ParameterName}" ^
   --output text
```

Aurora Serverless v1 のパラメータ値の変更

「パラメータグループを使用する」 で説明したように、デフォルトのパラメータグループの値は、そのタイプ (DBクラスターパラメータグループ、DBパラメータグループ) に関係なく、直接変更することはできません。代わりに、Aurora DB エンジンのデフォルトの DB クラスターパラメータグループに基づいてカスタムパラメータグループを作成し、そのパラメータグループの設定を必要に応じて変更します。例えば、Aurora Serverless v1 DB クラスターの設定の一部を変更して、クエリをログに記録したり、DB エンジン固有のログを Amazon CloudWatch にアップロードしたりできます。

カスタム DB クラスターパラメータグループを作成するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. [パラメータグループ] を選択します。
3. [パラメータグループの作成] をクリックして、パラメータグループの詳細ペインを開きます。
4. Aurora Serverless v1 DB クラスターに使用する DB エンジンに適切なデフォルト DB クラスターグループを選択します。必ず次のオプションを選択してください。

    a. [パラメータグループファミリー] で、選択した DB エンジンに適したファミリを選択します。選択内容の名前には、`aurora-` プレフィックスが含まれていることを確認してください。

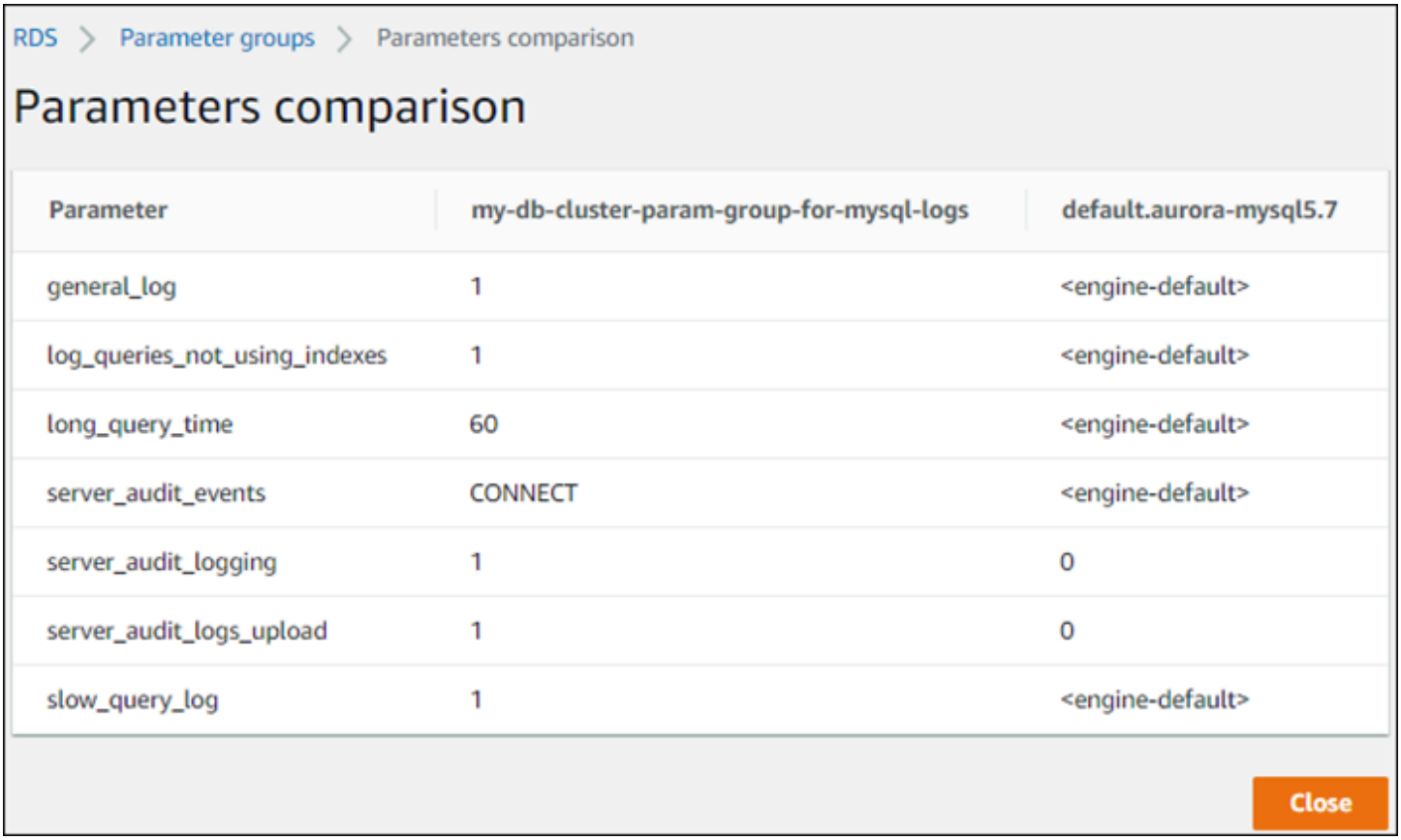
    b. [Type] で、[DB Cluster Parameter Group] を選択します。

    c. [グループ名] と [説明] に、ご自身や、Aurora Serverless v1 DB クラスターとそのパラメータを操作する必要がある他のユーザーにとって、わかりやすい名前を入力します。

    d. [作成] を選択します。

カスタム DB クラスターパラメータグループが、お使いの AWS リージョン で使用できる、パラメータグループのリストに追加されます。これで、新しい Aurora Serverless v1 DB クラスターを作成する際に、このカスタムDB クラスターパラメータグループを使用できるようになります。カスタム DB クラスターパラメータグループが使用されように、既存の Aurora Serverless v1 DB クラスターを変更することもできます。ご使用の Aurora Serverless v1 DB クラスターがカスタム DB クラスターパラメータグループの使用をスタートしたら、AWS Management Console または AWS CLI を使用して動的パラメータの値を変更できます。

コンソールを使用して、次のスクリーンショットに示すように、カスタム DB クラスターパラメータグループの値をデフォルトの DB クラスターパラメータグループと比較した値を並べて表示することもできます。



アクティブな DB クラスターのパラメータ値を変更すると、Aurora Serverless v1 は、パラメータの変更を適用するために、シームレスなスケーリングをスタートします。Aurora Serverless v1 DB クラスターは「一時停止」状態にあり、変更を行うことができるように再開してスケーリングをスタートします。パラメータグループ変更のスケーリング操作は常に容量の変更を強制するため、スケーリング期間中にスケーリングポイントが見つからない場合、パラメータを変更すると接続が切断される可能性があることに注意してください。

Aurora Serverless v1 のログ記録

デフォルトでは、Aurora Serverless v1 のエラーログが有効になっており、Amazon CloudWatch に自動的にアップロードされます。また、Aurora Serverless v1 DB クラスターは、Aurora データベースエンジン固有のログを CloudWatch にアップロードします。これを行うには、カスタム DB クラスターパラメータグループで設定パラメータを有効にします。Aurora Serverless v1 DB クラスターは、使用可能なすべてのログを Amazon CloudWatch にアップロードします。CloudWatch を使用すると、ログデータの分析や、CloudWatch を使用したアラームの作成、メトリクスの表示を行うことができます。

Aurora MySQL について、以下の表は有効にできるログを示しています。有効にすると、Aurora Serverless v1 DB クラスターから Amazon CloudWatch に自動的にアップロードされます。

Aurora MySQL ログ	説明
general_log	一般ログを作成します。オンにするには、1 に設定します。デフォルトはオフ (0) です。
log_queries_not_using_indexes	インデックスを使用しないスロークエリログにクエリを記録します。デフォルトはオフ (0) です。このログを有効にするには、1 に設定します。
long_query_time	高速実行クエリがスロークエリログに記録されないようにします。0 から 3,1536,000 までの浮動小数点数に設定できます。デフォルトは 0 (アクティブではない) です。
server_audit_events	ログにキャプチャするイベントのリスト。サポートされている値は CONNECT、QUERY、QUERY_DCL 、QUERY_DDL 、QUERY_DML 、TABLE です。
server_audit_logging	この値を 1 に設定すると、サーバー監査ログ記録がオンになります。これをオンにしている場合は、CloudWatch に送信する監査イベントを、server_audit_events パラメータにリストアップすることで指定できます。
slow_query_log	スロークエリログを作成します。スロークエリログをオンにするには、1 に設定します。デフォルトはオフ (0) です。

詳細については、「Amazon Aurora MySQL DB クラスターでのアドバンストな監査の使用」を参照してください。

Aurora PostgreSQL について、以下の表は有効にできるログを示しています。有効にすると、通常のエラーログと一緒に Aurora Serverless v1 DB クラスターから Amazon CloudWatch に自動的にアップロードされます。

Aurora PostgreSQL ログ	説明
log_connections	デフォルトで有効になっており、変更できません。これは、すべての新しいクライアント接続の詳細をログに記録します。
log_disconnections	デフォルトで有効になっており、変更できません。クライアントの切断をすべてログに記録します。
log_hostname	デフォルトでオフになっており、変更できません。ホスト名はログに記録されません。
log_lock_waits	デフォルトは 0 (オフ) です。ロック待機をログに記録するには、1 に設定します。
log_min_duration_statement	ステートメントがログに記録されるまでに実行される最小時間 (ミリ秒単位)。
log_min_messages	ログに記録するメッセージレベルを設定します。サポートされている値は debug5、debug4、debug3、debug2、debug1、inf です。 パフォーマンスデータを postgres ログに記録するには、値を debug1 に設定します。
log_temp_files	指定されたキロバイト (KB) を超えるテンポラリファイルの使用を記録します。
log_statement	ログに記録される特定の SQL ステートメントを制御します。サポートされている値は none、ddl、mod、all です。デフォルトは none です。

Aurora Serverless v1 DB クラスターの Aurora MySQL または Aurora PostgreSQL のログをオンにすると、CloudWatch でログを表示できます。

Amazon CloudWatch を使用して Aurora Serverless v1 ログを表示する

Aurora Serverless v1 は DB クラスターのカスタムパラメータグループで有効になっているすべてのログを Amazon CloudWatch に自動的にアップロード (公開) します。ログの種類を選択または指定する必要はありません。ログのアップロードは、ログ設定パラメータを有効にするとすぐにスタートされます。後でログパラメータを無効にすると、それ以降のアップロードが停止します。ただし、既に CloudWatch に公開されているすべてのログは、削除するまで残ります。

CloudWatch と Aurora MySQL ログの使用の詳細については、「Amazon CloudWatch でログイベントをモニタリングする」を参照してください。

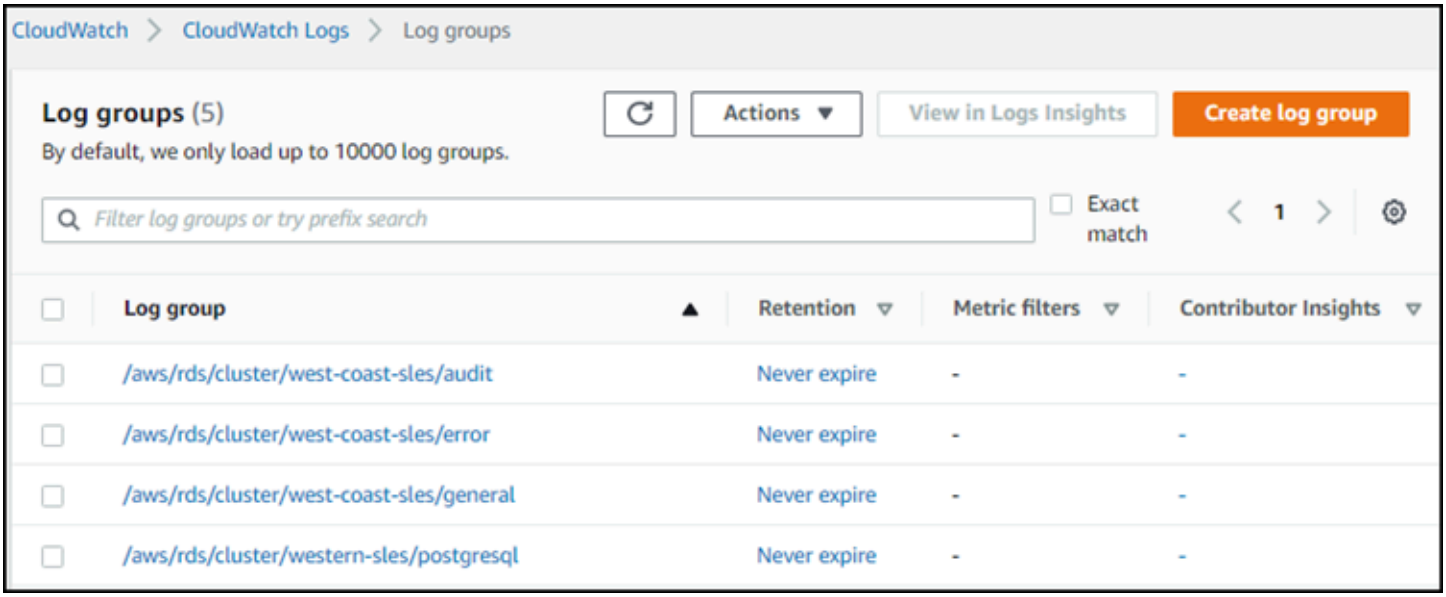
CloudWatch と Aurora PostgreSQL の詳細については、「Amazon CloudWatch Logs への Aurora PostgreSQL ログの発行」を参照してください。

Aurora Serverless v1 DB クラスターのログを表示するには

1. CloudWatch コンソール (https://console.aws.amazon.com/cloudwatch/) を開きます。
2. AWS リージョン を選択します。
3. [ロググループ] を選択します。
4. リストから Aurora Serverless v1 DB クラスターのログを選択します。エラーログの命名パターンは、次に従います。

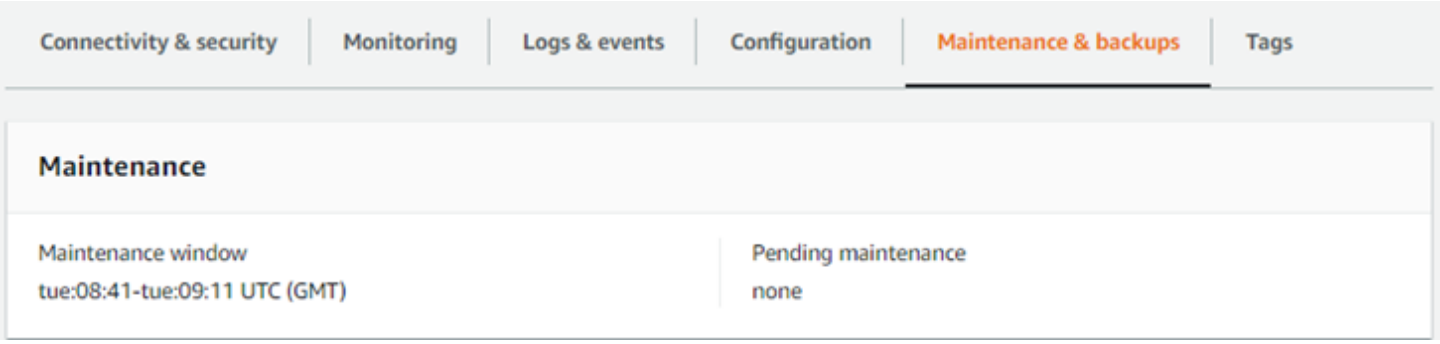
```
/aws/rds/cluster/cluster-name/error
```

例えば、次のスクリーンショットでは、`western-sles` という名前の Aurora PostgreSQL Aurora Serverless v1 DB クラスターに対して公開されたログのリストを見つけることができます。また、Aurora MySQL Aurora Serverless v1 DB クラスター、`west-coast-sles` のリストもいくつか見つけることができます。目的のログを選択して、そのコンテンツの探索をスタートします。



Aurora Serverless v1 とメンテナンス

最新の機能、修正、セキュリティ更新の適用など、Aurora Serverless v1 DB クラスターのメンテナンスは自動的に実行されます。Aurora Serverless v1 にはメンテナンス期間があり、Aurora Serverless v1 DB クラスターの [メンテナンスとバックアップ] の AWS Management Console で確認できます。次の図に示すように、メンテナンスが実行される予定日時と、Aurora Serverless v1 DB クラスターで保留されているメンテナンスがあるかどうかが確認できます。



メンテナンス期間は、Aurora Serverless v1 DB クラスターの作成時に設定でき、後で期間を変更できます。詳細については、「DB クラスターの適切なメンテナンスウィンドウの調整」を参照してください。

メンテナンス期間は、予定されたメジャーバージョンアップグレードに使用されます。マイナーバージョンのアップグレードとパッチは、スケーリング中に直ちに適用されます。スケーリングは、`TimeoutAction` の設定に従って行われます。

- `ForceApplyCapacityChange` - 変更は直ちに適用されます。

- `RollbackCapacityChange` - Aurora は最初のパッチ試行から 3 日後にクラスターを強制的に更新します。

適切なスケーリングポイント以外で強制される変更と同様に、この変更でもワークロードが中断されることがあります。

Aurora Serverless v1 は、可能な限り、中断せずに、メンテナンスを実行します。メンテナンスが実施される場合、Aurora Serverless v1 DB クラスターは必要な操作を処理するために容量をスケールアップします。スケーリングする前に、Aurora Serverless v1 はスケーリングポイントを探します。必要に応じて最大 3 日間行います。

Aurora Serverless v1 によりスケーリングポイントが見つけられない場合は、その 1 日の終わりにクラスターイベントが作成されます。このイベントは、保留中のメンテナンスについてと、メンテナンスを実行するためのスケーリングの必要性についてを通知します。通知には、Aurora Serverless v1 が DB クラスターを強制的にスケーリングを実行できる日付が含まれています。

詳細については、「容量の変更のタイムアウトアクション」を参照してください。

Aurora Serverless v1 とフェイルオーバー

Aurora Serverless v1 DB クラスターの DB インスタンスが使用不能になるか、その クラスターがあるアベイラビリティーゾーン (AZ) で障害が発生した場合、Aurora は別の AZ に DB インスタンスを再作成します。ただし、Aurora Serverless v1 クラスターはマルチ AZ クラスターではありません。これは、単一の AZ 内の単一の DB インスタンスで構成されているためです。このフェイルオーバーメカニズムは、プロビジョニングされたインスタンスや Aurora Serverless v2 インスタンスを持つ Aurora クラスターよりも時間がかかります。Aurora Serverless v1 のフェイルオーバー時間は、対象の AWS リージョン 内での、他の AZ の需要や利用可能な容量によって変動するため、定義されていません。

Aurora では、コンピューティング容量とストレージは分離されるため、クラスターのストレージボリュームは複数の AZ に分散されます。停止が DB インスタンスまたは関連する AZ に影響する場合でも、データは引き続き利用することができます。

Aurora Serverless v1 とスナップショット

Aurora Serverless v1 クラスターでは、クラスターボリュームは常に暗号化されます。暗号化キーは選択できますが、暗号化を無効にすることはできません。Aurora Serverless v1 クラスターのスナップショットをコピーまたは共有するには、独自の AWS KMS key を使用してスナップショットを暗号化します。詳細については、「DB クラスターのスナップショットのコピー」を参照してくださ

い。暗号化と Amazon Aurora の詳細については、「Amazon Aurora DB クラスターの暗号化」を参照してください。

Aurora Serverless v1 DB クラスターの作成

次の手順では、スキーマオブジェクトやデータを使用せずに Aurora Serverless v1 クラスターを作成します。既存のプロビジョニングされたクラスターまたは Aurora Serverless v1 クラスターの複製である Aurora Serverless v1 クラスターを作成する場合、代わりにスナップショットの復元またはクローンオペレーションを実行できます。これらの詳細については、「DB クラスターのスナップショットからの復元」および「Amazon Aurora DB クラスターのボリュームのクローン作成」を参照してください。既存のプロビジョニングされたクラスターを Aurora Serverless v1 に変換することはできません。また、既存の Aurora Serverless v1 クラスターをプロビジョニングされたクラスターに戻すことはできません。

Aurora Serverless v1 DB クラスターを作成する際に、そのクラスターの容量に、最小と最大の値をそれぞれ設定できます。容量ユニットは、コンピューティングとメモリに関する特定の構成と等価です。Aurora Serverless v1 は、CPU 使用率、接続、および使用可能なメモリの各しきい値に関するスケーリングルールを作成し、アプリケーションに必要な容量単位でシームレスにスケーリングすることができます。詳細については、「Aurora Serverless v1 アーキテクチャ」を参照してください。

Aurora Serverless v1 DB クラスターには、次の特定の値を設定できます。

- [最小 Aurora 容量ユニット] - Aurora Serverless v1 が容量ユニットを減らすことができる最小数です。
- [最大 Aurora 容量ユニット] - Aurora Serverless v1 が容量ユニットを増やすことができる最大数です。

また、次のオプションのスケーリング設定オプションを選択することもできます。

- タイムアウトの到達時にキャパシティーを指定した値に強制的にスケールする – タイムアウトする前にスケーリングポイントが見つからない場合でも、Aurora Serverless v1 によるスケーリングを Aurora Serverless v1 に強制させる場合は、この設定を選択します。スケーリングポイントが見つからない場合に、キャパシティーの変更を Aurora Serverless v1 にキャンセルさせたい場合は、この設定を選択しないでください。詳細については、「容量の変更のタイムアウトアクション」を参照してください。
- 連続する数分間非アクティブになった場合にコンピューティング性能を一時停止する - 指定した時間 DB クラスターにアクティビティがない場合に Aurora Serverless v1 が 0 にスケールするように

する場合は、この設定を選択します。この設定を有効にすると、Aurora Serverless v1 DB クラスターは自動的に処理を再開し、データベーストラフィックの再開時にワークロードを処理するために必要な容量にスケールします。詳細については、「Aurora Serverless v1 の一時停止と再開」を参照してください。

Aurora Serverless v1 DB クラスターを作成する前に、AWS アカウントが必要です。また、Amazon Aurora で作業するためのセットアップタスクも完了しておく必要があります。詳細については、「Amazon Aurora の環境をセットアップする」を参照してください。Aurora DB クラスターを作成するための他の準備ステップも完了する必要があります。詳細については、「Amazon Aurora DB クラスターの作成」を参照してください。

Aurora Serverless v1 を使用できるのは、一部の AWS リージョン と、特定の Aurora MySQL および Aurora PostgreSQL のバージョンのみです。詳細については、「Aurora Serverless v1 でサポートされているリージョンと Aurora DB エンジン」を参照してください。

> **Note**
>
> Aurora Serverless v1 クラスターでは、クラスターボリュームは常に暗号化されます。Aurora Serverless v1 DB クラスターを作成する際に、暗号化をオフにすることはできませんが、独自の暗号化キーを使用することは可能です。Aurora Serverless v2 では、クラスターボリュームを暗号化するかどうかを選択できます。

Aurora Serverless v1 DB クラスターは、AWS Management Console、AWS CLI、または RDS API を使用して作成できます。

> **Note**
>
> クラスターの作成時に次のエラーメッセージが表示される場合は、アカウントに追加のアクセス許可が必要です。
> `Unable to create the resource. Verify that you have permission to create service linked role. Otherwise wait and try again later.`
> 詳細については、「Amazon Aurora のサービスにリンクされたロールの使用」を参照してください。

Aurora Serverless v1 DB クラスターの DB インスタンスに直接接続することはできません。Aurora Serverless v1 DB クラスターに接続するには、データベースエンドポイントを使用します。Aurora

Serverless v1 DB クラスターのエンドポイントは、 AWS Management Console で、クラスターの [接続とセキュリティ] タブから確認できます。詳細については、「Amazon Aurora DB クラスターへの接続」を参照してください。

コンソール

次の一般的な手順を使用します。AWS Management Console による Aurora DB クラスターの作成の詳細については、「Amazon Aurora DB クラスターの作成」を参照してください。

新しい Aurora Serverless v1 DB クラスターを作成するには

1. AWS Management Consoleにサインインします。
2. Aurora Serverless v1 をサポートする AWS リージョン を選択します。
3. AWS サービスリストから [Amazon RDS] を選択します。
4. [データベースの作成] を選択します。
5. [データベースの作成] ページで、次の操作を行います。
   a. データベースの作成方法として [スタンダード作成] を選択します。
   b. 次の例に示すステップを使用して、Aurora Serverless v1 DB クラスターの作成に進みます。

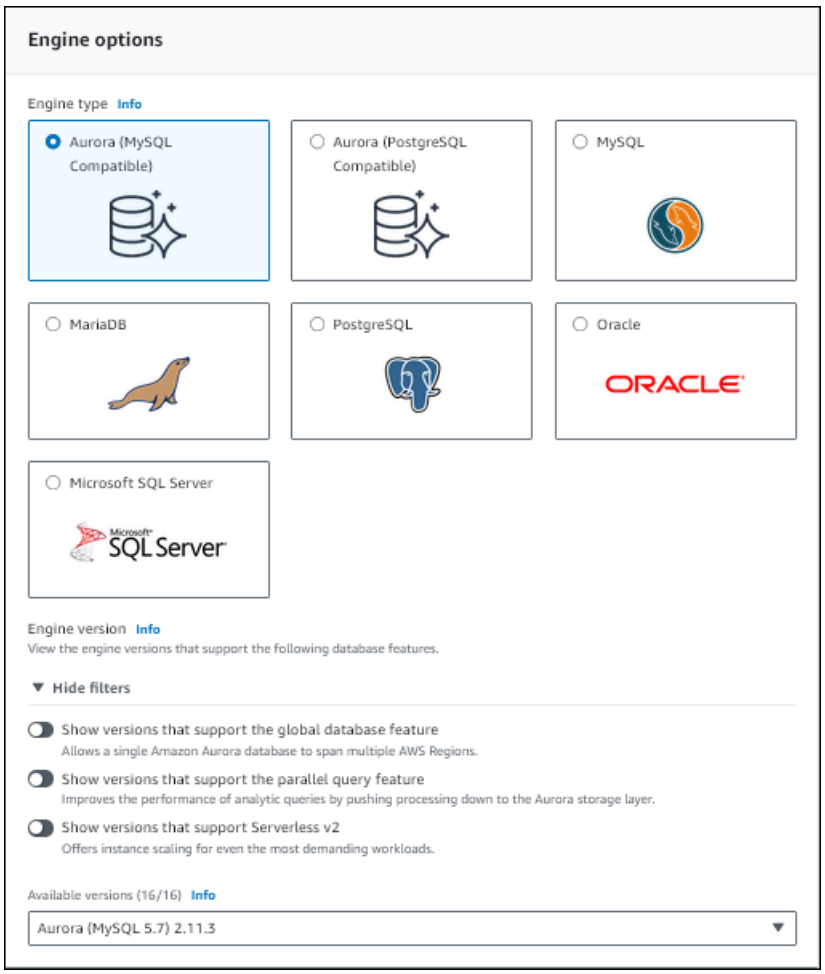
> **Note**
>
> Aurora Serverless v1 をサポートしていない DB エンジンのバージョンを選択した場合、[サーバーレス] オプションは [DB インスタンスクラス] に表示されません。

Aurora MySQL の例

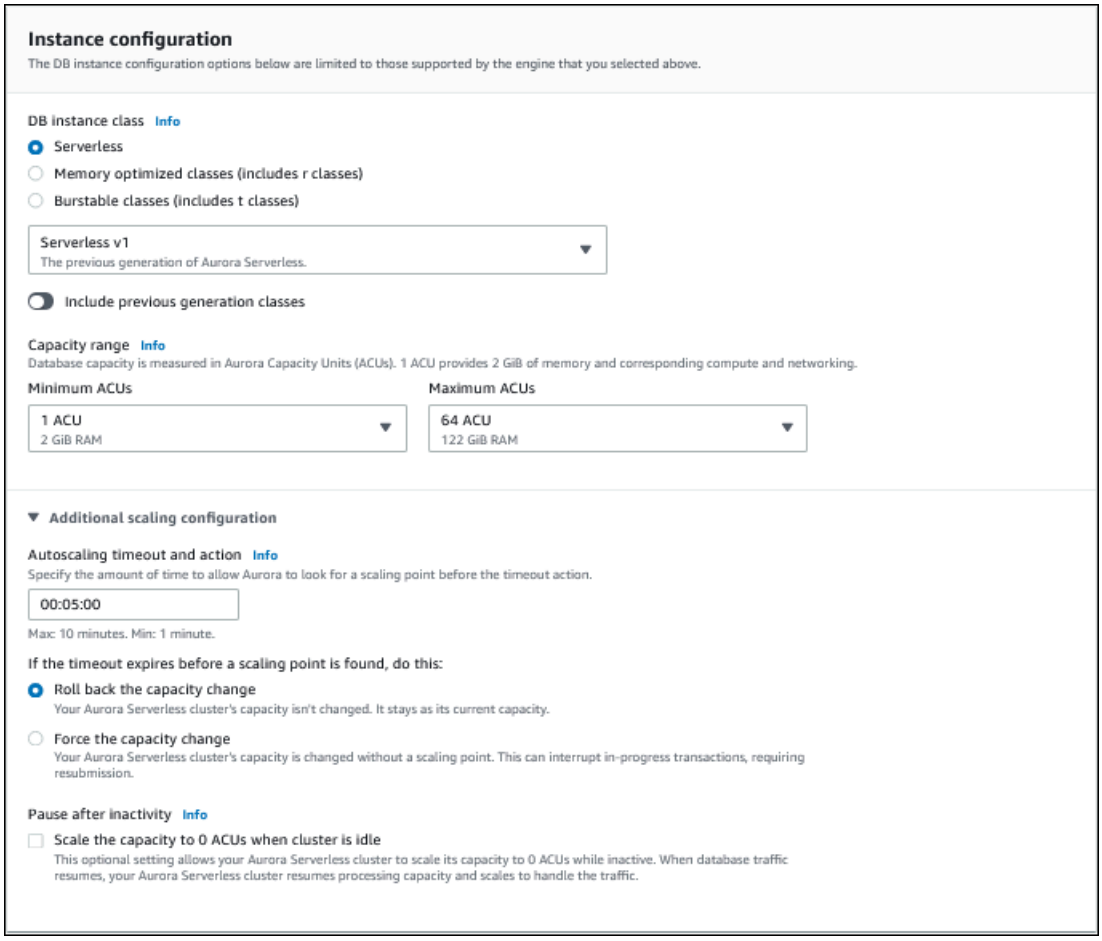
次の手順に従ってください。

Aurora MySQL 用の Aurora Serverless v1 DB クラスターを作成するには

1. [エンジンタイプ] として、[Aurora (MySQL 互換)] を選択します。
2. DB クラスターに使用する、Aurora Serverless v1 と互換性のある Aurora MySQL バージョンを選択します。サポートされているバージョンはページの右側にあります。



3. [DB インスタンスクラス] で、[サーバーレス] を選択します。
4. DB クラスターの[Capacity range] (容量範囲) を設定します。
5. ページの [Additional scaling configuration] (追加のスケーリング設定) セクションで、必要に応じて値を調整します。容量設定の詳細については、「Aurora Serverless v1 のオートスケーリング」を参照してください。



6. Aurora Serverless v1 DB クラスターの Data API を有効にするには、[Connectivity] (接続) セクションの [Additional configuration] (追加設定) の下にある [Data API] チェックボックスをオンにします。

   Data API の詳細については、「RDS Data API の使用」を参照してください。

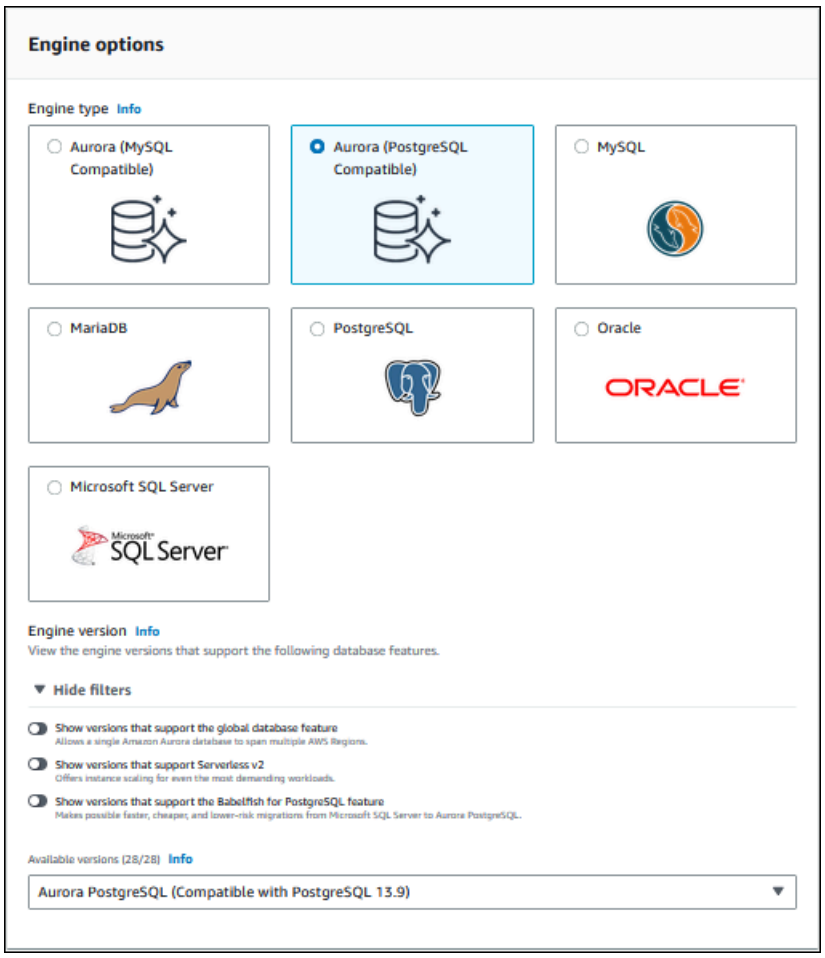
7. 必要に応じて他のデータベース設定を選択してから、[Create database] (データベースの作成) を選択します。

Aurora PostgreSQL の例

次の手順に従ってください。

Aurora PostgreSQL 用の Aurora Serverless v1 DB クラスターを作成するには

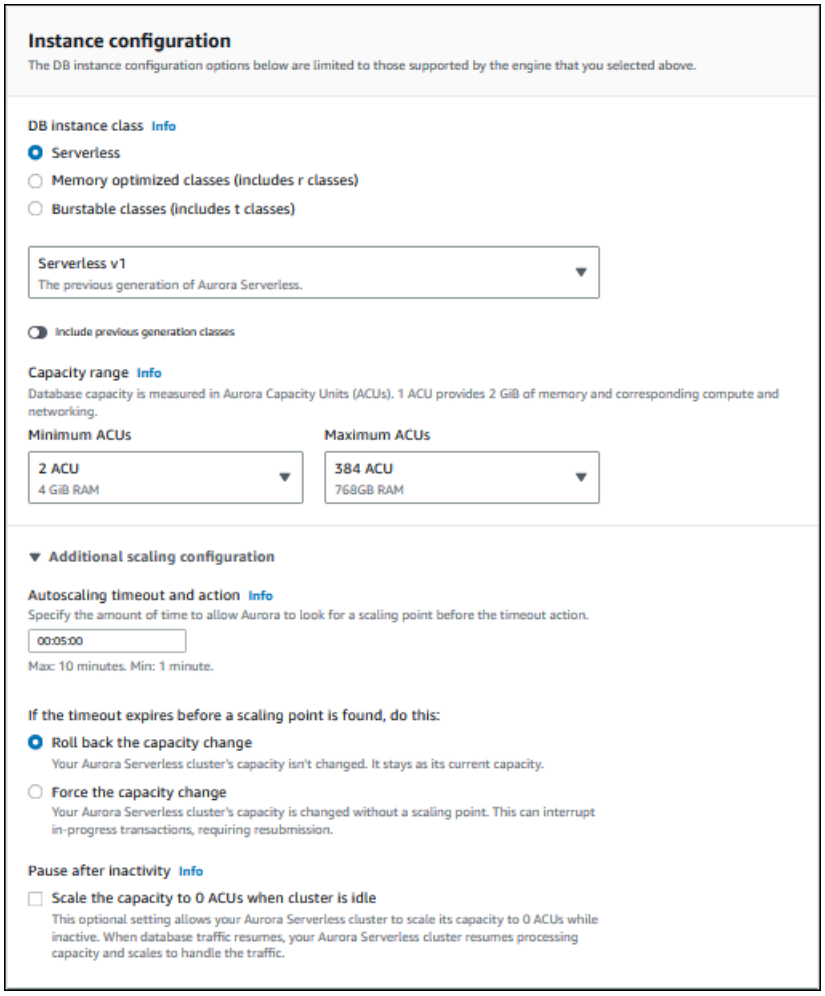
1. [エンジンタイプ] として、[Aurora (PostgreSQL 互換)] を選択します。
2. DB クラスターに使用する、Aurora Serverless v1 と互換性のある Aurora PostgreSQL バージョンを選択します。サポートされているバージョンはページの右側にあります。



3. [DB インスタンスクラス] で、[サーバーレス] を選択します。
4. Aurora PostgreSQL バージョン 13 のマイナーバージョンを選択した場合は、メニューから [Serverless v1] を選択します。

> **Note**
>
> また、Aurora PostgreSQL バージョン 13 は、Aurora Serverless v2 をサポートしています。

5. DB クラスターの[Capacity range] (容量範囲) を設定します。
6. ページの [Additional scaling configuration] (追加のスケーリング設定) セクションで、必要に応じて値を調整します。容量設定の詳細については、「Aurora Serverless v1 のオートスケーリング」を参照してください。



7. Aurora Serverless v1 DB クラスターで Data API を使用にするには、[接続] セクションの [追加設定] の下にある [Data API] チェックボックスをオンにします。

   Data API の詳細については、「RDS Data API の使用」を参照してください。

8. 必要に応じて他のデータベース設定を選択してから、[Create database] (データベースの作成) を選択します。

AWS CLI

Aurora Serverless v1 で新しい AWS CLI DB クラスターを作成するには、`serverless` オプションに `--engine-mode` を指定しながら、create-db-cluster コマンドを実行します。

必要に応じて `--scaling-configuration` オプションを指定し、最小容量、最大容量、および接続がないときの自動的な一時停止を設定できます。

次のコマンドの例では、--engine-mode オプションを serverless に設定して、新しいサーバーレス DB クラスターを作成します。この例では、--scaling-configuration オプションの値も指定します。

Aurora MySQL の例

以下のコマンドでは、Aurora MySQL 互換のサーバーレス DB クラスターを作成します。Aurora MySQL の有効な容量値は、1、2、4、8、16、32、64、128、および 256 です。

Linux、macOS、Unix の場合:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster \
    --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.4 \
    --engine-mode serverless \
    --scaling-configuration
 MinCapacity=4,MaxCapacity=32,SecondsUntilAutoPause=1000,AutoPause=true \
    --master-username username --master-user-password password
```

Windows の場合:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
    --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.4 ^
    --engine-mode serverless ^
    --scaling-configuration
 MinCapacity=4,MaxCapacity=32,SecondsUntilAutoPause=1000,AutoPause=true ^
    --master-username username --master-user-password password
```

Aurora PostgreSQL の例

次のコマンドでは、PostgreSQL 13.9 互換の、新しいサーバーレス DB クラスターを作成します。Aurora PostgreSQL の有効な容量値は、2、4、8、16、32、64、192、および 384 です。

Linux、macOS、Unix の場合:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster \
    --engine aurora-postgresql --engine-version 13.9 \
    --engine-mode serverless \
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=1000,AutoPause=true \
    --master-username username --master-user-password password
```

Windows の場合:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
    --engine aurora-postgresql --engine-version 13.9 ^
    --engine-mode serverless ^
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=1000,AutoPause=true ^
    --master-username username --master-user-password password
```

RDS API

RDS API で新しい Aurora Serverless v1 DB クラスターを作成するには、`serverless` パラメータに `EngineMode` を指定しながら、CreateDBCluster オペレーションを実行します。

必要に応じて `ScalingConfiguration` パラメータを指定し、最小容量、最大容量、および接続がないときの自動的な一時停止を設定できます。有効な容量値には次のようなものがあります。

- Aurora MySQL: 1、2、4、8、16、32、64、128、および 256 です。
- Aurora PostgreSQL: 2、4、8、16、32、64、192、および 384 です。

Aurora Serverless v1 DB クラスターの復元

プロビジョニングされた DB クラスターのスナップショットを、Aurora Serverless v1、AWS Management Console、または RDS API を使用しながら復元する際、AWS CLI DB クラスターを設定することができます。

スナップショットを Aurora Serverless v1 DB クラスターに復元する場合は、次の特定の値を設定することができます。

- [最小 Aurora 容量ユニット] - Aurora Serverless v1 が容量ユニットを減らすことができる最小数です。
- [最大 Aurora 容量ユニット] - Aurora Serverless v1 が容量ユニットを増やすことができる最大数です。
- タイムアウトアクション - スケーリングポイントが見つからないために、容量の変更がタイムアウトした場合に実行するアクション。Aurora Serverless v1[指定した値への容量のスケーリングを強制する...] オプションを設定した場合、DB クラスターはユーザーの DB クラスターに対し、新しい容量設定を強制できます。または、オプションを選択しなかった場合、容量の変更をロールバックしてキャンセルすることもできます。詳細については、「容量の変更のタイムアウトアクション」を参照してください。

- [Pause after inactivity (アイドル状態後の一時停止)] - データベーストラフィックがなくなってから処理容量をゼロにスケールするまでの時間を指定します。データベーストラフィックが再開されると、Aurora は自動的に処理容量を再開しトラフィックを処理できるようにスケールします。

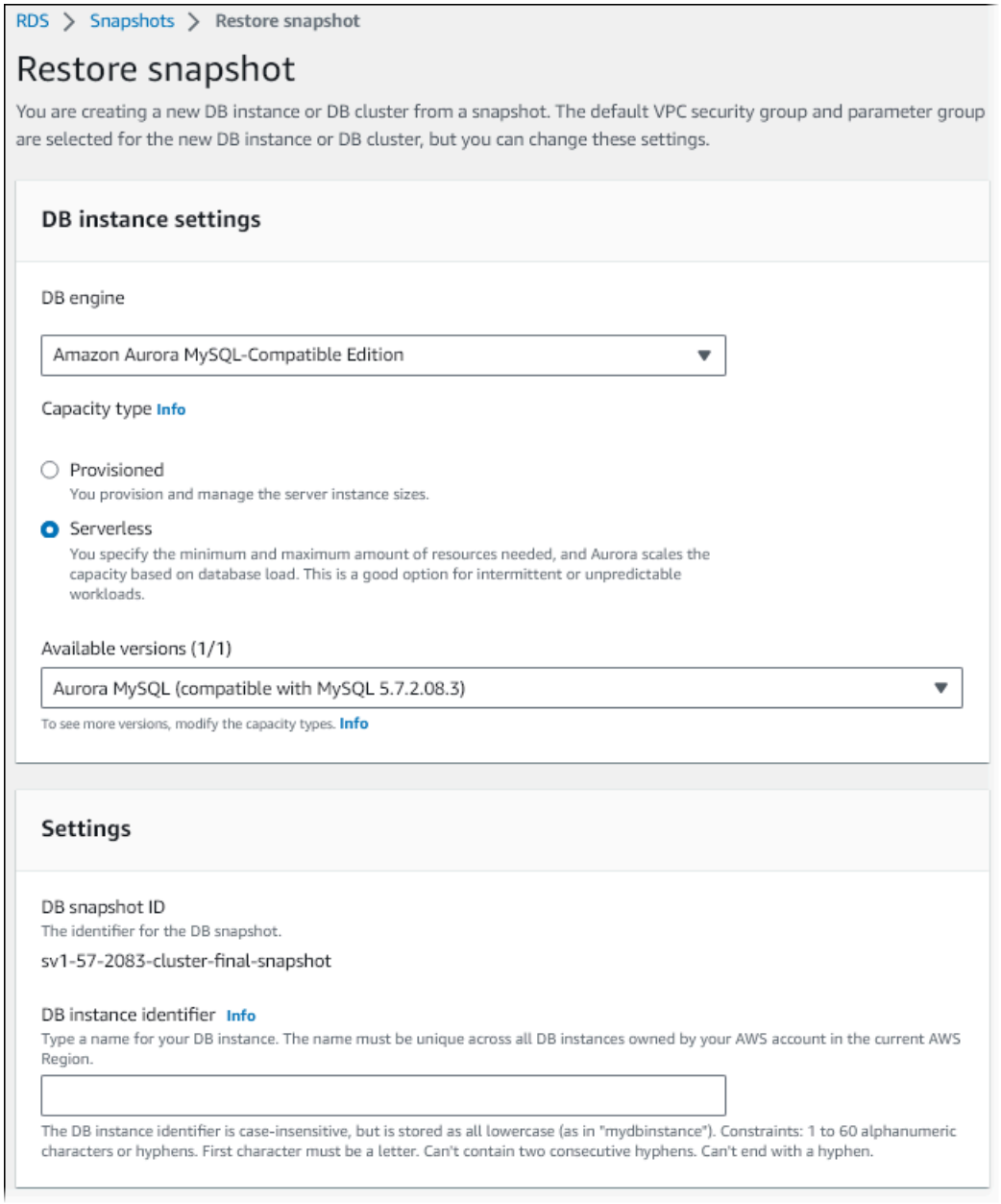
スナップショットから DB クラスターを復元するための全般情報については、「DB クラスターのスナップショットからの復元」を参照してください。

コンソール

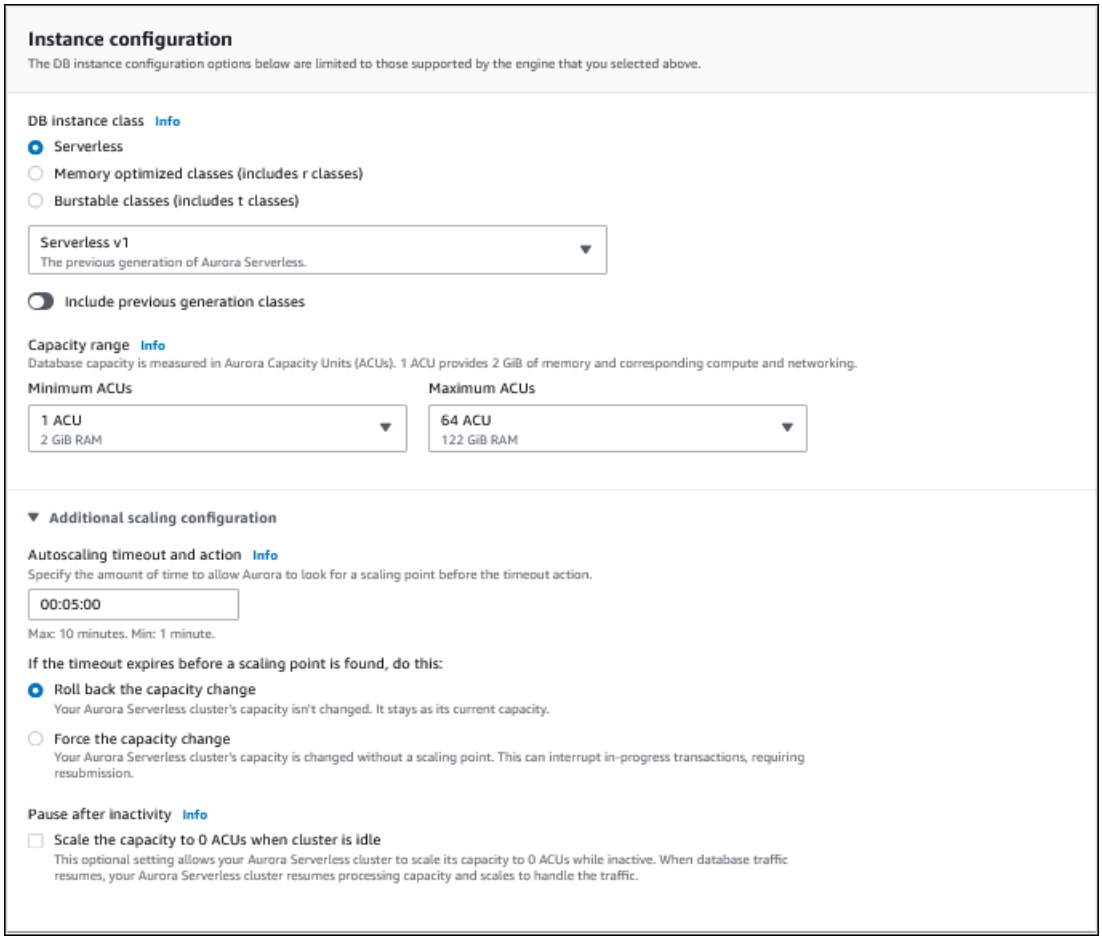
AWS Management Console を使用して、DB クラスターのスナップショットを Aurora DB クラスターに復元できます。

DB クラスターのスナップショットを Aurora DB クラスターに復元するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. AWS Management Console の右上で、ソース DB クラスターをホストする AWS リージョン を選択します。
3. ナビゲーションペインで、[Snapshots (スナップショット)] を選択し、復元する DB クラスターのスナップショットを選択します。
4. [アクション]、[スナップショットの復元] の順に選択します。
5. [Restore DB Cluster (DB クラスターの復元)] ページで、[Capacity type (容量タイプ)] として [Serverless (サーバーレス)] を選択します。



6. [DB cluster identifier (DB クラスターの識別子)] フィールドに、復元した DB クラスターの名前を入力し、他のフィールドに必要事項を入力します。
7. [Capacity settings (容量設定)] セクションで、スケーリング設定を変更します。



8. [Restore DB Cluster (DB クラスターの復元)] を選択します。

データベースエンドポイントを使用して、Aurora Serverless v1 DB クラスターに接続します。詳細については、「Amazon Aurora DB クラスターへの接続」の手順を参照してください。

> **Note**
>
> 以下のエラーメッセージが返された場合は、アカウントに追加のアクセス許可が必要です。
> `Unable to create the resource. Verify that you have permission to create service linked role. Otherwise wait and try again later.`
> 詳細については、「Amazon Aurora のサービスにリンクされたロールの使用」を参照してください。

AWS CLI

プロビジョニングされた DB クラスターのスナップショットを、Aurora Serverless、AWS Management Console、または RDS API を使用しながら復元する際、AWS CLI DB クラスターを設定することができます。

スナップショットを Aurora Serverless DB クラスターに復元する場合は、次の特定の値を設定することができます。

- [最小 Aurora 容量ユニット] - Aurora Serverless が容量ユニットを減らすことができる最小数です。
- [最大 Aurora 容量ユニット] - Aurora Serverless が容量ユニットを増やすことができる最大数です。
- タイムアウトアクション - スケーリングポイントが見つからないために、容量の変更がタイムアウトした場合に実行するアクション。Aurora Serverless v1[指定した値への容量のスケーリングを強制する...] オプションを設定した場合、DB クラスターはユーザーの DB クラスターに対し、新しい容量設定を強制できます。または、オプションを選択しなかった場合、容量の変更をロールバックしてキャンセルすることもできます。詳細については、「容量の変更のタイムアウトアクション」を参照してください。
- [Pause after inactivity (アイドル状態後の一時停止)] - データベーストラフィックがなくなってから処理容量をゼロにスケールするまでの時間を指定します。データベーストラフィックが再開されると、Aurora は自動的に処理容量を再開しトラフィックを処理できるようにスケールします。

> **Note**
>
> DB クラスタースナップショットのバージョンは、Aurora Serverless v1 と互換性がある必要があります。サポートされているバージョンのリストについては、「Aurora Serverless v1 でサポートされているリージョンと Aurora DB エンジン」を参照してください。

MySQL 5.7 互換の Aurora Serverless v1 クラスターにスナップショットを復元するには、次の追加パラメータも指定します。

- `--engine` **`aurora-mysql`**
- `--engine-version` **`5.7`**

--engine および --engine-version パラメータを使用すると、MySQL 5.6 互換の Aurora または Aurora Serverless v1 のスナップショットから、MySQL 5.7 互換の Aurora Serverless v1 クラスターを作成できます。次の例では、*mydbclustersnapshot* という名前の MySQL 5.6 互換クラスターのスナップショットから、*mynewdbcluster* という名前の MySQL 5.7 互換 Aurora Serverless v1 クラスターを復元しています。

Linux、macOS、Unix の場合:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mynewdbcluster \
    --snapshot-identifier mydbclustersnapshot \
    --engine-mode serverless \
    --engine aurora-mysql \
    --engine-version 5.7
```

Windows の場合:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-instance-identifier mynewdbcluster ^
    --db-snapshot-identifier mydbclustersnapshot ^
    --engine aurora-mysql ^
    --engine-version 5.7
```

必要に応じて --scaling-configuration オプションを指定し、最小容量、最大容量、および接続がないときの自動的な一時停止を設定できます。有効な容量値には次のようなものがあります。

- Aurora MySQL: 1、2、4、8、16、32、64、128、および 256 です。
- Aurora PostgreSQL: 2、4、8、16、32、64、192、および 384 です。

次の例では、以前に作成した *mydbclustersnapshot* という名前の DB クラススナップショットから *mynewdbcluster* という名前の新しい DB クラスターに復元します。新しい --scaling-configuration DB クラスターが必要に応じて 8 ACU から 64 ACU (Aurora 容量ユニット) にスケールしてワークロードを処理できるように Aurora Serverless v1 を設定します。処理が完了し、サポート対象の接続なしで 1,000 秒が経過すると、クラスターはシャットダウンします (接続リクエストがあると、再起動します)。

Linux、macOS、Unix の場合:

```
aws rds restore-db-cluster-from-snapshot \
```

```
    --db-cluster-identifier mynewdbcluster \
    --snapshot-identifier mydbclustersnapshot \
    --engine-mode serverless --scaling-configuration
MinCapacity=8,MaxCapacity=64,TimeoutAction='ForceApplyCapacityChange',SecondsUntilAutoPause=10
```

Windows の場合:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-instance-identifier mynewdbcluster ^
    --db-snapshot-identifier mydbclustersnapshot ^
    --engine-mode serverless --scaling-configuration
MinCapacity=8,MaxCapacity=64,TimeoutAction='ForceApplyCapacityChange',SecondsUntilAutoPause=10
```

RDS API

RDS API を使用して復元する際に Aurora Serverless v1 DB クラスターを設定するには、`serverless` パラメータとして `EngineMode` を指定しながら、RestoreDBClusterFromSnapshot オペレーションを実行します。

必要に応じて `ScalingConfiguration` パラメータを指定し、最小容量、最大容量、および接続がないときの自動的な一時停止を設定できます。有効な容量値には次のようなものがあります。

- Aurora MySQL: 1、2、4、8、16、32、64、128、および 256 です。
- Aurora PostgreSQL: 2、4、8、16、32、64、192、および 384 です。

Aurora Serverless v1 DB クラスターの変更

Aurora Serverless v1 DB クラスターを設定すると、AWS Management Console、AWS CLI、または RDS API を使用して、その特定のプロパティを変更できます。変更できるプロパティのほとんどは、他の種類の Aurora クラスターと同じです。

Aurora Serverless v1 について、最も関連性の高い変更点は以下のとおりです。

- スケーリング設定の変更
- メジャーバージョンをアップグレードする
- Aurora Serverless v1 からプロビジョン済みに変換する

Aurora Serverless v1 DB クラスターのスケーリング設定の変更

DB クラスターの最小と最大の容量を設定できます。各容量ユニットは、コンピューティングとメモリに関する特定の構成と等価です。Aurora Serverless は、CPU 使用率、接続、および使用可能なメモリの各しきい値に関するスケーリングルールを自動的に作成します。アクティビティがないときにデータベースを一時停止し、アクティビティがスタートされたときにデータベースを再開するように Aurora Serverless を設定することもできます。

スケーリング設定には、以下に示す値を設定できます。

- [最小 Aurora 容量ユニット] - Aurora Serverless が容量ユニットを減らすことができる最小数です。
- [最大 Aurora 容量ユニット] - Aurora Serverless が容量ユニットを増やすことができる最大数です。
- オートスケーリングのタイムアウトとアクション - このセクションでは、Aurora Serverless がタイムアウトする前にスケーリングポイントを見つけるまでの待機時間を指定します。また、スケーリングポイントが見つからないために容量の変更がタイムアウトしたときに実行するアクションも指定します。Aurora は、容量を可能な限り早く指定した値に設定するために、容量の変更を強制できます。または、容量の変更をロールバックして、キャンセルすることができます。詳細については、「容量の変更のタイムアウトアクション」を参照してください。
- アイドル状態になった後の一時停止 — オプションの [クラスターがアイドル状態のときに容量を 0 ACUにスケーリング] 設定を使用して、非アクティブな状態のデータベースを処理キャパシティゼロにスケーリングします。データベーストラフィックが再開されると、Aurora は自動的に処理容量を再開しトラフィックを処理できるようにスケールします。

コンソール

AWS Management Console を使用して、Aurora DB クラスターのスケーリング設定を変更できます。

Aurora Serverless v1 DB クラスターを変更するには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. 変更する Aurora Serverless v1 DB クラスターを選択します。
4. [アクション] で、[クラスターの変更] を選択します。
5. [Capacity settings (容量設定)] セクションで、スケーリング設定を変更します。

6. Continue (続行) をクリックします。
7. [DB クラスターの変更] ページで、変更内容を確認し、変更を適用するタイミングを選択します。
8. [クラスタークラスターの変更] を選択します。

AWS CLI

Aurora Serverless v1 を使用して AWS CLI DB クラスターのスケーリング設定を変更するには、AWS CLI の modify-db-cluster コマンドを実行します。`--scaling-configuration` オプションを指定し、最小容量、最大容量、および接続がないときの自動的な一時停止を設定します。有効な容量値には次のようなものがあります。

- Aurora MySQL: 1、2、4、8、16、32、64、128、および 256 です。
- Aurora PostgreSQL: 2、4、8、16、32、64、192、および 384 です。

次の例では、*`sample-cluster`* という名前の Aurora Serverless v1 DB クラスターのスケーリング設定を変更しています。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=500,TimeoutAction='ForceApplyCapacityChange
```

Windows の場合:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=500,TimeoutAction='ForceApplyCapacityChange
```

RDS API

ModifyDBCluster API オペレーションを使用して Aurora DB クラスターのスケーリング設定を変更できます。`ScalingConfiguration` パラメータを指定し、最小容量、最大容量、および接続がないときの自動的な一時停止を設定します。有効な容量値には次のようなものがあります。

- Aurora MySQL: 1、2、4、8、16、32、64、128、および 256 です。
- Aurora PostgreSQL: 2、4、8、16、32、64、192、および 384 です。

Aurora Serverless v1 DB クラスターの メジャーバージョンのアップグレード

PostgreSQL 11 と互換性のある Aurora Serverless v1 DB クラスターのメジャーバージョンを、対応する PostgreSQL 13 と互換性のあるバージョンにアップグレードできます。

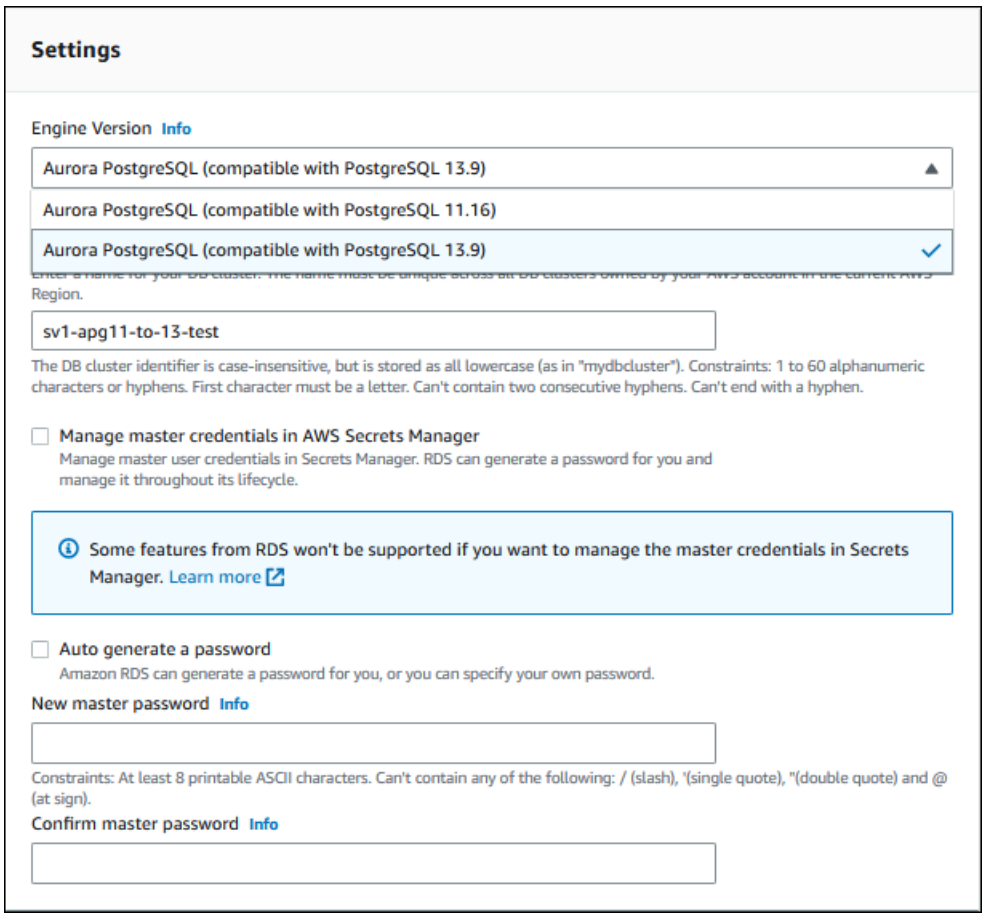
コンソール

AWS Management Console を使用して、Aurora Serverless v1 DB クラスターのインプレースアップグレードを実行できます。

Aurora Serverless v1 DB クラスターをアップグレードするには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. アップグレードする Aurora Serverless v1 DB クラスターを選択します。
4. [アクション] で、[クラスターの変更] を選択します。
5. [バージョン] として、Aurora PostgreSQL バージョン 13 のバージョン番号を選択します。

   次の例は、Aurora PostgreSQL 11.16 から 13.9 へのインプレースアップグレードを示しています。



メジャーバージョンアップグレードを実行する場合は、他のプロパティはすべて同じままにしておきます。他のプロパティを変更するには、アップグレード完了後にもう一度変更オペレーションを実行します。

6. Continue (続行) をクリックします。
7. [DB クラスターの変更] ページで、変更内容を確認し、変更を適用するタイミングを選択します。
8. [クラスタークラスターの変更] を選択します。

AWS CLI

PostgreSQL 11 互換 Aurora Serverless v1DB クラスターから PostgreSQL 13 互換 DB クラスターにインプレースアップグレードを実行するには、Aurora Serverless v1 と互換性がある Aurora PostgreSQL バージョン 13 のバージョン番号を持つ `--engine-version` パラメータを指定します。`--allow-major-version-upgrade` パラメータも含める必要があります。

この例では、sample-cluster という名前の PostgreSQL 11 互換 Aurora Serverless v1 DB クラスターのメジャーバージョンを変更します。これにより、PostgreSQL 13 互換 Aurora Serverless v1 DB クラスターへのインプレースアップグレードが実行されます。

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --engine-version 13.9 \
    --allow-major-version-upgrade
```

Windows の場合:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --engine-version 13.9 ^
    --allow-major-version-upgrade
```

RDS API

PostgreSQL 11 互換 Aurora Serverless v1DB クラスターから PostgreSQL 13 互換 DB クラスターにインプレースアップグレードを実行するには、Aurora Serverless v1 と互換性がある Aurora PostgreSQL バージョン 13 のバージョン番号を持つ EngineVersion パラメータを指定します。AllowMajorVersionUpgrade パラメータも含める必要があります。

Aurora Serverless v1 DB クラスターをプロビジョニング済みに変換する

Aurora Serverless v1 DB クラスターをプロビジョニング済み DB クラスターに変換できます。変換を実行するには、DB インスタンスクラスを [プロビジョニング済み] に変更します。この変換は、DB クラスターを Aurora Serverless v1 から Aurora Serverless v2 にアップグレードする際に使用できます。詳細については、「Aurora Serverless v1 クラスターから Aurora Serverless v2 クラスターへのアップグレード」を参照してください。

変換プロセスでは、DB クラスターにリーダー DB インスタンスを作成し、リーダーインスタンスをライターインスタンスに昇格させてから、元の Aurora Serverless v1 インスタンスを削除します。DB クラスターを変換する場合、DB エンジンのバージョンや DB クラスターパラメータグループの変更など、他の変更を同時に行うことはできません。変換オペレーションは直ちに適用され、元に戻すことはできません。

変換中に、エラーが発生した場合に備えて、DB クラスターのバックアップのスナップショットが取得されます。DB クラスタースナップショットの識別子は、pre-modify-engine-mode-*DB_cluster_identifier*-*timestamp* という形式です。

Aurora では、プロビジョニングされた DB クラスターに現在のデフォルトの DB マイナーエンジンバージョンを使用します。

変換された DB クラスターに DB インスタンスクラスを提供しない場合、Aurora は元の Aurora Serverless v1 DB クラスターの最大容量に基づいたクラスを推奨します。推奨容量とインスタンスクラスのマッピングを以下の表に示します。

Serverless 最大容量 (ACU)	プロビジョニング済み DB インスタンスクラス
1	db.t3.small
2	db.t3.medium
4	db.t3.large
8	db.r5.large
16	db.r5.xlarge
32	db.r5.2xlarge
64	db.r5.4xlarge
128	db.r5.8xlarge
192	db.r5.12xlarge
256	db.r5.16xlarge
384	db.r5.24xlarge

> **Note**
>
> 選択する DB インスタンスクラスとデータベースの使用状況によっては、Aurora Serverless v1 と比較してプロビジョニング済み DB クラスターの料金が異なる場合があります。Aurora Serverless v1 DB クラスターをバースト (db.t*) DB インスタンスクラスに変換すると、DB クラスターの使用に追加料金が発生する可能性があります。詳細については、「DB インスタンスクラスタイプ」を参照してください。

AWS CLI

Aurora Serverless v1 DB クラスターをプロビジョニング済みクラスターに変換するには、modify-db-cluster AWS CLI コマンドを実行します。

以下のパラメータは必須です。

- `--db-cluster-identifier` — プロビジョニング済みに変換している Aurora Serverless v1 DB クラスターです。
- `--engine-mode` — `provisioned` の値を使用します。
- `--allow-engine-mode-change`
- `--db-cluster-instance-class` — DB クラスターの容量に基づいて、プロビジョニング済み DB クラスターの Aurora Serverless v1 DB インスタンスクラスを選択します。

この例では、`sample-cluster` という名前の Aurora Serverless v1 DB クラスターを変換し、`db.r5.xlarge` DB インスタンスクラスを使用します。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --engine-mode provisioned \
    --allow-engine-mode-change \
    --db-cluster-instance-class db.r5.xlarge
```

Windows の場合:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --engine-mode provisioned ^
    --allow-engine-mode-change ^
    --db-cluster-instance-class db.r5.xlarge
```

RDS API

Aurora Serverless v1 DB クラスターをプロビジョニング済みクラスターに変換するには、ModifyDBCluster API オペレーションを使用します。

以下のパラメータは必須です。

- `DBClusterIdentifier` — プロビジョニング済みに変換している Aurora Serverless v1 DB クラスターです。
- `EngineMode` — `provisioned` の値を使用します。
- `AllowEngineModeChange`
- `DBClusterInstanceClass` — DB クラスターの容量に基づいて、プロビジョニング済み DB クラスターの Aurora Serverless v1 DB インスタンスクラスを選択します。

Aurora Serverless v1 DB クラスターの容量を手動でスケーリングする

通常、Aurora Serverless v1 DB クラスターはワークロードに基づいてシームレスにスケーリングされます。ただし、トランザクションの急激な増加など、突然発生する極端な事象に対応するのに十分な速度で、容量を拡張できるとは限りません。このような場合、新しい容量を値として設定することで、スケーリング操作を手動でスタートできます。容量を明示的に設定すると、Aurora Serverless v1 により、DB クラスターのスケーリングが自動的に実行されます。これは、スケーリングダウンのクールダウン期間に基づいて行われます。

Aurora Serverless v1、AWS Management Console、または RDS API を使用して、AWS CLI DB クラスターの容量を特定の値に明示的に設定できます。

コンソール

AWS Management Console を使用して、Aurora DB クラスターの容量を設定できます。

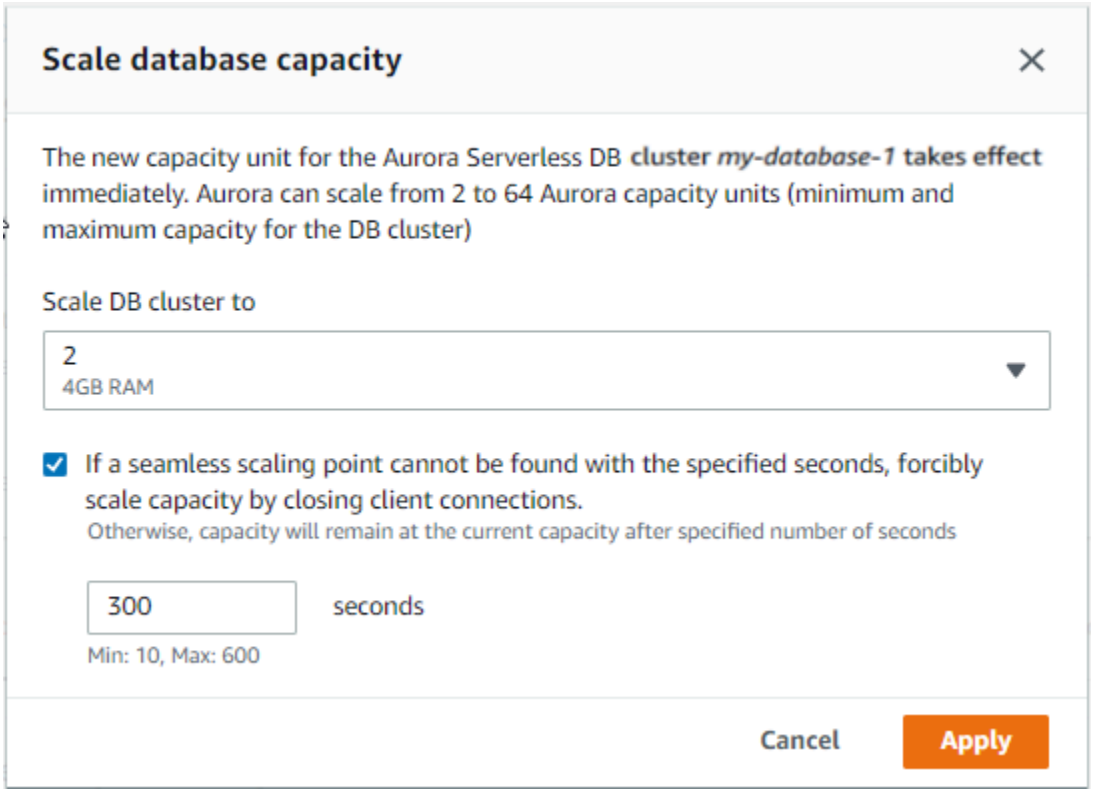
Aurora Serverless v1 DB クラスターを変更するには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. 変更する Aurora Serverless v1 DB クラスターを選択します。
4. [Actions (アクション)] で、[Set capacity (容量の設定)] を選択します。
5. データベース容量のスケーリング ウィンドウで、次の項目を選択します。
    a. [DB クラスターを次にスケーリング] ドロップダウンセレクタから、DB クラスターに必要な新しい容量を選択します。

b. [If a seamless scaling point cannot be found] (シームレススケーリングポイントが見つからない場合) チェックボックスで、Aurora Serverless v1 DB クラスターの `TimeoutAction` 設定に必要な動作を次のようにオンにします。

- タイムアウト前に Aurora Serverless v1 がスケーリングポイントを見つけられない場合に、元の容量を変更せず維持したい場合は、このオプションをオフにします。
- タイムアウト前にスケーリングポイントが見つからない場合でも、Aurora Serverless v1 DB クラスターに容量を強制的に変更させたい場合は、このオプションをオンにします。このオプションを使用すると、スケーリングポイントの検出を阻害している接続が、Aurora Serverless v1 により削除されることがあります。

c. [seconds] (秒数) に、タイムアウトする前のスケーリングポイントの検索を Aurora Serverless v1 DB クラスターに許可する時間の長さを入力します。10 秒から 600 秒 (10 分) までの任意の時間を指定できます。デフォルト設定は 5 分 (300 秒) です。次の例では、5 分以内にスケーリングポイントが見つからない場合でも、Aurora Serverless v1 DB クラスターの ACU を 2 個にスケールダウンさせます。



6. [Apply] を選択します。

スケーリングポイント、`TimeoutAction`、およびクールダウン期間の詳細については、「Aurora Serverless v1 のオートスケーリング」を参照してください。

AWS CLI

Aurora Serverless v1 を使用して AWS CLI DB クラスターの容量を設定するには、AWS CLI オプションを指定しながら、`--capacity` の modify-current-db-cluster-capacity コマンドを実行します。有効な容量値には次のようなものがあります。

- Aurora MySQL: 1、2、4、8、16、32、64、128、および 256 です。
- Aurora PostgreSQL: 2、4、8、16、32、64、192、および 384 です。

次の例では、*`sample-cluster`* という名前の Aurora Serverless v1 DB クラスターの容量値を *`64`* に設定しています。

```
aws rds modify-current-db-cluster-capacity --db-cluster-identifier sample-cluster --
capacity 64
```

RDS API

ModifyCurrentDBClusterCapacity API オペレーションを使用して Aurora DB クラスターの容量を設定できます。`Capacity` パラメータを指定します。有効な容量値には次のようなものがあります。

- Aurora MySQL: 1、2、4、8、16、32、64、128、および 256 です。
- Aurora PostgreSQL: 2、4、8、16、32、64、192、および 384 です。

Aurora Serverless v1 DB クラスターの表示

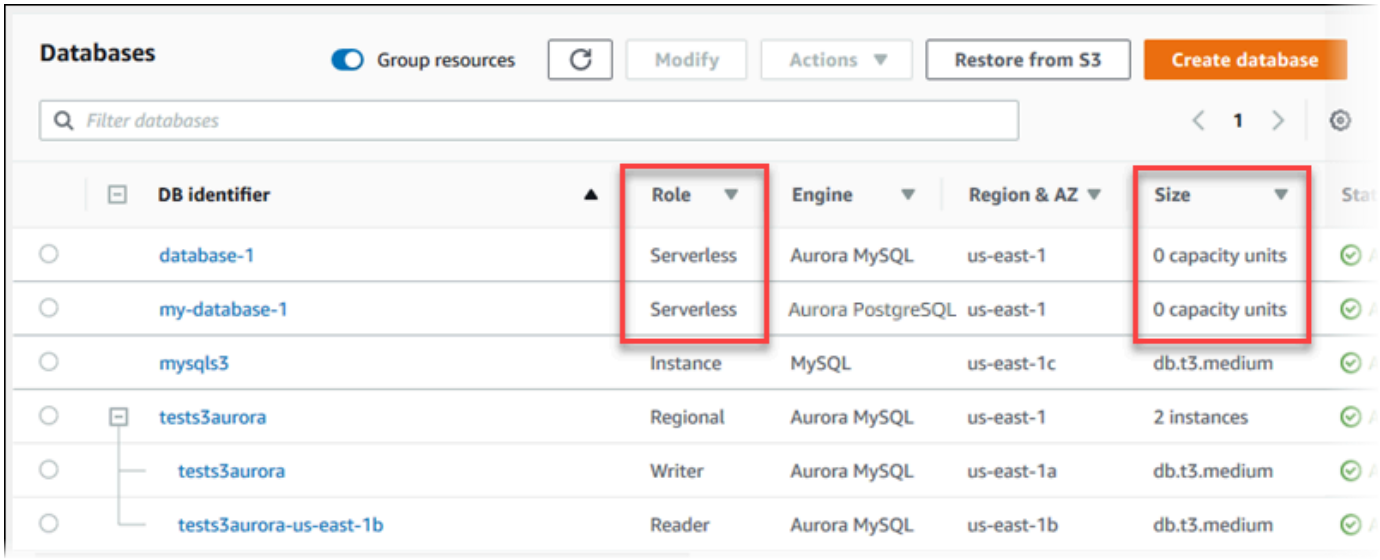
1 つ以上の Aurora Serverless v1 DB クラスターを作成している場合に、どの DB クラスターのタイプがサーバーレスであり、どの DB クラスターのタイプがインスタンスであるかを確認することができます。また、Aurora Serverless v1 DB クラスターごとに、現在使用されている Aurora 容量ユニット (ACU) の数を確認することもできます。各 ACU は、処理 (CPU) 容量とメモリ (RAM) 容量の組み合わせです。

Aurora Serverless v1 DB クラスターを表示するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。

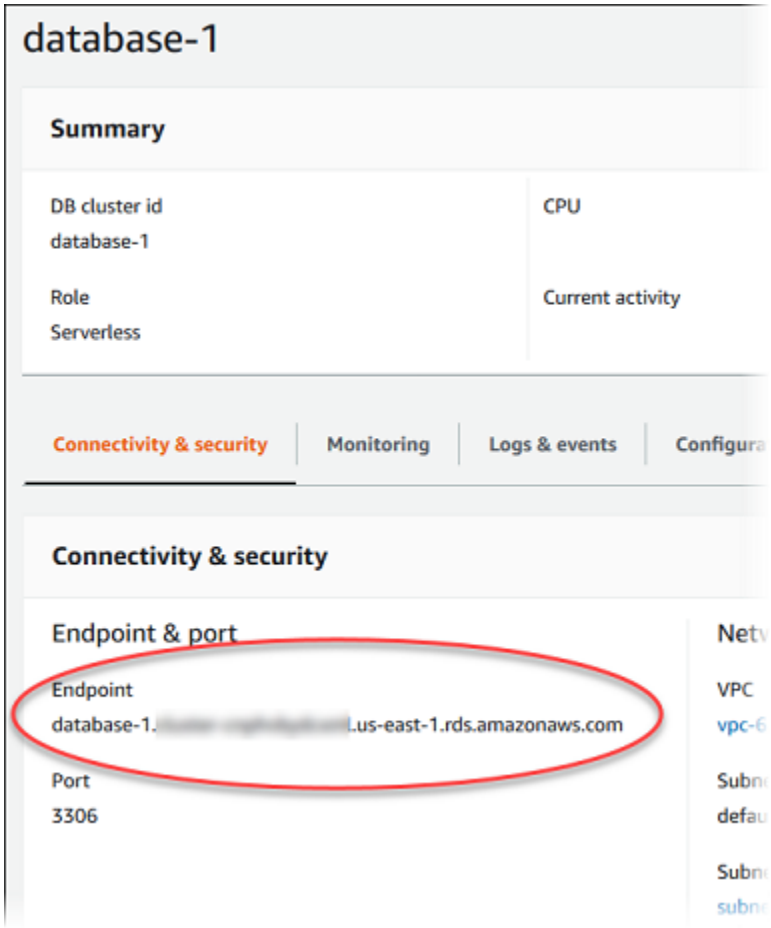
2. AWS Management Console の右上で、Aurora Serverless v1 DB クラスターを作成した AWS リージョン を選択します。
3. ナビゲーションペインで、[データベース] を選択します。

   DB クラスターごとに、DB クラスタータイプが [Role (ロール)] に表示されます。Aurora Serverless v1 DB クラスターのタイプに [サーバーレス] と表示されます。Aurora Serverless v1 DB クラスターの現在の容量は、[サイズ] で確認できます。

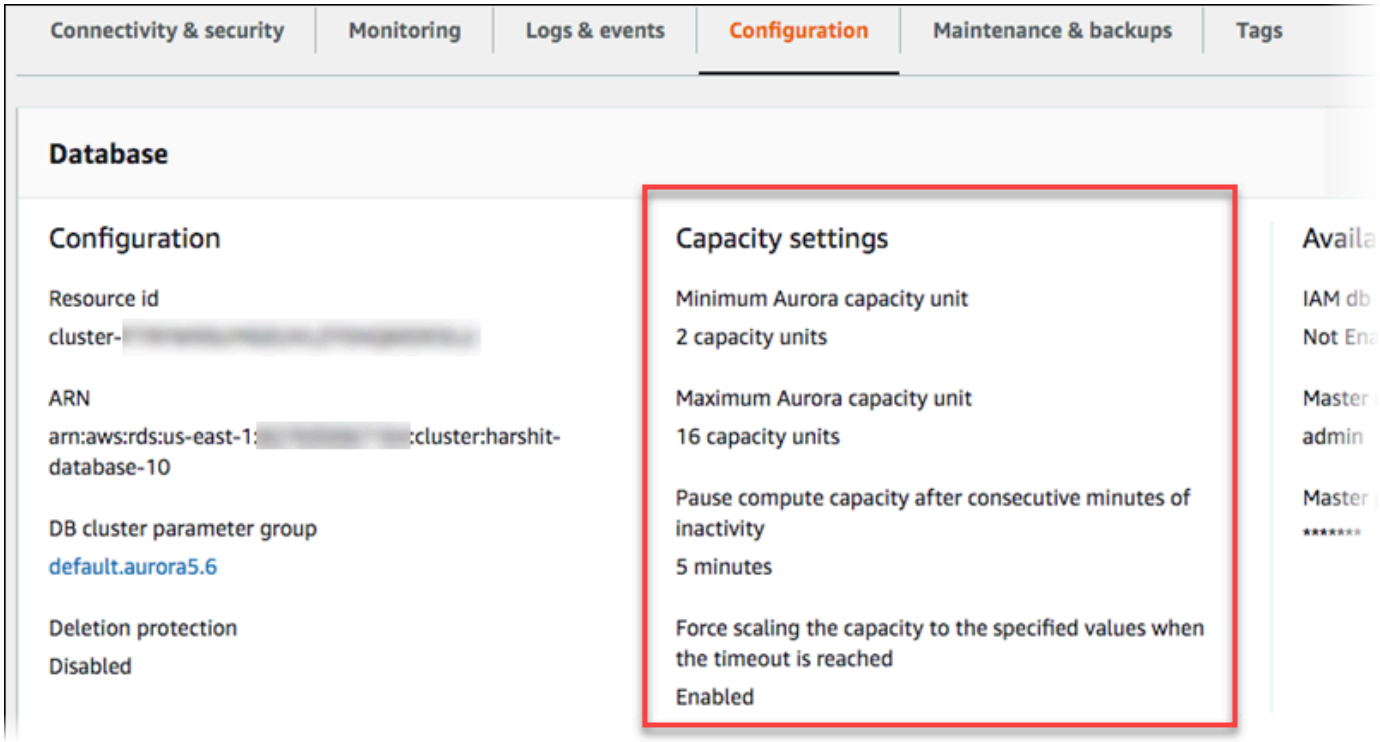


4. 詳細を表示するには、Aurora Serverless v1 DB クラスターの名前を選択します。

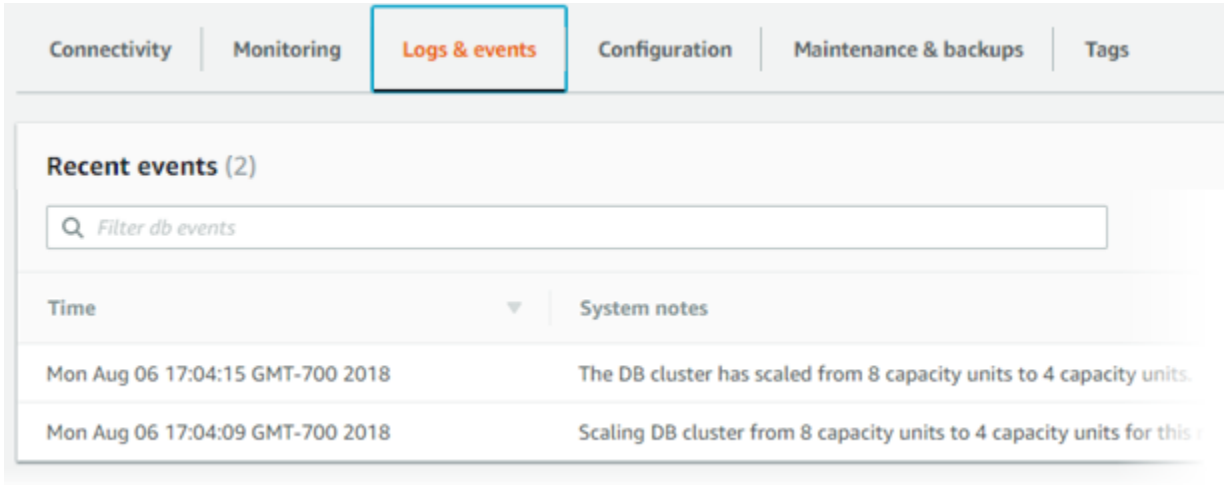
   [接続とセキュリティ] タブで、データベースエンドポイントを記録しておきます。このエンドポイントは Aurora Serverless v1 DB クラスターに接続するために使用します。



容量設定を表示するには、[設定] タブを選択します。



DB クラスターのスケールアップ、スケールダウン、一時停止、または再開に伴ってスケーリングイベントが生成されます。最近のイベントを表示するには、[ログとイベント] を選択します。これらのイベントの例を次のイメージに示します。



Aurora Serverless v1 DB クラスターの容量とスケーリングイベントのモニタリング

CloudWatch に Aurora Serverless v1 DB クラスターを表示して、`ServerlessDatabaseCapacity` メトリクスを使用しながら、その DB クラスターに割り当てられた容量をモニタリングできます。ま

た、CPUUtilization、DatabaseConnections、Queries など、Aurora CloudWatch のすべてのスタンダードメトリクスをモニタリングすることもできます。

Aurora で、一部またはすべてのデータベースログを CloudWatch に発行することができます。発行するログを選択するには、general_log クラスターに関連付けられている DB クラスターのパラメータグループで、slow_query_log や Aurora Serverless v1 などの設定パラメータを有効にします。プロビジョニングされたクラスターとは異なり、Aurora Serverless v1 クラスターでは、CloudWatch にアップロードするログタイプを DB クラスター設定で指定する必要はありません。Aurora Serverless v1 クラスターでは、使用可能なすべてのログが自動的にアップロードされます。ログ構成パラメータを無効にすると、ログの発行が CloudWatch 停止します。不要になったログは CloudWatch で削除することもできます。

Aurora Serverless v1 DB クラスターで Amazon CloudWatch の使用をスタートするには、「Amazon CloudWatch を使用して Aurora Serverless v1 ログを表示する」を参照してください。CloudWatch を使用して Aurora DB クラスターをモニタリングする方法の詳細については、「Amazon CloudWatch でログイベントをモニタリングする」を参照してください。

データベースエンドポイントを使用して、Aurora Serverless v1 DB クラスターに接続します。詳細については、「Amazon Aurora DB クラスターへの接続」を参照してください。

> **Note**
>
> Aurora Serverless v1 DB クラスター内の特定の DB インスタンスに直接接続することはできません。

Aurora Serverless v1 DB クラスターの削除

Aurora Serverless v1 を使用して AWS Management Console DB クラスターを作成すると、[デフォルトで保護を有効にする] オプションが、その選択を解除しない限り有効になります。つまり、削除保護が有効になっている Aurora Serverless v1 DB クラスターを、そのまま削除することはできません。削除保護された Aurora Serverless v1 DB クラスターを、AWS Management Console を使用して削除するには、まずクラスターを変更してこの保護を解除します。AWS CLI でこの作業を行う方法の詳細ついては、「AWS CLI」を参照してください。

AWS Management Console を使用して削除保護を無効にするには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。

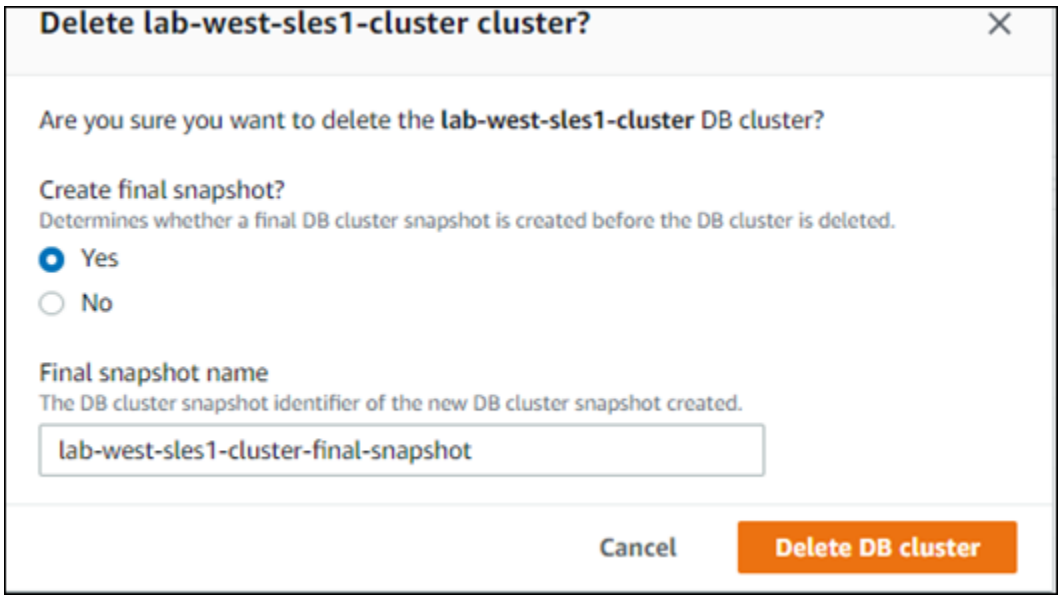
2. ナビゲーションペインで [DB クラスター] をクリックします。
3. リストから、対象の Aurora Serverless v1 DB クラスターを選択します。
4. [変更] をクリックして DB クラスターの設定を開きます。DB クラスターの変更ページで、Aurora Serverless v1 DB クラスターの設定、容量設定、およびその他の設定に関する詳細を開きます。削除保護は、[その他の設定] セクションにあります。
5. [Additional configuration] (その他の設定) のプロパティカードで、[Enable deletion protection] (削除保護を有効にする) チェックボックスをオフにします。
6. [Continue] を選択します。[変更の概要] が表示されます。
7. [クラスターの変更] をクリックして、変更の概要の内容を受け入れます。[戻る] をクリックして設定を修正したり、[キャンセル] をクリックして変更を破棄することもできます。

削除保護がアクティブでなくなったら、Aurora Serverless v1 を使用して AWS Management Console DB クラスターを削除できるようになります。

コンソール

Aurora Serverless v1 DB クラスターを削除するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. [リソース] セクションで、[DB クラスター] をクリックします。
3. 削除する Aurora Serverless v1 DB クラスターを選択します。
4. [アクション] で、[削除] を選択します。Aurora Serverless v1 DB クラスターを削除することを確認するプロンプトが表示されます。
5. 次のように、既に選択されているオプションを維持することを推奨します。
   - [最終スナップショットを作成しますか?] には [はい]
   - [最終スナップショット名] には Aurora Serverless v1 DB クラスターの名前と `-final-snapshot`。ただし、このフィールドでは、最終スナップショットの名前を変更できます。



[最終スナップショットを作成しますか?] で [いいえ] を選択している場合は、スナップショットまたはポイントインタイムリカバリを使用して、DB クラスターを復元することはできません。

6. [DB クラスターの削除] をクリックします。

Aurora Serverless v1DB クラスターが、 により削除されます。最終スナップショットを作成することを選択した場合は、削除される前に Aurora Serverless v1 DB クラスターのステータスが "Backing-up" に変わり、そのクラスターはリストに表示されなくなります。

AWS CLI

この作業を開始する前に、AWS アクセスキー ID、AWS シークレットアクセスキー、および Aurora Serverless v1 DB クラスターが置かれる AWS リージョン を指定しながら AWS CLI を設定します。詳細については、「AWS Command Line Interface ユーザーガイド」の「設定の基本」を参照してください。

この、Aurora Serverless v1 DB クラスターのオプションで設定された削除保護を無効にするまで、そのクラスターを削除することはできません。保護オプションが有効になっているクラスターを削除しようとすると、次のエラーメッセージが表示されます。

```
An error occurred (InvalidParameterCombination) when calling the DeleteDBCluster
  operation: Cannot delete protected Cluster, please disable deletion protection and
 try again.
```

次に示すように、Aurora Serverless v1 の modify-db-cluster コマンドを使用して、AWS CLI DB クラスターの削除保護の設定を変更することができます。

```
aws rds modify-db-cluster --db-cluster-identifier your-cluster-name --no-deletion-
protection
```

このコマンドは、指定された DB クラスターの変更済みとなったプロパティを返します。これで、Aurora Serverless v1 DB クラスターを削除できます。

Aurora Serverless v1 DB クラスターを削除するたびに、必ず最終スナップショットを作成することを推奨します。次に、AWS CLI の delete-db-cluster を使用する例を示します。DB クラスターの名前とスナップショットの名前を指定します。

Linux、macOS、Unix の場合:

```
aws rds delete-db-cluster --db-cluster-identifier \
  your-cluster-name --no-skip-final-snapshot \
  --final-db-snapshot-identifier name-your-snapshot
```

Windows の場合:

```
aws rds delete-db-cluster --db-cluster-identifier ^
  your-cluster-name --no-skip-final-snapshot ^
  --final-db-snapshot-identifier name-your-snapshot
```

Aurora Serverless v1 とAurora データベースエンジンのバージョン

Aurora Serverless v1 を使用できるのは、一部の AWS リージョン と、特定の Aurora MySQL および Aurora PostgreSQL のバージョンのみです。現在 Aurora Serverless v1 がサポートされている AWS リージョン のリスト、 および各リージョンで利用可能な、Aurora MySQL と Aurora PostgreSQL の特定のバージョンについては、「Aurora Serverless v1 でサポートされているリージョンと Aurora DB エンジン」を参照してください。

次のように、Aurora Serverless v1 では、対応する Aurora データベースエンジンを使用して、サポートされているデータベースエンジンごとにサポートされている特定のリリースを識別します。

- Aurora MySQL サーバーレス
- Aurora PostgreSQL サーバーレス

データベースエンジンのマイナーリリースが Aurora Serverless v1 で使用可能になると、それらのリリースは、Aurora Serverless v1 が利用可能なさまざまな AWS リージョン に対し自動的に適用されます。つまり、 クラスターの DB エンジン用として Aurora Serverless v1 で利用可能になった新しいマイナーリリースを取得するために、Aurora Serverless v1 DB クラスターをアップグレードする必要はありません。

Aurora MySQL サーバーレス

Aurora Serverless v1 DB クラスターに Aurora MySQL 互換エディションを使用する場合は、MySQL 5.7 と互換性のある Aurora MySQL バージョン 2 を選択できます。Aurora MySQL バージョン 2 の機能強化とバグ修正については、「Aurora MySQL のリリースノート」の「Amazon Aurora MySQL バージョン 2 のデータベースエンジンの更新」を参照してください。

Aurora PostgreSQL サーバーレス

Aurora Serverless v1 DB クラスターに Aurora PostgreSQL を使用する場合、Aurora PostgreSQL 11 互換バージョンと 13 互換バージョンから選択できます。Aurora PostgreSQL 互換エディションのマイナーリリースには、下位互換性のある変更のみが含まれます。AWS リージョン で、Aurora PostgreSQL のマイナーリリースが Aurora Serverless v1 用にリリースされると、Aurora Serverless v1 DB クラスターは透過的にアップグレードされます。

例えば、マイナーバージョンの Aurora PostgreSQL 11.16 リリースは、以前の Aurora PostgreSQL バージョンを実行していたすべての Aurora Serverless v1 DB クラスターに透過的に適用されました。Aurora PostgreSQL バージョン 11.16 での更新の詳細については、「Aurora PostgreSQL のリリースノート」の「PostgreSQL 11.16」を参照してください。

RDS Data API の使用

RDS Data API (Data API) を使用すると、Aurora DB クラスターへのウェブサービスインターフェイスを操作できます。Data API は、DB クラスターへの永続的な接続を必要としません。代わりに、セキュア HTTP エンドポイントおよび AWS SDK との統合を利用できます。エンドポイントを使用して、接続を管理せずに SQL ステートメントを実行することができます。

Data API へのすべての呼び出しは同期的です。デフォルトでは、呼び出しの処理が 45 秒以内に完了していない場合、呼び出しはタイムアウトします。ただし、`continueAfterTimeout` パラメータを使用して呼び出しがタイムアウトになった場合は、SQL ステートメントの実行を続行できます。例については、「SQL トランザクションの実行」を参照してください。

Data API は AWS Secrets Manager に保存されたデータベース認証情報を使用するため、ユーザーは Data API の呼び出しで認証情報を渡す必要はありません。Secrets Manager に認証情報を保存するには、Secrets Manager を使用する適切なアクセス許可に加えて Data API もユーザーに付与されている必要があります。ユーザー承認の詳細については、RDS Data API へのアクセスの承認 を参照してください。

Data API を使用して、Amazon Aurora を他の AWS アプリケーション (AWS Lambda、AWS AppSync、および AWS Cloud9 など) と統合することもできます。Data API では、AWS Lambda をより安全に使用することができます。Virtual Private Cloud (VPC) 内のリソースにアクセスするように Lambda 関数を設定しなくても、DB クラスターにアクセスすることができます。詳細については、「AWS Lambda」、「AWS AppSync」、および「AWS Cloud9」を参照してください。

Data API は、Aurora DB クラスターの作成時に有効にできます。後で設定を変更することもできます。詳細については、「RDS Data API の有効化」を参照してください。

Data API を有効にした後、クエリエディタを使用して、VPC 内の Aurora にアクセスするようにクエリツールを設定しなくても、アドホッククエリを実行することもできます。詳細については、「クエリエディタの使用」を参照してください。

トピック

- リージョンとバージョンの可用性
- RDS Data API の制限事項
- RDS Data API と Serverless v2 およびプロビジョンド、および Aurora Serverless v1 の比較
- RDS Data API へのアクセスの承認

- RDS Data API の有効化
- RDS Data API の Amazon VPC エンドポイント (AWS PrivateLink) の作成
- RDS Data API の呼び出し
- RDS Data API 用の Java クライアントライブラリの使用
- JSON 形式でクエリ結果を処理する
- RDS Data API の問題のトラブルシューティング
- AWS CloudTrail での RDS Data API コールのログ記録

リージョンとバージョンの可用性

Data API で使用できるリージョンとエンジンのバージョンについては、以下のセクションを参照してください。

クラスタータイプ	リージョンとバージョンの可用性
Aurora PostgreSQL プロビジョンドおよび Serverless v2	Aurora PostgreSQL Serverless v2 とプロビジョニングされた Data API
Aurora PostgreSQL Serverless v1	Aurora PostgreSQL Serverless v1 を使用する Data API
Aurora MySQL Serverless v1	Aurora MySQL Serverless v1 で Data API を使用する

> **Note**
>
> 現在、Data API は Aurora MySQL プロビジョンド DB クラスターまたは Serverless v2 DB クラスターでは使用できません。

コマンドラインインターフェイスまたは API を使用してデータ API にアクセスするときに FIPS 140-2 検証済みの暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-2」を参照してください。

RDS Data API の制限事項

RDS Data API (Data API) には以下の制限があります。

- Data API クエリは、DB クラスターのライターインスタンスでのみ実行できます。ただし、ライターインスタンスは書き込みクエリと読み取りクエリの両方を受け入れることができます。
- Aurora グローバルデータベースでは、プライマリ DB クラスターとセカンダリ DB クラスターの両方で Data API を有効にできます。ただし、セカンダリクラスターがプライマリに昇格されるまで、ライターインスタンスはありません。したがって、セカンダリに送信する Data API クエリは失敗します。昇格したセカンダリに使用可能なライターインスタンスが作成されると、その DB インスタンスに対する Data API クエリが成功します。
- Performance Insights は、Data API を使用して行ったデータベースクエリのモニタリングをサポートしていません。
- Data API は T DB インスタンスクラスではサポートされていません。
- Aurora PostgreSQL Serverless v2 およびプロビジョニングされた DB クラスターの場合、RDS Data API は一部のデータ型をサポートしていません。サポートされているタイプのリストについては、「the section called “Serverless v2 およびプロビジョンド、および Aurora Serverless v1 の比較”」を参照してください。
- Aurora PostgreSQL バージョン 14 以降のデータベースでは、Data API はパスワード暗号化のために scram-sha-256 のみをサポートします。

RDS Data API と Serverless v2 およびプロビジョンド、および Aurora Serverless v1 の比較

次の表は、RDS Data API (Data API) と Aurora PostgreSQL Serverless v2 とプロビジョニングされた DB クラスター、および Aurora Serverless v1 DB クラスターの違いを示しています。

違い	Aurora PostgreSQL Serverless v2 とプロビジョンド	Aurora Serverless v1
1 秒あたりの最大リクエスト数。	無制限	1,000
RDS API または AWS CLI を使用する既存のデータベース	• RDS API – `EnableHttpEndpoint` と	• RDS API – `ModifyDBCluster` オペレーション

違い	Aurora PostgreSQL Serverless v2 とプロビジョンド	Aurora Serverless v1
での Data API の有効化または無効化	`DisableHttpEndpoint` オペレーションを使用します。 • AWS CLI – `enable-http-endpoint` と `disable-http-endpoint` オペレーションを使用します。	を使用し、必要に応じて `EnableHttpEndpoint` パラメータに `true` または `false` を指定します。 • AWS CLI – 必要に応じて、`--enable-http-endpoint` または `--no-enable-http-endpoint` オプションを指定して `modify-db-cluster` オペレーションを使用します。
CloudTrail のイベント	Data API コールからのイベントはデータイベントです。これらのイベントは、デフォルトで証跡で自動的に除外されます。詳細については、「the section called “CloudTrail の証跡からの Data API イベントの包含”」を参照してください。	Data API コールからのイベントは管理イベントです。これらのイベントは、デフォルトで証跡で自動的に含まれます。詳細については、「the section called “CloudTrail の証跡からの Data API イベントの除外 (Aurora Serverless v1 のみ)”」を参照してください。
マルチステートメントのサポート	マルチステートメントはサポートされていません。この場合、Data API は `ValidationException: Multistatements aren't supported` をスローします。	Aurora PostgreSQL の場合、マルチステートメントは最初のクエリレスポンスのみを返します。Aurora MySQL では、マルチステートメントはサポートされていません。
BatchExecuteStatement	更新結果で生成されたフィールドオブジェクトは空です。	更新結果で生成されたフィールドオブジェクトには、挿入された値が含まれます。

違い	Aurora PostgreSQL Serverless v2 とプロビジョンド	Aurora Serverless v1
ExecuteSQL	サポートされていません	廃止済み

違い	Aurora PostgreSQL Serverless v2 とプロビジョンド	Aurora Serverless v1
ExecuteStatement	`ExecuteStatement` は、多次元配列の列の取得をサポートしていません。この場合、Data API は `UnsupportedResultException` をスローします。 Data API は、ジオメトリ型や金額型など、一部のデータ型をサポートしていません。この場合、Data API は `UnsupportedResultException: The result contains the unsupported data type` *`data_type`* をスローします。 次のタイプのみがサポートされています。 • `BOOL` • `BYTEA` • `DATE` • `CIDR` • `DECIMAL`, `NUMERIC` • `ENUM` • `FLOAT8`, `DOUBLE PRECISION` • `INET` • `INT`, `INT4`, `SERIAL`	`ExecuteStatement` は、多次元配列の列とすべての高度なデータ型の取得をサポートします。

違い	Aurora PostgreSQL Serverless v2 とプロビジョンド	Aurora Serverless v1
	• INT2, SMALLINT, SMALLSERIAL • INT8, BIGINT, BIGSERIAL • JSONB, JSON • REAL, FLOAT • TEXT, CHAR(N), VARCHAR, NAME • TIME • TIMESTAMP • UUID • VECTOR 次の配列型のみがサポートされています。 • BOOL[], BIT[] • DATE[] • DECIMAL[] , NUMERIC[] • FLOAT8[], DOUBLE PRECISION[] • INT[], INT4[] • INT2[] • INT8[], BIGINT[] • JSON[] • REAL[], FLOAT[] • TEXT[], CHAR(N)[] , VARCHAR[] , NAME[] • TIME[] • TIMESTAMP[]	

違い	Aurora PostgreSQL Serverless v2 とプロビジョンド	Aurora Serverless v1
	• UUID[]	

RDS Data API へのアクセスの承認

ユーザーは、許可されている場合にのみ RDS Data API (Data API) オペレーションを呼び出すことができます。アクセス許可を定義する AWS Identity and Access Management (IAM) ポリシーをアタッチすることで、Data API を使用するアクセス許可をユーザーに付与できます。IAM ロールを使用している場合は、ポリシーをロールにアタッチすることもできます。AWS 管理ポリシー AmazonRDSDataFullAccess には、Data API のアクセス許可が含まれています。

AmazonRDSDataFullAccess ポリシーには、ユーザーが AWS Secrets Manager からシークレットの値を取得するためのアクセス許可も含まれています。ユーザーは、Secrets Manager を使用して、Data API の呼び出しで使用できるシークレットを保存する必要があります。シークレットを使用すると、ユーザーは Data API の呼び出しでターゲットとするリソースのデータベース認証情報を含める必要がなくなります。Data API は透過的に Secrets Manager を呼び出し、シークレットに対するユーザーのリクエストを許可 (または拒否) します。Data API で使用するシークレットの設定については、「AWS Secrets Manager へのデータベース認証情報の保存」を参照してください。

AmazonRDSDataFullAccess ポリシーは、(Data API を介して) リソースへの完全なアクセスを提供します。リソースの Amazon リソースネーム (ARN) を指定する独自のポリシーを定義することで、スコープを絞り込むことができます。

例えば、次のポリシーは、ARN によって識別される DB クラスターの Data API にアクセスするためにユーザーが最低限必要なアクセス許可の例を示しています。このポリシーには、Secrets Manager にアクセスし、ユーザーの DB インスタンスの承認を取得するために必要なアクセス許可が含まれます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SecretsManagerDbCredentialsAccess",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
```

```
            "Resource": "arn:aws:secretsmanager:*:*:secret:rds-db-credentials/*"
        },
        {
            "Sid": "RDSDataServiceAccess",
            "Effect": "Allow",
            "Action": [
                "rds-data:BatchExecuteStatement",
                "rds-data:BeginTransaction",
                "rds-data:CommitTransaction",
                "rds-data:ExecuteStatement",
                "rds-data:RollbackTransaction"
            ],
            "Resource": "arn:aws:rds:us-east-2:111122223333:cluster:prod"
        }
    ]
}
```

ポリシーステートメントの「Resources」要素には、ワイルドカード (*) ではなく、特定の ARN を使用することをお勧めします (例を参照)。

タグベースの承認の操作

RDS Data API (Data API) と Secrets Manager はどちらも、タグベースの認証をサポートしています。タグは、RDS クラスターなどのリソースに、追加の文字列値を使用してラベル付けするキーと値のペアです。次に例を示します。

- `environment:production`
- `environment:development`

コスト配分、オペレーションサポート、アクセス制御など、さまざまな理由でリソースにタグを適用できます。(リソースにタグがまだないため、タグを適用する場合は、「Amazon RDS リソースにタグを付ける」で詳細を参照してください)。ポリシーステートメントでタグを使用して、これらのタグでラベル付けされた RDS クラスターへのアクセスを制限できます。例として、Aurora DB クラスターには、その環境を本番環境または開発環境として識別するタグが存在する場合があります。

以下の例は、ポリシーステートメントでタグを使用する方法を示しています。このステートメントでは、Data API リクエストで渡されたクラスターとシークレットの両方に `environment:production` タグが必要です。

ポリシーが適用される方法は次のとおりです。ユーザーが Data API を使用して呼び出しを行うと、リクエストがサービスに送信されます。Data API はまず、リクエストで渡されたクラスター ARN が environment:production でタグ付けされていることを確認します。次に、Secrets Manager を呼び出し、リクエスト内のユーザーのシークレット値を取得します。また、Secrets Manager は environment:production を使用して、ユーザーのシークレットがタグ付け済みかどうかを検証します。その場合、Data API は取得した値をユーザーの DB パスワードに使用します。その値も正しい場合、最後にユーザーに対して Data API リクエストが正常に呼び出されます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SecretsManagerDbCredentialsAccess",
            "Effect": "Allow",
            "Action": [
                 "secretsmanager:GetSecretValue"
               ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:rds-db-credentials/*",
            "Condition": {
                    "StringEquals": {
                        "aws:ResourceTag/environment": [
                                        "production"
                                       ]
                     }
             }
        },
        {
            "Sid": "RDSDataServiceAccess",
            "Effect": "Allow",
            "Action": [
                  "rds-data:*"
               ],
            "Resource": "arn:aws:rds:us-east-2:111122223333:cluster:*",
            "Condition": {
                    "StringEquals": {
                        "aws:ResourceTag/environment": [
                                        "production"
                                       ]
                     }
             }
         }
     ]
```

```
}
```

この例は、Data API と Secrets Manager の `rds-data` および `secretsmanager` に対する個別のアクションを示しています。ただし、特定のユースケースをサポートするために、さまざまな方法でアクションを組み合わせたり、タグ条件を定義したりできます。詳細については、「Secrets Manager でアイデンティティベースのポリシー (IAM ポリシー) を使用する」を参照してください。

ポリシーの「Condition」要素では、次の項目の中からタグキーを選択できます。

- `aws:TagKeys`
- `aws:ResourceTag/${TagKey}`

リソースタグについてと `aws:TagKeys` の使用方法の詳細については、「リソースタグを使用した AWS リソースへのアクセスの制御」を参照してください。

> **Note**
>
> Data API と AWS Secrets Manager のどちらもユーザーを承認します。ポリシーで定義されているすべてのアクションに対するアクセス許可を持っていない場合は、`AccessDeniedException` エラーが発生します。

AWS Secrets Manager へのデータベース認証情報の保存

RDS Data API (Data API) を呼び出すと、Secrets Manager のシークレットを使用して Aurora DB クラスターの認証情報を渡すことができます。この方法で認証情報を渡すには、シークレットの名前またはシークレットの Amazon リソースネーム (ARN) を指定します。

DB クラスター認証情報をシークレットに保存するには

1. Secrets Manager を使用して Aurora DB クラスターの認証情報が含まれているシークレットを作成します。

   手順については、AWS Secrets Managerユーザーガイド の「データベースシークレットを作成する」を参照してください。
2. Secrets Manager コンソールを使用して、作成したシークレットの詳細を表示するか、AWS CLI の `aws secretsmanager describe-secret` コマンドを実行します。

シークレットの名前と ARN を書き留めます。これらは、Data API への呼び出しで使用できます。

Secrets Manager の使用の詳細については、AWSSecrets Manager ユーザーガイドを参照してください。

Amazon Aurora が Identity and Access Management をどのように管理するかについては、「Amazon Aurora で IAM を使用する方法」を参照してください。

IAM ポリシーの作成方法の詳細については、『IAM ユーザーガイド』の「IAM ポリシーの作成」を参照してください。IAM ポリシーをユーザーに追加する方法については、IAM ユーザーガイドのIAMアイデンティティ許可の追加および削除を参照してください。

RDS Data API の有効化

RDS Data API (Data API) を使用するには、Aurora DB クラスター用に有効にする必要があります。Data API の有効化は、DB クラスターの作成時または変更時に行うことができます。

> **Note**
>
> Aurora PostgreSQL の場合、Data API は Aurora Serverless v2、Aurora Serverless v1、および でプロビジョニングされたデータベースでサポートされています。Aurora MySQL では、Data API は Aurora Serverless v1 データベースでのみサポートされています。

トピック

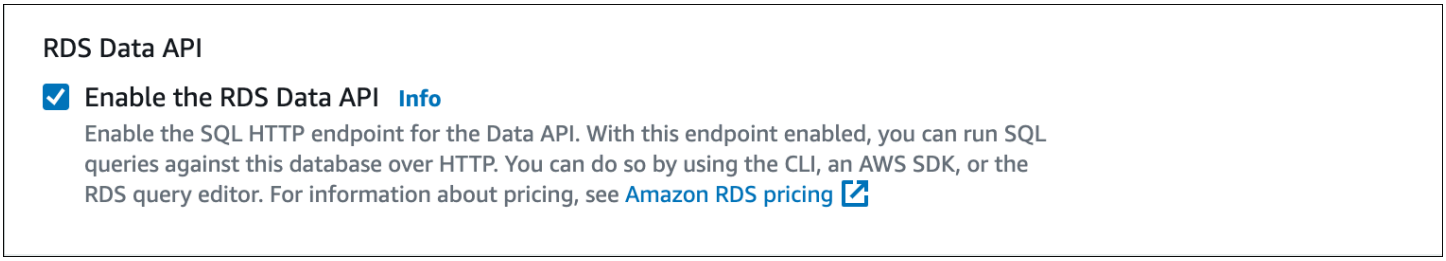
- データベースの作成時の RDS Data API の有効化
- 既存のデータベースでの RDS Data API の有効化

データベースの作成時の RDS Data API の有効化

RDS Data API (Data API) をサポートするデータベースを作成するときに、この機能を有効にできます。次の手順では、AWS Management Console、AWS CLI または RDS API を使用するときにこれを行う方法について説明します。

コンソール

DB クラスターの作成時に Data API を有効にするには、次のスクリーンショットのように、[データベースの作成] ページの [接続] セクションで [RDS Data API を有効にする] チェックボックスをオンにします。



データベースの作成方法に関する手順については、以下を参照してください。

- Aurora PostgreSQL Serverless v2 とプロビジョンドデータベース – Amazon Aurora DB クラスターの作成
- Aurora Serverless v1 – Aurora Serverless v1 DB クラスターの作成 の場合

AWS CLI

Aurora DB クラスターの作成中に Data API を有効にするには、`--enable-http-endpoint` オプションを指定して create-db-cluster AWS CLI コマンドを実行します。

次の例では、Data API を有効にして Aurora PostgreSQL DB クラスターを作成します。

Linux、macOS、Unix の場合:

```
aws rds create-db-cluster \
    --db-cluster-identifier my_pg_cluster \
    --engine aurora-postgresql \
    --enable-http-endpoint
```

Windows の場合:

```
aws rds create-db-cluster ^
    --db-cluster-identifier my_pg_cluster ^
    --engine aurora-postgresql ^
    --enable-http-endpoint
```

RDS API

Aurora DB クラスターの作成中に Data API を有効にするには、`EnableHttpEndpoint` パラメータの値を `true` に設定して CreateDBCluster オペレーションを使用します。

既存のデータベースでの RDS Data API の有効化

RDS Data API (Data API) をサポートする DB クラスターを変更して、この機能を有効または無効にできます。

トピック

- Data API (Aurora PostgreSQL Serverless v2 およびプロビジョンド) の有効化と無効化
- Data API の有効化または無効化 (Aurora Serverless v1 のみ)

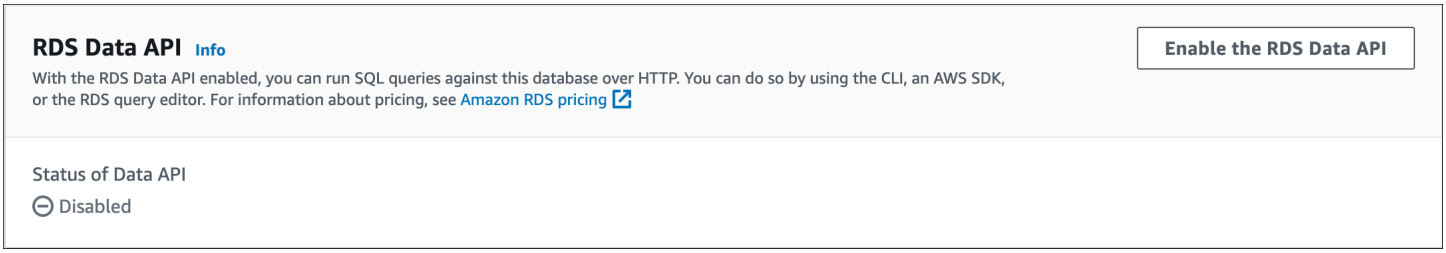
Data API (Aurora PostgreSQL Serverless v2 およびプロビジョンド) の有効化と無効化

Aurora PostgreSQL Serverless v2 およびプロビジョニングされたデータベースで Data API を有効または無効にするには、次の手順に従います。Aurora Serverless v1 データベースで Data API を有効または無効にするには、「the section called “Data API の有効化または無効化 (Aurora Serverless v1 のみ)”」の手順を使用します。

コンソール

Data API を有効または無効にするには、この機能をサポートする DB クラスターの RDS コンソールを使用します。これを行うには、Data API を有効または無効にするデータベースのクラスター詳細ページを開き、[接続とセキュリティ] タブで [RDS Data API] セクションに移動します。このセクションでは、Data API のステータスを表示し、それを有効または無効にできます。

次のスクリーンショットは、[RDS Data API] が有効になっていないことを示しています。



AWS CLI

既存のデータベースで Data API を有効または無効にするには、enable-http-endpoint または disable-http-endpoint AWS CLI コマンドを実行し、DB クラスターの ARN を指定します。

次の例では、Data API を有効にします。

Linux、macOS、Unix の場合:

```
aws rds enable-http-endpoint \
    --resource-arn cluster_arn
```

Windows の場合:

```
aws rds enable-http-endpoint ^
    --resource-arn cluster_arn
```

RDS API

既存のデータベースで Data API を有効または無効にするには、EnableHttpEndpoint オペレーションと DisableHttpEndpoint オペレーションを使用します。

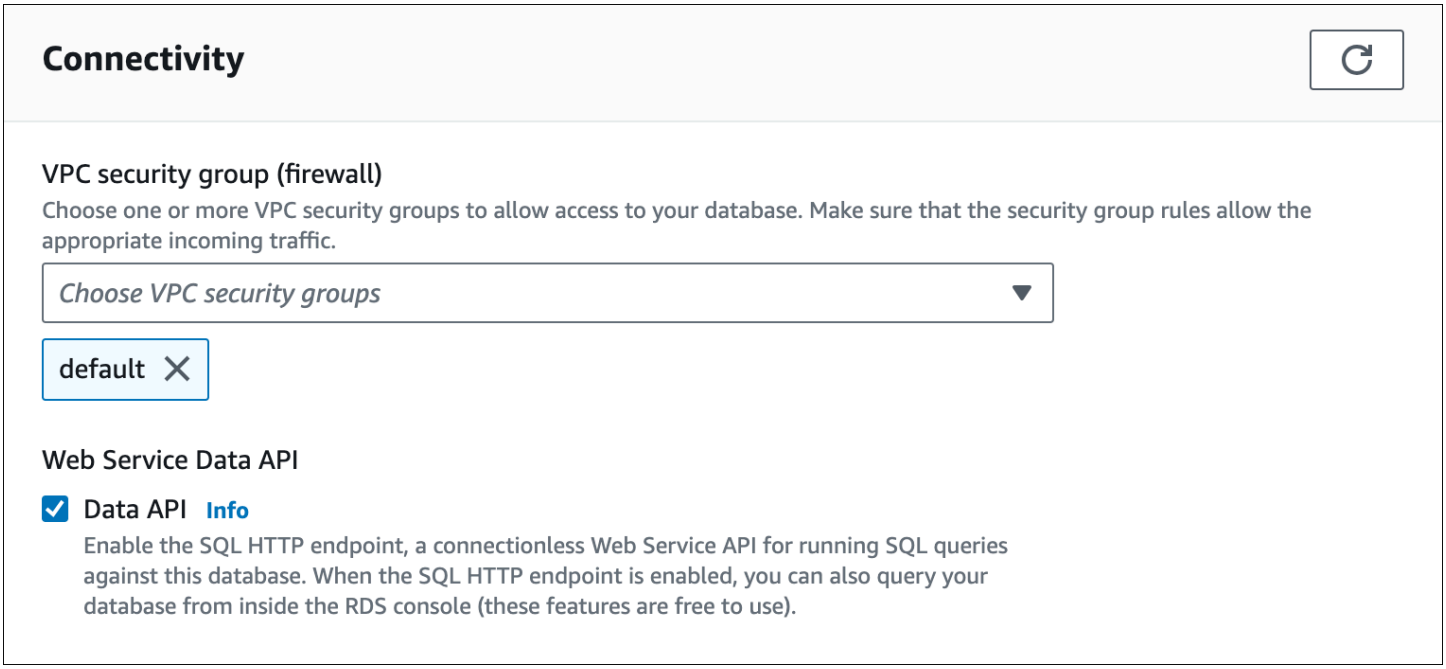
Data API の有効化または無効化 (Aurora Serverless v1 のみ)

Aurora Serverless v1 データベースで Data API を有効または無効にするには、次の手順を使用します。Aurora PostgreSQL Serverless v2 およびプロビジョニングされたデータベースで Data API を有効または無効にするには、the section called “Data API の有効化または無効化” の手順を使用します。

コンソール

Aurora Serverless v1 DB クラスターの変更時には、RDS コンソールの [接続] セクションで Data API を有効にします。

次のスクリーンショットは、Aurora DB クラスターを変更時に有効になった [Data API] を示しています。



Aurora Serverless v1 DB クラスターを変更する手順については、「Aurora Serverless v1 DB クラスターの変更」を参照してください。

AWS CLI

Data API を有効または無効にするには、必要に応じて `--enable-http-endpoint`または `--no-enable-http-endpoint` を使用して modify-db-cluster AWS CLI コマンドを実行します。

次の例では、`sample-cluster` で Data API を有効にします。

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --enable-http-endpoint
```

Windows の場合:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --enable-http-endpoint
```

RDS API

Data API を有効にするには、ModifyDBCluster オペレーションを使用し、必要に応じて `EnableHttpEndpoint` の値を `true` または `false` に設定します。

RDS Data API の Amazon VPC エンドポイント (AWS PrivateLink) の作成

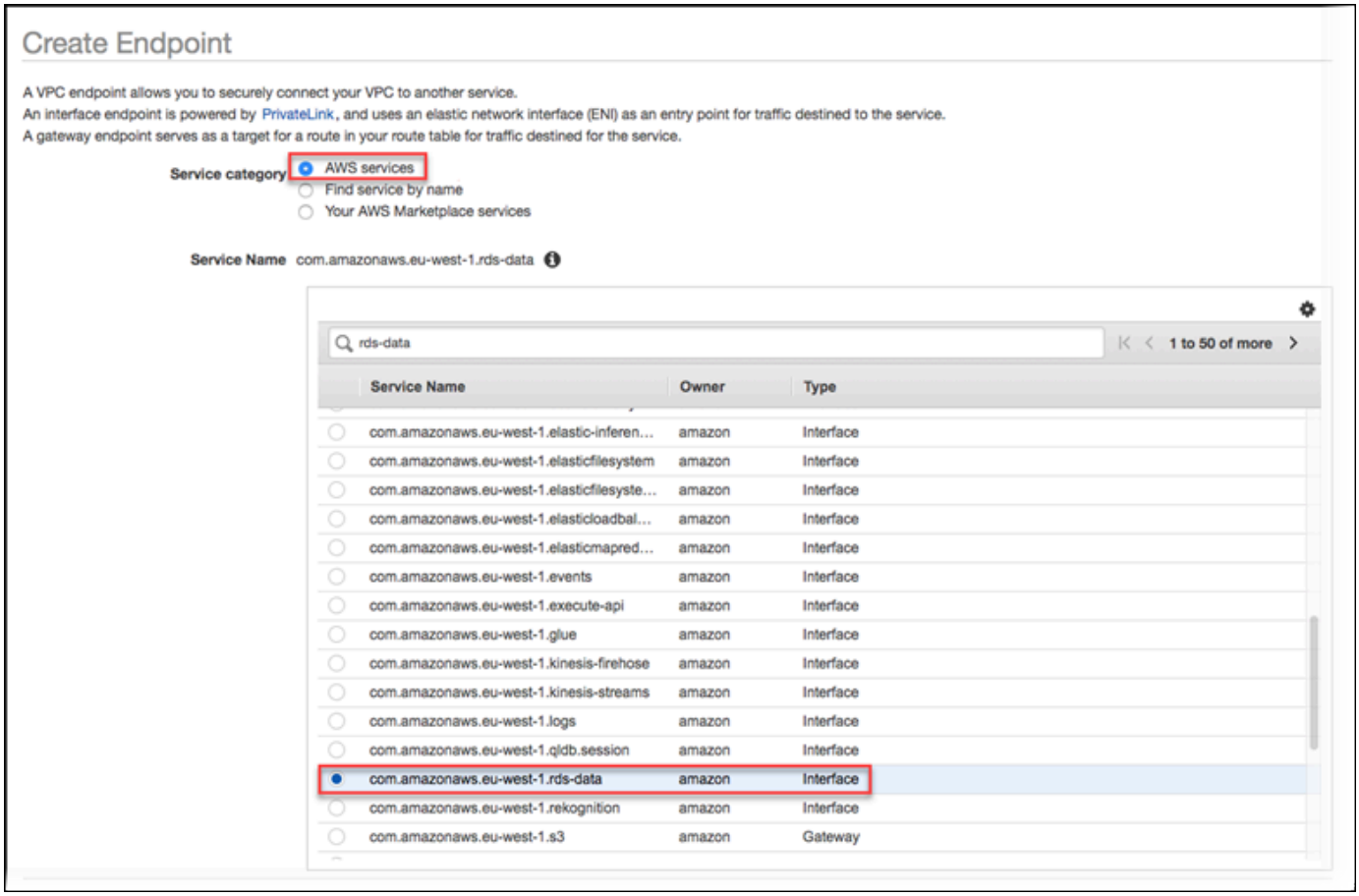
Amazon VPC を使用すると、Aurora DB クラスターやアプリケーションなどの AWS リソースを Virtual Private Cloud (VPC) 内に起動できます。AWS PrivateLink は、VPC と AWS のサービス間のプライベート接続を、Amazon ネットワーク上で高い安全性とともに提供します。AWS PrivateLink を使用すると、Amazon VPC エンドポイントを作成できます。これにより、Amazon VPC に基づいて、異なるアカウントと VPC のサービスに接続することが可能になります。AWS PrivateLink の詳細については、Amazon Virtual Private Cloud ユーザーガイドの「VPC エンドポイントサービス (AWS PrivateLink)」を参照してください。

Amazon VPC エンドポイントを使用して RDS Data API (Data API) を呼び出すことができます。Amazon VPC エンドポイントを使用することで、パブリック IP アドレスなしで Amazon VPC 内のアプリケーションと AWS ネットワーク内の Data API 間のトラフィックを維持できます。Amazon VPC エンドポイントは、公共のインターネット接続の制限に関連するコンプライアンスおよび規制要件を満たすのに役立ちます。例えば、Amazon VPC エンドポイントを使用する場合、Amazon EC2 インスタンスで実行されているアプリケーションと、それらを含む VPC 内の Data API 間のトラフィックを維持できます。

Amazon VPC エンドポイントを作成したら、アプリケーションでコードや設定を変更せずに、エンドポイントの使用をスタートできます。

Data API に Amazon VPC エンドポイントを作成するには

1. AWS Management Console にサインインして、Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
2. [エンドポイント] を選択し、[エンドポイントの作成] を選択します。
3. [エンドポイントの作成] ページの [サービスカテゴリ] で [AWS サービス] を選択します。[サービス名] で、[rds-data] を選択します。



4. [VPC] の場合は、VPC を選択してエンドポイントを作成します。

   Data API コールを行うアプリケーションを含む VPC を選択します。

5. [サブネット] で、アプリケーションを実行している AWS のサービスによって使用される各アベイラビリティーゾーン (AZ) のサブネットを選択します。

   Amazon VPC エンドポイントを作成するには、エンドポイントにアクセスできるプライベート IP アドレスの範囲を指定します。これを行うには、各アベイラビリティーゾーンのサブネットを選択します。これにより、VPC エンドポイントは各アベイラビリティーゾーンに固有のプラ

イベート IP アドレス範囲に制限され、各アベイラビリティーゾーンに Amazon VPC エンドポイントが作成されます。

6. [DNS 名を有効にする] で、[このエンドポイントで有効にする] を選択します。

プライベート DNS は、スタンダードの Data API DNS ホスト名 (`https://rds-data.region.amazonaws.com`) を、Amazon VPC エンドポイントに固有の DNS ホスト名に関連付けられたプライベート IP アドレスに解決します。その結果、Data API エンドポイント URL を更新するためのコードや設定を変更せずに、AWS CLI または AWS SDK を使用して Data API VPC エンドポイントにアクセスできます。

7. セキュリティグループで、Amazon VPC エンドポイントに関連付けるセキュリティグループを選択します。

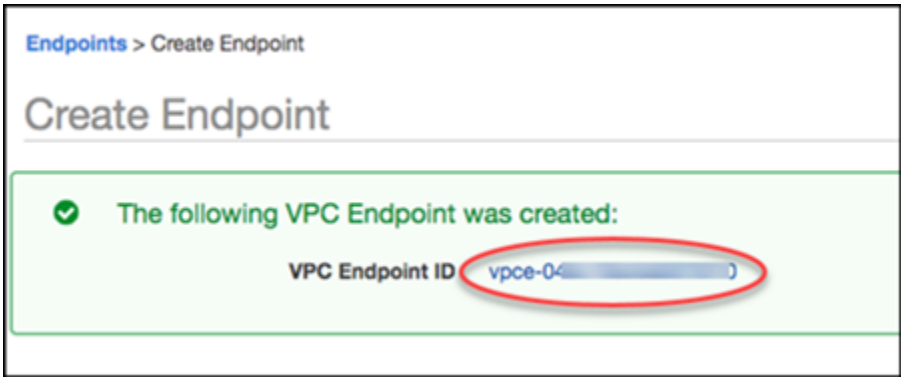
   アプリケーションを実行している AWS のサービスへのアクセスを許可するセキュリティグループを選択します。例えば、Amazon EC2 インスタンスでアプリケーションを実行している場合は、Amazon EC2 インスタンスへのアクセスを許可するセキュリティグループを選択します。セキュリティグループを使用すると、VPC 内のリソースから Amazon VPC エンドポイントへのトラフィックを制御できます。

8. [ポリシー] で [フルアクセス] を選択すると、Amazon VPC 内部の誰でもこのエンドポイントを介して Data API にアクセスできるようになります。または、[カスタム] を選択して、アクセスを制限するポリシーを指定します。

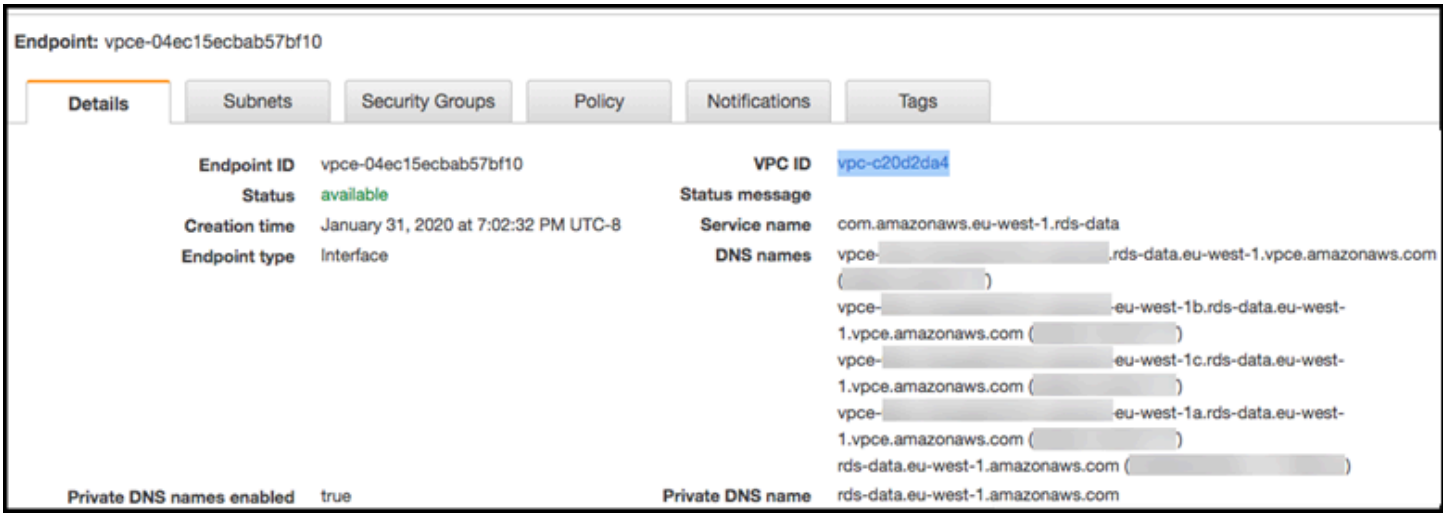
   [カスタム] を選択した場合は、ポリシー作成ツールでポリシーを入力します。

9. [エンドポイントの作成] を選択します。

エンドポイントを作成したら、AWS Management Console でリンクを選択して、エンドポイントの詳細を表示します。



エンドポイントの [Details (詳細)] タブには、Amazon VPC エンドポイントの作成中に生成された DNS ホスト名が表示されます。



スタンダードエンドポイント (`rds-data.`*`region`*`.amazonaws.com`) または VPC 固有のエンドポイントの 1 つを使用して、Amazon VPC 内の Data API を呼び出すことができます。スタンダード Data API エンドポイントは、自動的に Amazon VPC エンドポイントにルーティングします。このルーティングは、Amazon VPC エンドポイントの作成時にプライベート DNS ホスト名が有効になったために発生します。

Data API コールで Amazon VPC エンドポイントを使用すると、アプリケーションと Data API 間のすべてのトラフィックは、それらを含む Amazon VPC に残ります。Amazon VPC エンドポイントは、任意のタイプの Data API コールに使用できます。Data API を呼び出す方法については、「RDS Data API の呼び出し」を参照してください。

RDS Data API の呼び出し

Aurora DB クラスターで RDS Data API (Data API) を有効にした状態で、Data API または AWS CLI により、Aurora DB クラスターで SQL ステートメントを実行できます。Data API は、AWS SDK で

サポートされているプログラミング言語をサポートしています。詳細については、AWS で構築するツールを参照してください。

> **Note**
>
> 現在、Data API はユニバーサル固有識別子 (UUID) の配列をサポートしていません。

Data API では、以下のオペレーションを使用して、SQL ステートメントを実行することができます。

Data API オペレーション	AWS CLI コマンド	説明
ExecuteStatement	aws rds-data execute-statement	データベースで SQL ステートメントを実行します。
BatchExecuteStatement	aws rds-data batch-execute-statement	データの配列に対してバッチ SQL ステートメントを実行して、一括更新オペレーションおよび挿入オペレーションを行います。パラメータセットの配列を使用して、データ操作言語 (DML) ステートメントを実行できます。バッチ SQL ステートメントは、挿入ステートメントと更新ステートメントを個々に実行するよりもパフォーマンスは大幅に向上します。

どちらかの操作を使用して、個々の SQL ステートメントを実行したり、トランザクションを実行したりできます。トランザクションの場合、Data API は次のオペレーションを提供します。

Data API オペレーション	AWS CLI コマンド	説明
BeginTransaction	aws rds-data begin-transaction	SQL トランザクションをスタートします。
CommitTransaction	aws rds-data commit-transaction	SQL トランザクションを終了し、変更をコミットします。

Data API オペレーション	AWS CLI コマンド	説明
RollbackTransaction	aws rds-data rollback-transaction	トランザクションをロールバックします。

SQL ステートメントを実行し、トランザクションをサポートするためのオペレーションには、以下の共通の Data API パラメータおよび AWS CLI オプションがあります。他のパラメータやオプションをサポートするオペレーションもあります。

Data API オペレーションのパラメータ	AWS CLI コマンドオプション	必須	説明
resourceArn	--resource-arn	はい	Aurora DB クラスターの Amazon リソースネーム (ARN)。
secretArn	--secret-arn	はい	DB クラスターへのアクセスを有効にするシークレットの ARN の名前。

パラメータは、Data API の ExecuteStatement および BatchExecuteStatement への呼び出しと、AWS CLI の execute-statement コマンドおよび batch-execute-statement コマンドの実行時に使用できます。パラメータを使用するには、名前と値のペアを SqlParameter データ型で指定します。値は、Field データ型で指定します。次の表は、Data API 呼び出しで指定したデータ型に Java Database Connectivity (JDBC) データ型をマッピングしたものです。

JDBC データ型	Data API のデータ型
INTEGER, TINYINT, SMALLINT, BIGINT	LONG+ または STRING-
FLOAT, REAL, DOUBLE	DOUBLE
DECIMAL	STRING

JDBC データ型	Data API のデータ型
BOOLEAN, BIT	BOOLEAN
BLOB, BINARY, LONGVARBINARY, VARBINARY	BLOB
CLOB	STRING
その他の型 (日時に関する型も含む)	STRING

> **Note**
>
> データベースによって返される LONG 値の Data API 呼び出しで、STRING または LONG データ型を指定できます。非常に大きな数値では精度を失うといった、JavaScript での作業時に発生する可能性のある事態を避けるために、これを行うことをお勧めします。

DECIMAL や TIME などの特定のタイプでは、Data API が String 値を正しいタイプとしてデータベースに渡せるように、ヒントが必要です。ヒントを使用するには、typeHint データタイプの SqlParameter 値を含めます。typeHint に指定できる値は以下のとおりです。

- DATE - 対応する String パラメータ値は、DATE タイプのオブジェクトとしてデータベースに送信されます。受け入れられる形式は YYYY-MM-DD です。
- DECIMAL - 対応する String パラメータ値は、DECIMAL タイプのオブジェクトとしてデータベースに送信されます。
- JSON - 対応する String パラメータ値は、JSON タイプのオブジェクトとしてデータベースに送信されます。
- TIME - 対応する String パラメータ値は、TIME タイプのオブジェクトとしてデータベースに送信されます。受け入れられる形式は HH:MM:SS[.FFF] です。
- TIMESTAMP - 対応する String パラメータ値は、TIMESTAMP タイプのオブジェクトとしてデータベースに送信されます。受け入れられる形式は YYYY-MM-DD HH:MM:SS[.FFF] です。
- UUID - 対応する String パラメータ値は、UUID タイプのオブジェクトとしてデータベースに送信されます。

> **Note**
>
> Amazon Aurora PostgreSQL の場合、Data API は常に UTC タイムゾーンの Aurora PostgreSQL データ型 `TIMESTAMPTZ` を返します。

AWS CLI を使用した RDS Data API の呼び出し

RDS Data API (Data API) は、AWS CLI を使用して呼び出すことができます。

以下の例では、Data API で AWS CLI を使用しています。詳細については、AWS CLI reference for the Data API を参照してください。

それぞれの例で、DB クラスターの Amazon リソースネーム (ARN) を Aurora DB クラスターの ARN に置き換えます。また、シークレット ARN を Secrets Manager のシークレットの ARN に置き換え、DB クラスターへのアクセスを許可します。

> **Note**
>
> AWS CLI はレスポンスを JSON でフォーマットできます。

トピック

SQL トランザクションのスタート

SQL トランザクションをスタートするには、CLI の `aws rds-data begin-transaction` コマンドを使用します。コールによって、トランザクション識別子が返ります。

> **Important**
>
> 3 分以内にトランザクション ID を使用する呼び出しがないと、トランザクションはタイムアウトします。トランザクションがコミットされる前にタイムアウトした場合は、自動的にロールバックされます。
> トランザクション内のデータ定義言語 (DDL) ステートメントは、暗黙的なコミットを引き起こします。DDL ステートメントは、`execute-statement` コマンドに `--continue-after-timeout` オプションを指定して、ひとつずつ実行することをお勧めします。

共通オプションに加えて、`--database` オプションを指定します。オプションには、データベースの名前を指定します。

例えば、次の CLI コマンドでは、SQL トランザクションを起動します。

Linux、macOS、Unix の場合:

```
aws rds-data begin-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret"
```

Windows の場合:

```
aws rds-data begin-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret"
```

次は、レスポンスの例です。

```
{
    "transactionId": "ABC1234567890xyz"
}
```

SQL ステートメントの実行

SQL ステートメントを実行するには、CLI の `aws rds-data execute-statement` を使用します。

SQL ステートメントをトランザクションで実行するには、--transaction-id オプションとあわせてトランザクション識別子を指定します。トランザクションをスタートするには、CLI の aws rds-data begin-transaction コマンドを使用します。トランザクションを終了し、コミットするには、CLI の aws rds-data commit-transaction コマンドを使用します。

> **Important**
>
> --transaction-id オプションを指定しない場合、呼び出しによる変更は自動的にコミットされます。

共通オプションに加えて、次のオプションを指定します。

- --sql (必須) - DB クラスターで実行する SQL ステートメント。
- --transaction-id (オプション) - CLI の begin-transaction コマンドを使用してスタートされたトランザクションの識別子。SQL ステートメントを含めるトランザクションのトランザクション ID を指定します。
- --parameters (オプション) - SQL ステートメントのパラメータ。
- --include-result-metadata | --no-include-result-metadata (オプション) - 結果にメタデータを含むかどうかを示す値。デフォルト: --no-include-result-metadata。
- --database (オプション) - データベースの名前。

  --database オプションは、前のリクエストで --sql "use *database_name*;" を実行した後に SQL ステートメントを実行すると、機能しない場合があります。--sql "use *database_name*;" ステートメントを実行する代わりに --database オプションを使用することをお勧めします。
- --continue-after-timeout | --no-continue-after-timeout (オプション) - 呼び出しのタイムアウト後、ステートメントの実行を継続するかどうかを示す値。デフォルト: --no-continue-after-timeout。

  データ定義言語 (DDL) ステートメントでは、エラーやデータ構造の破損の可能性を回避するために、呼び出しがタイムアウトした後もステートメントを実行し続けることをお勧めします。
- --format-records-as "JSON"|"NONE" - 結果セットを JSON 文字列としてフォーマットするかどうかを指定するオプションの値です。デフォルト: "NONE"。JSON 結果セットの処理の詳細については、「JSON 形式でクエリ結果を処理する」を参照してください。

DB クラスターは呼び出しに対してレスポンスを返します。

> **Note**
>
> レスポンスサイズの上限は 1 MiB です。1 MiB を超えるレスポンスデータが返されると、その呼び出しは終了します。
> Aurora Serverless v1 では、1 秒あたりの最大リクエスト数は 1,000 です。サポートされている他のすべてのデータベースには、制限はありません。

例えば、次の CLI コマンドでは単一の SQL ステートメントを実行して、結果からメタデータを除外します (デフォルト)。

Linux、macOS、Unix の場合:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "select * from mytable"
```

Windows の場合:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "select * from mytable"
```

次は、レスポンスの例です。

```
{
    "numberOfRecordsUpdated": 0,
    "records": [
        [
            {
                "longValue": 1
            },
            {
                "stringValue": "ValueOne"
            }
```

```
        ],
        [
            {
                "longValue": 2
            },
            {
                "stringValue": "ValueTwo"
            }
        ],
        [
            {
                "longValue": 3
            },
            {
                "stringValue": "ValueThree"
            }
        ]
    ]
}
```

次の CLI コマンドでは、--transaction-id オプションを指定して、トランザクション内の単一の SQL ステートメントを実行します。

Linux、macOS、Unix の場合:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "update mytable set quantity=5 where id=201" --transaction-id "ABC1234567890xyz"
```

Windows の場合:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "update mytable set quantity=5 where id=201" --transaction-id "ABC1234567890xyz"
```

次は、レスポンスの例です。

```
{
```

```
    "numberOfRecordsUpdated": 1
}
```

次の CLI コマンドでは、パラメータを指定して単一の SQL ステートメントを実行します。

Linux、macOS、Unix の場合:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "insert into mytable values (:id, :val)" --parameters "[{\"name\": \"id\",
 \"value\": {\"longValue\": 1}},{\"name\": \"val\", \"value\": {\"stringValue\":
 \"value1\"}}]"
```

Windows の場合:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "insert into mytable values (:id, :val)" --parameters "[{\"name\": \"id\",
 \"value\": {\"longValue\": 1}},{\"name\": \"val\", \"value\": {\"stringValue\":
 \"value1\"}}]"
```

次は、レスポンスの例です。

```
{
    "numberOfRecordsUpdated": 1
}
```

次の CLI コマンドでは、データ定義言語 (DDL) の SQL ステートメントを実行します。DDL ステートメントによって、job 列の名前は role に変更されます。

> **Important**
>
> DDL ステートメントでは、呼び出しがタイムアウトした後もステートメントを実行し続けることをお勧めします。実行が終了する前に DDL ステートメントが終了すると、エラーが発生したり、データ構造が破損したりする恐れがあります。呼び出しがタイムアウトした後も

ステートメントの実行を続けるには、--continue-after-timeout オプションを指定します。

Linux、macOS、Unix の場合:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "alter table mytable change column job role varchar(100)" --continue-after-
timeout
```

Windows の場合:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "alter table mytable change column job role varchar(100)" --continue-after-
timeout
```

次は、レスポンスの例です。

```
{
    "generatedFields": [],
    "numberOfRecordsUpdated": 0
}
```

Note

generatedFields データは Aurora PostgreSQL によってサポートされていません。生成されたフィールドの値を取得するには、RETURNING 句を使用します。詳細については、PostgreSQL ドキュメントの「変更済みの行からデータを返す」を参照してください。

データ配列に対するバッチ SQL ステートメントの実行

データの配列に対してバッチ SQL ステートメントを実行するには、CLI の `aws rds-data batch-execute-statement` コマンドを使用します。このコマンドを使用して、一括インポートまたは一括更新オペレーションを実行できます。

SQL ステートメントをトランザクションで実行するには、`--transaction-id` オプションとあわせてトランザクション識別子を指定します。トランザクションをスタートするには、CLI の `aws rds-data begin-transaction` コマンドを使用します。トランザクションを終了し、コミットするには、CLI の `aws rds-data commit-transaction` コマンドを使用します。

> **Important**
>
> `--transaction-id` オプションを指定しない場合、呼び出しによる変更は自動的にコミットされます。

共通オプションに加えて、次のオプションを指定します。

- `--sql` (必須) - DB クラスターで実行する SQL ステートメント。

  > **Tip**
  >
  > MySQL 互換のステートメントでは、`--sql` パラメータの末尾にセミコロンを含めないでください。末尾のセミコロンは、構文エラーを引き起こす可能性があります。

- `--transaction-id` (オプション) - CLI の `begin-transaction` コマンドを使用してスタートされたトランザクションの識別子。SQL ステートメントを含めるトランザクションのトランザクション ID を指定します。
- `--parameter-set` (オプション) - バッチオペレーション用のパラメータセット。
- `--database` (オプション) - データベースの名前。

DB クラスターは、呼び出しに対してレスポンスを返します。

> **Note**
>
> パラメータセットの数に固定された上限はありません。ただし、データ API を介して送信される HTTP リクエストの最大サイズは 4 MiB です。リクエストがこの制限を超えると、デー

タ API はエラーを返し、リクエストを処理しません。この 4 MiB の制限には、リクエスト内の HTTP ヘッダーのサイズと JSON 表記が含まれます。したがって、含めることができるパラメータセットの数は、SQL ステートメントのサイズや各パラメータセットのサイズなどの要素の組み合わせによって異なります。
レスポンスサイズの上限は 1 MiB です。1 MiB を超えるレスポンスデータが返されると、その呼び出しは終了します。
Aurora Serverless v1 では、1 秒あたりの最大リクエスト数は 1,000 です。サポートされている他のすべてのデータベースには、制限はありません。

例えば、次の CLI コマンドは、パラメータセットを使用してデータの配列に対してバッチ SQL ステートメントを実行します。

Linux、macOS、Unix の場合:

```
aws rds-data batch-execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "insert into mytable values (:id, :val)" \
--parameter-sets "[[{\"name\": \"id\", \"value\": {\"longValue\": 1}},{\"name\":
 \"val\", \"value\": {\"stringValue\": \"ValueOne\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 2}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueTwo\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 3}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueThree\"}}]]"
```

Windows の場合:

```
aws rds-data batch-execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "insert into mytable values (:id, :val)" ^
--parameter-sets "[[{\"name\": \"id\", \"value\": {\"longValue\": 1}},{\"name\":
 \"val\", \"value\": {\"stringValue\": \"ValueOne\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 2}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueTwo\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 3}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueThree\"}}]]"
```

> **Note**
>
> `--parameter-sets` オプションに改行を含めないでください。

SQL トランザクションのコミット

CLI の `aws rds-data commit-transaction` コマンドを使用すると、`aws rds-data begin-transaction` でスタートした SQL トランザクションを終了し、変更をコミットすることができます。

共通オプションに加えて、次のオプションを指定します。

- `--transaction-id` (必須) - CLI の `begin-transaction` コマンドを使用してスタートされたトランザクションの識別子。終了し、コミットするトランザクションのトランザクション ID を指定します。

例えば、次の CLI コマンドでは、SQL トランザクションを終了し、変更をコミットします。

Linux、macOS、Unix の場合:

```
aws rds-data commit-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--transaction-id "ABC1234567890xyz"
```

Windows の場合:

```
aws rds-data commit-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--transaction-id "ABC1234567890xyz"
```

次は、レスポンスの例です。

```
{
    "transactionStatus": "Transaction Committed"
}
```

SQL トランザクションのロールバック

CLI の aws rds-data rollback-transaction コマンドを使用すると、aws rds-data begin-transaction でスタートした SQL トランザクションをロールバックすることができます。トランザクションをロールバックすると、変更はキャンセルされます。

> **Important**
>
> トランザクション ID の有効期限が切れている場合、トランザクションは自動的にロールバックされます。この場合、有効期限切れのトランザクション ID を指定する aws rds-data rollback-transaction コマンドによって、エラーが返ります。

共通オプションに加えて、次のオプションを指定します。

- --transaction-id (必須) - CLI の begin-transaction コマンドを使用してスタートされたトランザクションの識別子。ロールバックするトランザクションのトランザクション ID を指定します。

例えば、次の AWS CLI コマンドでは、SQL トランザクションをロールバックします。

Linux、macOS、Unix の場合:

```
aws rds-data rollback-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--transaction-id "ABC1234567890xyz"
```

Windows の場合:

```
aws rds-data rollback-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--transaction-id "ABC1234567890xyz"
```

次は、レスポンスの例です。

```
{
    "transactionStatus": "Rollback Complete"
```

```
    }
```

Python アプリケーションからの RDS Data API の呼び出し

Python アプリケーションから RDS Data API (Data API) を呼び出すことができます。

以下の例では、AWS SDK for Python (Boto) を使用します。Boto の詳細については、AWS SDK for Python (Boto 3) ドキュメントを参照してください。

それぞれの例で、DB クラスターの Amazon リソースネーム (ARN) を Aurora DB クラスターの ARN に置き換えます。また、シークレット ARN を Secrets Manager のシークレットの ARN に置き換え、DB クラスターへのアクセスを許可します。

トピック

- SQL クエリの実行
- DML SQL ステートメントの実行
- SQL トランザクションの実行

SQL クエリの実行

Python アプリケーションを使用して、SELECT ステートメントを実行し、結果を取得することができます。

以下の例では、SQL クエリを実行します。

```
import boto3

rdsData = boto3.client('rds-data')

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

response1 = rdsData.execute_statement(
            resourceArn = cluster_arn,
            secretArn = secret_arn,
            database = 'mydb',
            sql = 'select * from employees limit 3')

print (response1['records'])
[
    [
```

```
        {
            'longValue': 1
        },
        {
            'stringValue': 'ROSALEZ'
        },
        {
            'stringValue': 'ALEJANDRO'
        },
        {
            'stringValue': '2016-02-15 04:34:33.0'
        }
    ],
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'DOE'
        },
        {
            'stringValue': 'JANE'
        },
        {
            'stringValue': '2014-05-09 04:34:33.0'
        }
    ],
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'STILES'
        },
        {
            'stringValue': 'JOHN'
        },
        {
            'stringValue': '2017-09-20 04:34:33.0'
        }
    ]
]
```

DML SQL ステートメントの実行

データ操作言語 (DML) ステートメントを実行して、データベースのデータを挿入、更新、または削除することができます。また、DML ステートメントでパラメータを使用することもできます。

> **Important**
>
> `transactionID` パラメータが含まれていないため、呼び出しがトランザクションの一部ではない場合、呼び出しによる変更は自動的にコミットされます。

以下の例では、SQL の挿入ステートメントを実行して、パラメータを使用します。

```
import boto3

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

rdsData = boto3.client('rds-data')


param1 = {'name':'firstname', 'value':{'stringValue': 'JACKSON'}}
param2 = {'name':'lastname', 'value':{'stringValue': 'MATEO'}}
paramSet = [param1, param2]

response2 = rdsData.execute_statement(resourceArn=cluster_arn,
                                      secretArn=secret_arn,
                                      database='mydb',
                                      sql='insert into employees(first_name, last_name)
 VALUES(:firstname, :lastname)',
                                      parameters = paramSet)

print (response2["numberOfRecordsUpdated"])
```

SQL トランザクションの実行

SQL トランザクションのスタートから、1 つ以上の SQL ステートメントの実行、Python アプリケーションによる変更のコミットまで行うことができます。

> **Important**
>
> 3 分以内にトランザクション ID を使用する呼び出しがないと、トランザクションはタイムアウトします。トランザクションがコミットされる前にタイムアウトした場合は、自動的にロールバックされます。
> トランザクション ID を指定しない場合、呼び出しによる変更は自動的にコミットされます。

以下の例では、テーブルに行を挿入する SQL トランザクションを実行します。

```
import boto3

rdsData = boto3.client('rds-data')

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

tr = rdsData.begin_transaction(
     resourceArn = cluster_arn,
     secretArn = secret_arn,
     database = 'mydb')

response3 = rdsData.execute_statement(
     resourceArn = cluster_arn,
     secretArn = secret_arn,
     database = 'mydb',
     sql = 'insert into employees(first_name, last_name) values('XIULAN', 'WANG')',
     transactionId = tr['transactionId'])

cr = rdsData.commit_transaction(
     resourceArn = cluster_arn,
     secretArn = secret_arn,
     transactionId = tr['transactionId'])

cr['transactionStatus']
'Transaction Committed'

response3['numberOfRecordsUpdated']
1
```

> **Note**
>
> データ定義言語 (DDL) ステートメントを実行する場合は、呼び出しがタイムアウトした後もステートメントを実行し続けることをお勧めします。実行が終了する前に DDL ステートメントが終了すると、エラーが発生したり、データ構造が破損したりする恐れがあります。呼び出しがタイムアウトした後もステートメントの実行を続けるには、`continueAfterTimeout` パラメータを `true` に設定します。

Java アプリケーションからの RDS Data API の呼び出し

Java アプリケーションから RDS Data API (Data API) を呼び出すことができます。

以下の例では、AWS SDK for Java を使用します。詳細については、「AWS SDK for Java デベロッパーガイド」を参照してください。

それぞれの例で、DB クラスターの Amazon リソースネーム (ARN) を Aurora DB クラスターの ARN に置き換えます。また、シークレット ARN を Secrets Manager のシークレットの ARN に置き換え、DB クラスターへのアクセスを許可します。

トピック

- SQL クエリの実行
- SQL トランザクションの実行
- SQL のバッチオペレーションを実行する

SQL クエリの実行

Java アプリケーションを使用して、`SELECT` ステートメントを実行し、結果を取得することができます。

以下の例では、SQL クエリを実行します。

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.ExecuteStatementRequest;
import com.amazonaws.services.rdsdata.model.ExecuteStatementResult;
```

```
import com.amazonaws.services.rdsdata.model.Field;

import java.util.List;

public class FetchResultsExample {
  public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
  public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

  public static void main(String[] args) {
    AWSRDSData rdsData = AWSRDSDataClient.builder().build();

    ExecuteStatementRequest request = new ExecuteStatementRequest()
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withDatabase("mydb")
            .withSql("select * from mytable");

    ExecuteStatementResult result = rdsData.executeStatement(request);

    for (List<Field> fields: result.getRecords()) {
      String stringValue = fields.get(0).getStringValue();
      long numberValue = fields.get(1).getLongValue();

      System.out.println(String.format("Fetched row: string = %s, number = %d",
 stringValue, numberValue));
    }
  }
}
```

SQL トランザクションの実行

SQL トランザクションのスタートから、1 つ以上の SQL ステートメントの実行、Java アプリケーションによる変更のコミットまで行うことができます。

> **Important**
>
> 3 分以内にトランザクション ID を使用する呼び出しがないと、トランザクションはタイムアウトします。トランザクションがコミットされる前にタイムアウトした場合は、自動的にロールバックされます。

トランザクション ID を指定しない場合、呼び出しによる変更は自動的にコミットされます。

以下の例では、SQL トランザクションを実行します。

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.BeginTransactionRequest;
import com.amazonaws.services.rdsdata.model.BeginTransactionResult;
import com.amazonaws.services.rdsdata.model.CommitTransactionRequest;
import com.amazonaws.services.rdsdata.model.ExecuteStatementRequest;

public class TransactionExample {
  public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
  public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

  public static void main(String[] args) {
    AWSRDSData rdsData = AWSRDSDataClient.builder().build();

    BeginTransactionRequest beginTransactionRequest = new BeginTransactionRequest()
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withDatabase("mydb");
    BeginTransactionResult beginTransactionResult =
 rdsData.beginTransaction(beginTransactionRequest);
    String transactionId = beginTransactionResult.getTransactionId();

    ExecuteStatementRequest executeStatementRequest = new ExecuteStatementRequest()
            .withTransactionId(transactionId)
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withSql("INSERT INTO test_table VALUES ('hello world!')");
    rdsData.executeStatement(executeStatementRequest);

    CommitTransactionRequest commitTransactionRequest = new CommitTransactionRequest()
            .withTransactionId(transactionId)
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN);
```

```
    rdsData.commitTransaction(commitTransactionRequest);
  }
}
```

> **Note**
>
> データ定義言語 (DDL) ステートメントを実行する場合は、呼び出しがタイムアウトした後もステートメントを実行し続けることをお勧めします。実行が終了する前に DDL ステートメントが終了すると、エラーが発生したり、データ構造が破損したりする恐れがあります。呼び出しがタイムアウトした後もステートメントの実行を続けるには、continueAfterTimeout パラメータを true に設定します。

SQL のバッチオペレーションを実行する

Java アプリケーションを使用して、データの配列対して、一括挿入および一括更新オペレーションを実行できます。DML ステートメントは、パラメータセットの配列を使用して実行できます。

> **Important**
>
> トランザクション ID を指定しない場合、呼び出しによる変更は自動的にコミットされます。

以下の例では、バッチ挿入オペレーションを実行します。

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.BatchExecuteStatementRequest;
import com.amazonaws.services.rdsdata.model.Field;
import com.amazonaws.services.rdsdata.model.SqlParameter;

import java.util.Arrays;

public class BatchExecuteExample {
  public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
  public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";
```

```
  public static void main(String[] args) {
      AWSRDSData rdsData = AWSRDSDataClient.builder().build();

    BatchExecuteStatementRequest request = new BatchExecuteStatementRequest()
            .withDatabase("test")
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withSql("INSERT INTO test_table2 VALUES (:string, :number)")
            .withParameterSets(Arrays.asList(
                    Arrays.asList(
                            new SqlParameter().withName("string").withValue(new
 Field().withStringValue("Hello")),
                            new SqlParameter().withName("number").withValue(new
 Field().withLongValue(1L))
                    ),
                    Arrays.asList(
                            new SqlParameter().withName("string").withValue(new
 Field().withStringValue("World")),
                            new SqlParameter().withName("number").withValue(new
 Field().withLongValue(2L))
                    )
            ));

    rdsData.batchExecuteStatement(request);
  }
}
```

RDS Data API 用の Java クライアントライブラリの使用

RDS Data API (Data API) 用の Java クライアントライブラリをダウンロードして使用できます。この Java クライアントライブラリは、Data API を使用する代替手段を提供します。このライブラリを使用すると、クライアント側のクラスを Data API のリクエストとレスポンスにマッピングできます。このマッピングのサポートにより、`Date`、`Time`、`BigDecimal` などの特定の Java タイプとの統合が容易になります。

Data API 用の Java クライアントライブラリのダウンロード

Data API Java クライアントライブラリは、GitHub の次の場所にあるオープンソースです。

https://github.com/awslabs/rds-data-api-client-library-java

ソースファイルからライブラリを手動で構築できますが、ベストプラクティスは、Apache Maven 依存関係管理を使用してライブラリを使用する方法です。Maven POM ファイルに次の依存関係を追加します。

バージョン 2.x (AWS SDK 2.x 互換) では、以下を使用します。

```
<dependency>
   <groupId>software.amazon.rdsdata</groupId>
   <artifactId>rds-data-api-client-library-java</artifactId>
   <version>2.0.0</version>
</dependency>
```

バージョン 1.x (AWS SDK 1.x 互換) では、以下を使用します。

```
<dependency>
    <groupId>software.amazon.rdsdata</groupId>
    <artifactId>rds-data-api-client-library-java</artifactId>
    <version>1.0.8</version>
</dependency>
```

Java クライアントライブラリの例

以下に、Data API Java クライアントライブラリを使用する一般的な例をいくつか示します。これらの例では、accounts と accountId の 2 つの列を含むテーブル name があることを前提としています。また、次のデータ転送オブジェクト (DTO) も使用されています。

```
public class Account {
    int accountId;
    String name;
    // getters and setters omitted
}
```

クライアントライブラリを使用すると、DTO を入力パラメータとして渡すことができます。次の例は、顧客 DTO が入力パラメータセットにマップされる方法を示しています。

```
var account1 = new Account(1, "John");
var account2 = new Account(2, "Mary");
client.forSql("INSERT INTO accounts(accountId, name) VALUES(:accountId, :name)")
         .withParamSets(account1, account2)
```

```
        .execute();
```

場合によっては、単純な値を入力パラメータとして使用する方が簡単です。これには、次の構文を使用します。

```
client.forSql("INSERT INTO accounts(accountId, name) VALUES(:accountId, :name)")
        .withParameter("accountId", 3)
        .withParameter("name", "Zhang")
        .execute();
```

以下は、入力パラメータとして単純な値を使用するもう 1 つの例です。

```

client.forSql("INSERT INTO accounts(accountId, name) VALUES(?, ?)", 4, "Carlos")
        .execute();

```

クライアントライブラリは、結果が返されたときに DTO への自動マッピングを提供します。次の例は、結果が DTO にどのようにマップされるかを示しています。

```
List<Account> result = client.forSql("SELECT * FROM accounts")
          .execute()
          .mapToList(Account.class);

Account result = client.forSql("SELECT * FROM accounts WHERE account_id = 1")
          .execute()
          .mapToSingle(Account.class);
```

多くの場合、データベースの結果セットには 1 つの値しか含まれていません。このような結果の取得を簡素化するために、クライアントライブラリは次の API を提供しています。

```
int numberOfAccounts = client.forSql("SELECT COUNT(*) FROM accounts")
          .execute()
          .singleValue(Integer.class);
```

> **Note**
>
> mapToList 関数は、SQL 結果セットをユーザー定義のオブジェクトリストに変換します。Java クライアントライブラリの ExecuteStatement コールで .withFormatRecordsAs(RecordsFormatType.JSON) ステートメントの使用は、同じ

目的を果たすためサポートしていません。詳細については、「JSON 形式でクエリ結果を処理する」を参照してください。

JSON 形式でクエリ結果を処理する

ExecuteStatement オペレーションを呼び出す場合、クエリ結果を JSON 形式の文字列として返すように選択できます。これにより、プログラミング言語の JSON 解析機能を使用して結果セットを解釈し、再フォーマットできます。そうすることで、結果セットをループして各列の値を解釈するために、余分なコードを記述せずに済むようになります。

JSON 形式で結果セットをリクエストするには、オプションの formatRecordsAs パラメータに JSON の値を渡します。JSON 形式の結果セットは、ExecuteStatementResponse 構造の formattedRecords に返されます。

BatchExecuteStatement アクションは結果セットを返しません。したがって、JSON オプションはそのアクションには適用されません。

JSON ハッシュ構造のキーをカスタマイズするには、結果セットにカラムのエイリアスを定義します。SQL クエリの列リスト内で AS 句を指定することで定義できます。

JSON 機能を使用することで、結果セットが読みやすくなり、その内容を言語固有のフレームワークにマッピングしやすくなります。ASCII でエンコードされた結果セットのボリュームはデフォルトの表記よりも大きいため、大きな行または列値を返すクエリではアプリケーションで使用可能なメモリよりも多くのメモリを消費することになり、この場合はデフォルトの表記を選択する方が良いでしょう。

トピック

- JSON 形式でクエリ結果を取得する
- データ型のマッピング
- トラブルシューティング
- 例

JSON 形式でクエリ結果を取得する

結果セットを JSON 文字列として受け取るには、ExecuteStatement の呼び出しに .withFormatRecordsAs(RecordsFormatType.JSON) を含めます。戻り

値は、formattedRecords フィールドに JSON 文字列として戻ります。この場合、columnMetadata は null です。列ラベルは、各行を表すオブジェクトのキーです。これらの列名は、結果セットの行ごとに繰り返されます。列値は引用符で囲まれた文字列、数値、または true、false、null を表す特殊な値です。長さの制約、数値、文字列の正確な型などの列のメタデータは、JSON レスポンスでは保持されません。

.withFormatRecordsAs() 呼び出しを省略、または NONE のパラメータを指定した場合、結果セットは Records と columnMetadata のフィールドを使用してバイナリ形式で返されます。

データ型のマッピング

結果セットの SQL 値は、より小さい JSON 型のセットにマッピングされます。値は JSON で文字列、数値、および true、false、null などの特殊な定数で表現されます。これらの値は、プログラミング言語に適した強い型付けまたは弱い型付けを使用して、アプリケーション内の変数に変換できます。

JDBC データ型	JSON データ型
INTEGER, TINYINT, SMALLINT, BIGINT	デフォルトでは数値です。LongReturnType オプションが STRING に設定されている場合の文字列です。
FLOAT, REAL, DOUBLE	数
DECIMAL	デフォルトでは文字列です。DecimalReturnType オプションが DOUBLE_OR_LONG に設定されている場合の数値です。
STRING	文字列
BOOLEAN, BIT	ブール値
BLOB, BINARY, VARBINARY , LONGVARBINARY	base64 コンコードの文字列です。
CLOB	文字列
ARRAY	配列
NULL	null

JDBC データ型	JSON データ型
その他の型 (日時に関する型も含む)	文字列

トラブルシューティング

JSON レスポンスは 10 メガバイトに制限されています。レスポンスがこの制限よりも大きい場合、プログラムには BadRequestException エラーが返されます。この場合、以下のいずれかの方法でエラーを解決できます。

- 結果セットの行数を減らします。これを行うには、LIMIT 句を追加します。LIMIT と OFFSET の句を持つ複数のクエリを送信することで、大きな結果セットを複数の小さな結果セットに分割できます。

  アプリケーションロジックによって、結果セットにフィルタリングされた行が含まれている場合は、WHERE 句にさらに条件を追加することで、結果セットから該当の行を削除できます。
- 結果セットの列数を減らします。これを行うには、クエリの選択リストから項目を削除します。
- クエリで列エイリアスを使用することにより、列ラベルを短くします。各列名は、結果セット内の各行に対して JSON 文字列で繰り返されます。そのため、長い列名と多くの行を含むクエリ結果の場合、サイズ制限を超える可能性があります。特に、複雑な式には列のエイリアスを使用して、JSON 文字列で式全体が繰り返されないようにします。
- SQL では、列のエイリアスを使用して同じ名前の列を複数持つ結果セットを作成できますが、JSON ではキー名の重複は許可されません。JSON 形式で結果セットをリクエストし、複数の列が同じ名前の場合、RDS データ API はエラーを返します。そのため、すべての列のラベルに一意の名前があることを確認します。

例

以下の Java の例では、レスポンスを JSON 形式の文字列として ExecuteStatement を呼び出し、結果セットを解釈する方法を示しています。*databaseName*、*secretStoreArn*、*clusterArn* パラメータを適切な値に置き換えます。

以下の Java の例では、結果セットに 10 進数の値を返すクエリを示しています。assertThat 呼び出しは、JSON 結果セットのルールに基づいて、レスポンスのフィールドが、期待されるプロパティを持っているかをテストします。

この例では、以下のスキーマとサンプルデータで動作します。

```
create table test_simplified_json (a float);
insert into test_simplified_json values(10.0);
```

```
public void JSON_result_set_demo() {
    var sql = "select * from test_simplified_json";
    var request = new ExecuteStatementRequest()
      .withDatabase(databaseName)
      .withSecretArn(secretStoreArn)
      .withResourceArn(clusterArn)
      .withSql(sql)
      .withFormatRecordsAs(RecordsFormatType.JSON);
    var result = rdsdataClient.executeStatement(request);
}
```

前のプログラムの formattedRecords フィールドの値は以下のとおりです。

```
[{"a":10.0}]
```

JSON 結果セットが存在するため、レスポンスの Records と ColumnMetadata フィールドは両方とも null になります。

以下の Java の例では、結果セットに整数の値を返すクエリを示しています。この例では getFormattedRecords を呼び出します。JSON 形式の文字列のみを返し、他のレスポンスフィールドが空白または null の場合は無視します。この例では、結果をレコードのリストを表す構造体に逆シリアル化しています。各レコードには、結果セットの列のエイリアスに対応する名前を持つフィールドがあります。この方法では、結果セットを解析するコードを単純化します。アプリケーションでは、結果セットの行と列をループ処理して、各値を適切な型に変換する必要がありません。

この例では、以下のスキーマとサンプルデータで動作します。

```
create table test_simplified_json (a int);
insert into test_simplified_json values(17);
```

```
public void JSON_deserialization_demo() {
    var sql = "select * from test_simplified_json";
    var request = new ExecuteStatementRequest()
      .withDatabase(databaseName)
      .withSecretArn(secretStoreArn)
      .withResourceArn(clusterArn)
      .withSql(sql)
```

```
      .withFormatRecordsAs(RecordsFormatType.JSON);
    var result = rdsdataClient.executeStatement(request)
      .getFormattedRecords();

/* Turn the result set into a Java object, a list of records.
   Each record has a field 'a' corresponding to the column
   labelled 'a' in the result set. */
    private static class Record { public int a; }
    var recordsList = new ObjectMapper().readValue(
        response, new TypeReference<List<Record>>() {
        });
}
```

前のプログラムの formattedRecords フィールドの値は以下のとおりです。

```
[{"a":17}]
```

結果行 0 の a 列を取得するには、アプリケーションは recordsList.get(0).a を参照します。

一方で、以下の Java の例は、JSON 形式を使用しない場合に、結果セットを保持するデータ構造を構築するために必要なコードの種類を示しています。この場合、結果セットの各行には 1 人のユーザーに関する情報を含むフィールドが含まれています。結果セットを表すデータ構造を構築するには、行をループする必要があります。各行について、コードによって各フィールドの値を取得して、適切な型変換を実行し、その行を表すオブジェクト内の対応するフィールドに結果を代入します。次に、各ユーザーを表すオブジェクトを、結果セット全体を表すデータ構造に追加します。クエリの結果セット内でフィールドの並べ替え、追加、削除などで変更があった場合、アプリケーションコードも変更する必要があります。

```
/* Verbose result-parsing code that doesn't use the JSON result set format */
for (var row: response.getRecords()) {
    var user = User.builder()
      .userId(row.get(0).getLongValue())
      .firstName(row.get(1).getStringValue())
      .lastName(row.get(2).getStringValue())
      .dob(Instant.parse(row.get(3).getStringValue()))
      .build();
    result.add(user);
  }
```

以下のサンプル値は、列数、列のエイリアス、列のデータ型が異なる結果セットの formattedRecords フィールドの値を示しています。

結果セットに複数の行が含まれている場合、各行は配列要素であるオブジェクトとして表されます。結果セット内の各列は、オブジェクトのキーになります。このキーは、結果セットの行ごとに繰り返されます。このため、多くの行と列で構成される結果セットでは、レスポンス全体の長さ制限を超えないように、短い列のエイリアスを定義する必要があります。

この例では、以下のスキーマとサンプルデータで動作します。

```
create table sample_names (id int, name varchar(128));
insert into sample_names values (0, "Jane"), (1, "Mohan"), (2, "Maria"), (3, "Bruce"),
 (4, "Jasmine");
```

```
[{"id":0,"name":"Jane"},{"id":1,"name":"Mohan"},
{"id":2,"name":"Maria"},{"id":3,"name":"Bruce"},{"id":4,"name":"Jasmine"}]
```

結果セットの列が式として定義されている場合、式のテキストが JSON キーになります。したがって、通常、クエリの選択リスト内の式ごとに説明的な列のエイリアスを定義すると便利です。例えば、以下のクエリでは、関数呼び出しや算術演算などの式が選択リストに含まれています。

```
select count(*), max(id), 4+7 from sample_names;
```

これらの式は JSON 結果セットにキーとして渡されます。

```
[{"count(*)":5,"max(id)":4,"4+7":11}]
```

AS 列に説明的なラベルを追加することで、JSON 結果セットでキーが簡単に解釈できます。

```
select count(*) as rows, max(id) as largest_id, 4+7 as addition_result from
 sample_names;
```

変更された SQL クエリでは、AS 句で定義した列のラベルがキー名として使用されます。

```
[{"rows":5,"largest_id":4,"addition_result":11}]
```

JSON 文字列の各キーバリューペアの値は、引用符で囲まれた文字列にできます。文字列には Unicode 文字が含まれている場合があります。文字列にエスケープシーケンスが含まれている場合、または " や \ 文字の場合、それらの文字の前にバックスラッシュ (エスケープ) 文字が付きます。以下に、可能性のある JSON 文字列の例を示します。例えば、string_with_escape_sequences

の結果には、特殊文字バックスペース、改行、キャリッジリターン、タブ、フォームフィード、\ などがあります。

```
[{"quoted_string":"hello"}]
[{"unicode_string":"####"}]
[{"string_with_escape_sequences":"\b \n \r \t \f \\ '"}]
```

JSON 文字列の各キーバリューペアの値は、数値で表すことができます。数値は、整数、浮動小数点数、負の値、指数表記などで表される値になる場合があります。以下に、可能性のある JSON 文字列の例を示します。

```
[{"integer_value":17}]
[{"float_value":10.0}]
[{"negative_value":-9223372036854775808,"positive_value":92233720368547758O7}]
[{"very_small_floating_point_value":4.9E-324,"very_large_floating_point_value":1.79769313486231
```

ブール値と NULL 値は、引用符で囲まれていない特殊キーワード `true`、`false`、`null` で表されます。以下に、可能性のある JSON 文字列の例を示します。

```
[{"boolean_value_1":true,"boolean_value_2":false}]
[{"unknown_value":null}]
```

BLOB タイプの値を選択した場合、結果は base64 でエンコードされた値として JSON 文字列で表されます。値を元の表現に戻すには、アプリケーションの言語で適切なデコード関数を使用できます。例えば、Java では、`Base64.getDecoder().decode()` 関数を呼び出します。以下の出力例は、`hello world` の BLOB 値を選択し、結果セットを JSON 文字列として返した結果を示しています。

```
[{"blob_column":"aGVsbG8gd29ybGQ="}]
```

以下の Python の例は、Python の `execute_statement` 関数の呼び出し結果から、値にアクセスする方法を示しています。結果セットは、`response['formattedRecords']` フィールド内の文字列の値です。このコードでは、`json.loads` 関数を呼び出すことで、JSON 文字列をデータ構造に変換します。また、結果セットの各行はデータ構造内のリスト要素であり、各行内では、結果セットの各フィールドを名前で参照できます。

```
import json

result = json.loads(response['formattedRecords'])
```

```
print (result[0]["id"])
```

以下の JavaScript の例は、JavaScript の `executeStatement` 関数の呼び出し結果から、値にアクセスする方法を示しています。結果セットは、`response.formattedRecords` フィールド内の文字列の値です。このコードでは、`JSON.parse` 関数を呼び出すことで、JSON 文字列をデータ構造に変換します。また、結果セットの各行はデータ構造内の配列要素であり、各行内では、結果セットの各フィールドを名前で参照できます。

```
<script>
    const result = JSON.parse(response.formattedRecords);
    document.getElementById("display").innerHTML = result[0].id;
</script>
```

RDS Data API の問題のトラブルシューティング

RDS Data API (Data API) に関する問題のトラブルシューティングには、「一般的なエラーメッセージ」というタイトルの以下のセクションを使用します。

トピック

- トランザクション <transaction_ID> が見つかりません
- クエリのパケットが大きすぎる
- データベース応答がサイズ制限を超えている
- HttpEndpoint がクラスター <cluster_ID> に対して有効になっていない

トランザクション <transaction_ID> が見つかりません

この場合、Data API 呼び出しで指定されたトランザクション ID は見つかりませんでした。この問題の原因は、エラーメッセージに追加されますが、次のいずれかであることが考えられます。

- トランザクションの有効期限が切れている可能性があります。

  各トランザクションの呼び出しが、前のトランザクション呼び出しから 3 分以内に実行されることを確認します。

  指定されたトランザクション ID が、BeginTransaction コールによって作成されていない可能性もあります。呼び出しに有効なトランザクション ID があることを確認します。
- 前回の呼び出しで、トランザクションが終了しました。

トランザクションは、お客様の CommitTransaction または RollbackTransaction コールによってすでに終了しています。

- 前回の呼び出しのエラーにより、トランザクションが中断されました。

  以前の呼び出しで例外が発生したかどうかを確認します。

トランザクションの実行については、「RDS Data API の呼び出し」を参照してください。

クエリのパケットが大きすぎる

この場合、1 行に対して返った結果セットが大きすぎます。Data API のサイズ制限は、データベースから返る結果セットの 1 行あたり 64 KB です。

この問題を解決するには、結果セットの各行が 64 KB 以下であることを確認します。

データベース応答がサイズ制限を超えている

この場合、データベースから返る結果セットのサイズが大きすぎます。Data API の制限は、データベースから返る結果セットの 1 MiB です。

この問題を解決するには、Data API への呼び出しで返るデータが 1 MiB 以下になるようにします。1 MiB を超えるデータを返す必要がある場合は、クエリでLIMIT句を使用して、ExecuteStatement呼び出しを複数回行います。

LIMIT 句の詳細については、MySQL ドキュメントの「SELECT 構文」を参照してください。

HttpEndpoint がクラスター <cluster_ID> に対して有効になっていない

この問題の原因として考えられるものとして、以下の項目を確認してください。

- Aurora DB クラスターは Data API をサポートしていません。例えば、Aurora MySQL の場合、Data API は Aurora Serverless v1 でのみ使用できます。RDS Data API がサポートする DB クラスターのタイプについては、「the section called “リージョンとバージョンの可用性”」を参照してください。
- Data API が Aurora DB クラスターに対して有効になっていません。Aurora DB クラスターで Data API を使用するには、DB クラスターに対して Data API が有効になっている必要があります。Data API を有効にする方法については、「RDS Data API の有効化」を参照してください。
- Data API が有効になった後、DB クラスターの名前が変更されました。この場合は、クラスターのデータ API を一度オフにしてから、再度有効にします。

- 指定された ARN がクラスターの ARN と正確に一致しません。別のソースから返された ARN、またはプログラムロジックによって構築された ARN が、クラスターの ARN と正確に一致していることを確認します。例えば、使用する ARN は、アルファベットの大文字と小文字がすべて一致しているかを確認します。

AWS CloudTrail での RDS Data API コールのログ記録

RDS Data API (Data API) は AWS CloudTrail と統合されています。これは、Data API のユーザー、ロール、または AWS のサービスによって実行されたアクションを記録するサービスです。CloudTrail は、Amazon RDS コンソールからの呼び出しとデータ API オペレーションへのコード呼び出しを含む、すべての データ API の API コールをイベントとしてキャプチャします。証跡を作成する場合は、Data API のイベントなど、Amazon S3 バケットへの CloudTrail イベントの継続的配信を有効にすることができます。CloudTrail によって収集されたデータを使用して、多くの情報を判断できます。この情報には、Data API に対するリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細が含まれます。

CloudTrail の詳細については、AWS CloudTrail ユーザーガイドを参照してください。

CloudTrail での Data API 情報の使用

AWS アカウントを作成すると、そのアカウントに対して CloudTrail が有効になります。Data API でサポートされているアクティビティ (管理イベント) が発生すると、そのアクティビティは [イベント履歴] で AWS のその他サービスのイベントと共に CloudTrail イベントに記録されます。最近の管理イベントは、AWS アカウントで表示、検索、ダウンロードできます。詳細については、「AWS CloudTrail ユーザーガイド」の「CloudTrail イベント履歴の使用」を参照してください。

Data API のイベントなど、AWS アカウントのイベントを継続的に記録する場合は、証跡を作成します。証跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成すると、すべての AWS リージョンに証跡が適用されます。追跡では、AWS パーティション内のすべての AWS リージョンからのイベントをログに記録し、指定した Simple Storage Service (Amazon S3)バケットにログファイルを配信します。さらに、CloudTrail·ログで収集したイベントデータをより詳細に分析し、それに基づく対応するためにその他の AWS サービスを設定できます。詳細については、『AWS CloudTrail ユーザーガイド:』の以下のトピックを参照してください。

- 証跡作成の概要
- CloudTrail がサポートされているサービスおよび統合

- CloudTrail の Amazon SNS 通知の設定
- CloudTrail ログファイルを複数のリージョンから受け取ると複数のアカウントから CloudTrail ログファイルを受け取る

すべての Data API オペレーションは、CloudTrail によってログに記録されます。また、Amazon RDS データサービス API リファレンスに記載されています。例えば、`BatchExecuteStatement`, `BeginTransaction` オペレーション、`CommitTransaction` オペレーション、および `ExecuteStatement` オペレーションへのコールに伴って、CloudTrail ログファイルにエントリが生成されます。

各イベントまたはログエントリには、リクエストの生成者に関する情報が含まれます。ID 情報は次の判断に役立ちます。

- リクエストが、ルートとユーザー認証情報のどちらを使用して送信されたか。
- リクエストが、ロールとフェデレーションユーザーの一時的なセキュリティ認証情報のどちらを使用して送信されたか。
- リクエストが、別の AWS サービスによって送信されたかどうか。

詳細については、「CloudTrail userIdentity 要素」を参照してください。

AWS CloudTrail の証跡からの Data API イベントの包含と除外

多くの Data API ユーザーは、Data API オペレーションの記録のために、AWS CloudTrail の証跡内のイベントを使用しています。イベントデータは、Data API へのリクエストでデータベース名、スキーマ名、または SQL ステートメントを公開しません。ただし、特定の時点でどのユーザーが特定の DB クラスターに対して特定のコールタイプを実行したかを把握すると、異常なアクセスパターンの検出に役立ちます。

AWS CloudTrail の証跡からの Data API イベントの包含

Aurora PostgreSQL Serverless v2 およびプロビジョニングされたデータベースの場合、次の Data API オペレーションはデータイベント として AWS CloudTrail に記録されます。データイベントは、CloudTrail がデフォルトではログ記録しない大量のデータプレーン API オペレーションです。追加の変更がイベントデータに適用されます。CloudTrail の料金の詳細については、「AWS CloudTrail の料金」を参照してください。

- BatchExecuteStatement

- BeginTransaction
- CommitTransaction
- ExecuteStatement
- RollbackTransaction

CloudTrail コンソール、AWS CLI、または CloudTrail API オペレーションを使用して、これらの Data API オペレーションをログに記録できます。CloudTrail コンソールで、データイベントタイプの [RDS Data API - DB クラスター] を選択します。詳細については、「AWS CloudTrail ユーザーガイド」の「AWS Management Console によるデータイベントのログ記録」を参照してください。

AWS CLI を使用して `aws cloudtrail put-event-selectors` コマンドを実行し、証跡のこれらの Data API オペレーションをログに記録します。DB クラスターのすべての Data API イベントを ログに記録するには、リソースタイプに `AWS::RDS::DBCluster` を指定します。次の例では、DB クラスター上のすべての Data API イベントをログに記録します。詳細については、「AWS CloudTrail ユーザーガイド」の「AWS Command Line Interface によるデータイベントのログ記録」を参照してください。

```
aws cloudtrail put-event-selectors --trail-name trail_name --advanced-event-selectors \
'{
   "Name": "RDS Data API Selector",
   "FieldSelectors": [
      {
         "Field": "eventCategory",
         "Equals": [
            "Data"
         ]
      },
      {
         "Field": "resources.type",
         "Equals": [
            "AWS::RDS::DBCluster"
         ]
      }
   ]
}'
```

高度なイベントセレクタを設定して、`readOnly`、`eventName`, および `resources.ARN` フィールドでさらにフィルタリングできます。これらのフィールドの詳細については、「AdvancedFieldSelector」を参照してください。

AWS CloudTrail の証跡からの Data API イベントの除外 (Aurora Serverless v1 のみ)

Aurora Serverless v1 の場合、Data APIイベントは管理イベントです。デフォルトでは、すべての Data API イベントが AWS CloudTrail の証跡に含まれます。ただし、Data API では多数のイベントが生成されるため、これらのイベントを CloudTrail の証跡から除外したい場合があります。[Amazon RDS のデータ API イベントを除外] 設定は、証跡からすべての Data API イベントを除外します。特定の Data API イベントを除外することはできません。

証跡から Data API イベントを除外するには、次の手順を実行します。

- CloudTrail コンソールで、証跡を作成するまたは証跡を更新する際に、[Amazon RDS Data API イベントを除外する] 設定をオンにします。
- CloudTrail API では、PutEventSelectors オペレーションを実行します。アドバンストイベントセレクタを使用している場合は、 [eventSource] フィールドを`rdsdata.amazonaws.com` と等しくないと設定すると、Data API イベントを除外できます。ベーシックイベントセレクタを使用している場合は、`ExcludeManagementEventSources` 属性の値を `rdsdata.amazonaws.com` に設定すると、Data API イベントを除外できます。詳細については、「AWS CloudTrail ユーザーガイド」の「AWS Command Line Interface によるイベントのログ記録」を参照してください。

> **Warning**
>
> CloudTrail ログから Data API イベントを除外すると、Data API アクションが不確定になることがあります。このオペレーションを実行するために必要な `cloudtrail:PutEventSelectors` アクセス許可をプリンシパルに与えるときは注意してください。

この除外は、コンソール設定または証跡のイベントセレクタを変更することで、いつでも無効化できます。その後、証跡が Data API イベントの記録をスタートします。ただし、除外が有効であった期間に発生した Data API イベントはリカバリされません。

コンソールまたは API を使用して Data API イベントを除外すると、それにより実行される CloudTrail の API オペレーション `PutEventSelectors` も CloudTrail ログに記録されます。Data API イベントが CloudTrail ログに表示されない場合は、`ExcludeManagementEventSources` 属性が `rdsdata.amazonaws.com` に設定されている `PutEventSelectors` イベントを探します。

詳細については、AWS CloudTrail ユーザーガイドの「証跡の管理イベントのログ記録」を参照してください。

Data API ログファイルのエントリについて

「トレイル」は、指定した Simple Storage Service (Amazon S3) バケットにイベントをログファイルとして配信するように設定できます。CloudTrail のログファイルは、単一か複数のログエントリを含みます。イベント は、任意の出典からの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストパラメータなどに関する情報が含まれます。CloudTrail ログファイルは、パブリック API 呼び出しの順序付けられたスタックトレースではないため、特定の順序では表示されません。

Aurora PostgreSQL Serverless v2 とプロビジョンド

次の例は、Aurora PostgreSQL Serverless v2 およびプロビジョニングされたデータベースの `ExecuteStatement` オペレーションを示す CloudTrail ログエントリを示しています。これらのデータベースでは、すべての Data API イベントは、イベントソースが rdsdataapi.amazonaws.com で、イベントタイプが Rds Data Service であるデータイベントです。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T00:49:34Z",
    "eventSource": "rdsdataapi.amazonaws.com",
    "eventName": "ExecuteStatement",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.16.102 Python/3.7.2 Windows/10 botocore/1.12.92",
    "requestParameters": {
        "continueAfterTimeout": false,
        "database": "**********",
        "includeResultMetadata": false,
        "parameters": [],
        "resourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-database-1",
        "schema": "**********",
        "secretArn": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:dataapisecret-ABC123",
```

```
        "sql": "**********"
    },
    "responseElements": null,
    "requestID": "6ba9a36e-b3aa-4ca8-9a2e-15a9eada988e",
    "eventID": "a2c7a357-ee8e-4755-a0d0-aed11ed4253a",
    "eventType": "Rds Data Service",
    "recipientAccountId": "123456789012"
}
```

Aurora Serverless v1

次の例は、Aurora Serverless v1 の前述の CloudTrail ログエントリの例がどのように表示されるかを示しています。Aurora Serverless v1 の場合、すべてのイベントは管理イベントで、イベントソースは rdsdata.amazonaws.com、イベントタイプは AwsApiCall です。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T00:49:34Z",
    "eventSource": "rdsdata.amazonaws.com",
    "eventName": "ExecuteStatement",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.16.102 Python/3.7.2 Windows/10 botocore/1.12.92",
    "requestParameters": {
        "continueAfterTimeout": false,
        "database": "**********",
        "includeResultMetadata": false,
        "parameters": [],
        "resourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-database-1",
        "schema": "**********",
        "secretArn": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:dataapisecret-ABC123",
        "sql": "**********"
    },
```

```
    "responseElements": null,
    "requestID": "6ba9a36e-b3aa-4ca8-9a2e-15a9eada988e",
    "eventID": "a2c7a357-ee8e-4755-a0d0-aed11ed4253a",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

クエリエディタの使用

クエリエディタを使用して、RDS コンソールで SQL クエリを実行できます。DB クラスターでは、データ操作やデータ定義 SQL ステートメントを実行できます。実行できる SQL には、Data API の制限が適用されます。詳細については、「the section called “制限事項”」を参照してください。

クエリエディタは、RDS Data API (Data API) が有効になっている Aurora DB クラスターを必要とします。Data API をサポートする DB クラスターとその有効化方法については、「RDS Data API の使用」を参照してください。

クエリエディタの可用性

クエリエディタは、Data API をサポートする特定の Aurora MySQL と Aurora PostgreSQL エンジンバージョンを使用する Aurora DB クラスターで、Data API が利用可能な AWS リージョン で使用できます。詳細については、「RDS Data API でサポートされているリージョンと Aurora DB エンジン」を参照してください。

クエリエディタへのアクセスの承認

ユーザーは、クエリエディタでクエリを実行するための承認を受ける必要があります。ユーザーがクエリエディタでクエリを実行することを承認するには、`AmazonRDSDataFullAccess` ポリシー (事前定義済みの AWS Identity and Access Management (IAM) ポリシー) をユーザーに追加します。

> **Note**
>
> ユーザーを作成するときには、マスターユーザー名とパスワードなど、データベースユーザーに対して行ったときと同じユーザー名とパスワードを使用してください。詳細については、AWS Identity and Access Managementユーザーガイドの「AWS アカウント での IAM ユーザーの作成」を参照してください。

クエリエディタにアクセス権を付与する IAM ポリシーを作成することもできます。作成したポリシーは、クエリエディタにアクセスする必要がある各ユーザーに追加します。

次のポリシーでは、クエリエディタにアクセスするための必要最低限のアクセス許可をユーザーに付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QueryEditor0",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutResourcePolicy",
                "secretsmanager:PutSecretValue",
                "secretsmanager:DeleteSecret",
                "secretsmanager:DescribeSecret",
                "secretsmanager:TagResource"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:rds-db-credentials/*"
        },
        {
            "Sid": "QueryEditor1",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword",
                "tag:GetResources",
                "secretsmanager:CreateSecret",
                "secretsmanager:ListSecrets",
                "dbqms:CreateFavoriteQuery",
                "dbqms:DescribeFavoriteQueries",
                "dbqms:UpdateFavoriteQuery",
                "dbqms:DeleteFavoriteQueries",
                "dbqms:GetQueryString",
                "dbqms:CreateQueryHistory",
                "dbqms:UpdateQueryHistory",
                "dbqms:DeleteQueryHistory",
                "dbqms:DescribeQueryHistory",
                "rds-data:BatchExecuteStatement",
                "rds-data:BeginTransaction",
                "rds-data:CommitTransaction",
                "rds-data:ExecuteStatement",
                "rds-data:RollbackTransaction"
            ],
            "Resource": "*"
        }
    ]
}
```

IAM ポリシーの作成については、AWS Identity and Access ManagementIAM ユーザーガイドの「IAM ポリシーの作成」を参照してください。

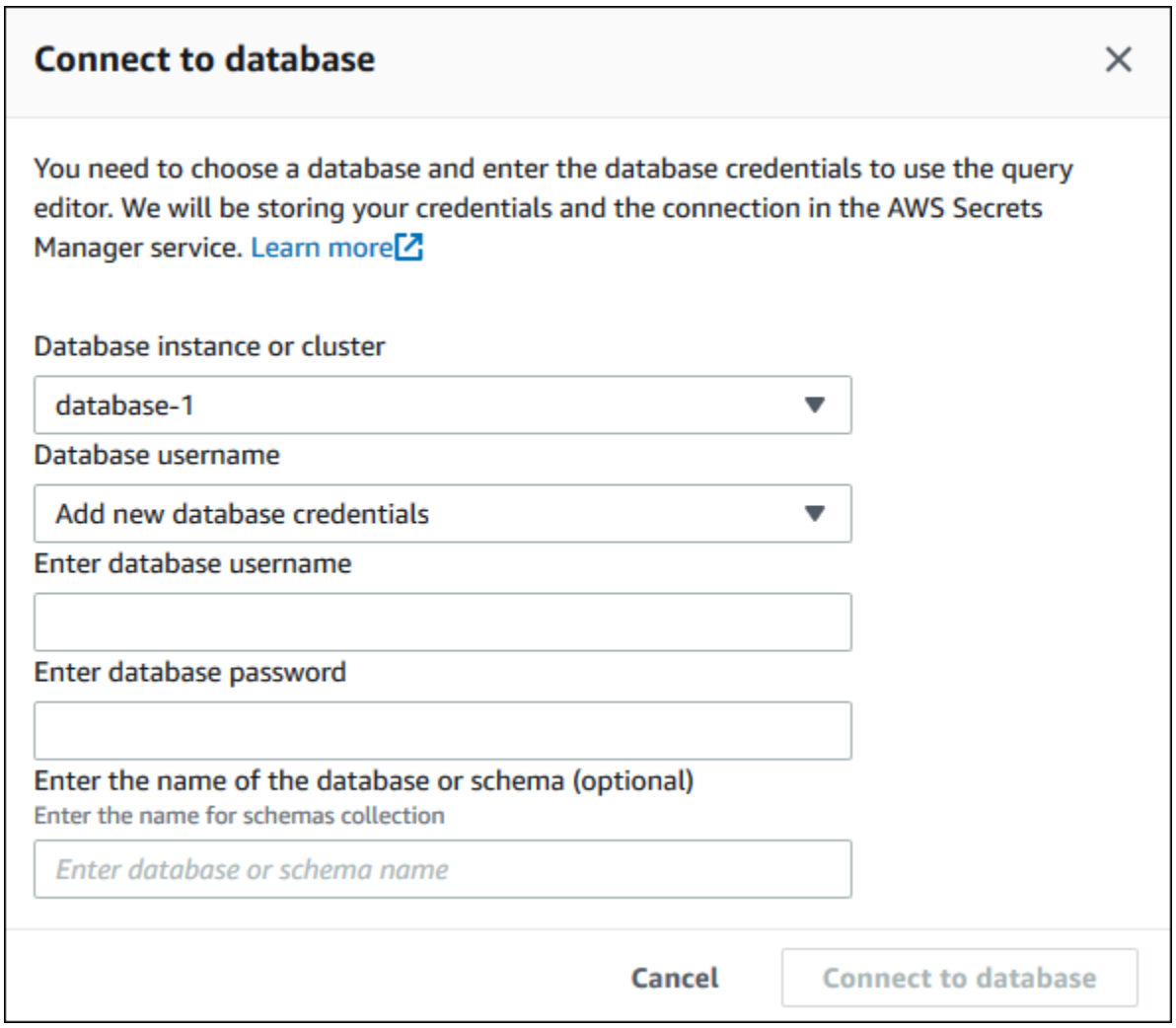
IAM ポリシーをユーザーに追加する方法については、AWS Identity and Access Management ユーザーガイドの「IAM ID のアクセス許可の追加および削除」を参照してください。

クエリエディタでのクエリの実行

クエリエディタで Aurora DB クラスターに対して SQL ステートメントを実行できます。実行できる SQL には、Data API の制限が適用されます。詳細については、「the section called “制限事項”」を参照してください。

クエリエディタでクエリを実行するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. AWS Management Console の右上で、クエリを実行する対象の Aurora DB クラスターを作成した AWS リージョン を選択します。
3. ナビゲーションペインで、データベースを選択します。
4. SQL クエリを実行する Aurora DB クラスターを選択します。
5. [アクション]、[クエリ] の順に選択します。まだデータベースに接続していない場合は、[Connect to database (データベースに接続)] ページが開きます。



6. 次の情報を入力します。

   a. [データベースインスタンスまたはクラスター] で、SQL クエリを実行する Aurora DB クラスターを選択します。

   b. [データベースユーザーネーム] で、接続するデータベースユーザーのユーザーネームを選択するか、[Add new database credentials (新しいデータベース認証情報の追加)] を選択します。[Add new database credentials (新しいデータベース認証情報の追加)] を選択した場合は、[Enter database username (データベースユーザーネームを入力)] に新しいデータベース認証情報のユーザーネームを入力します。

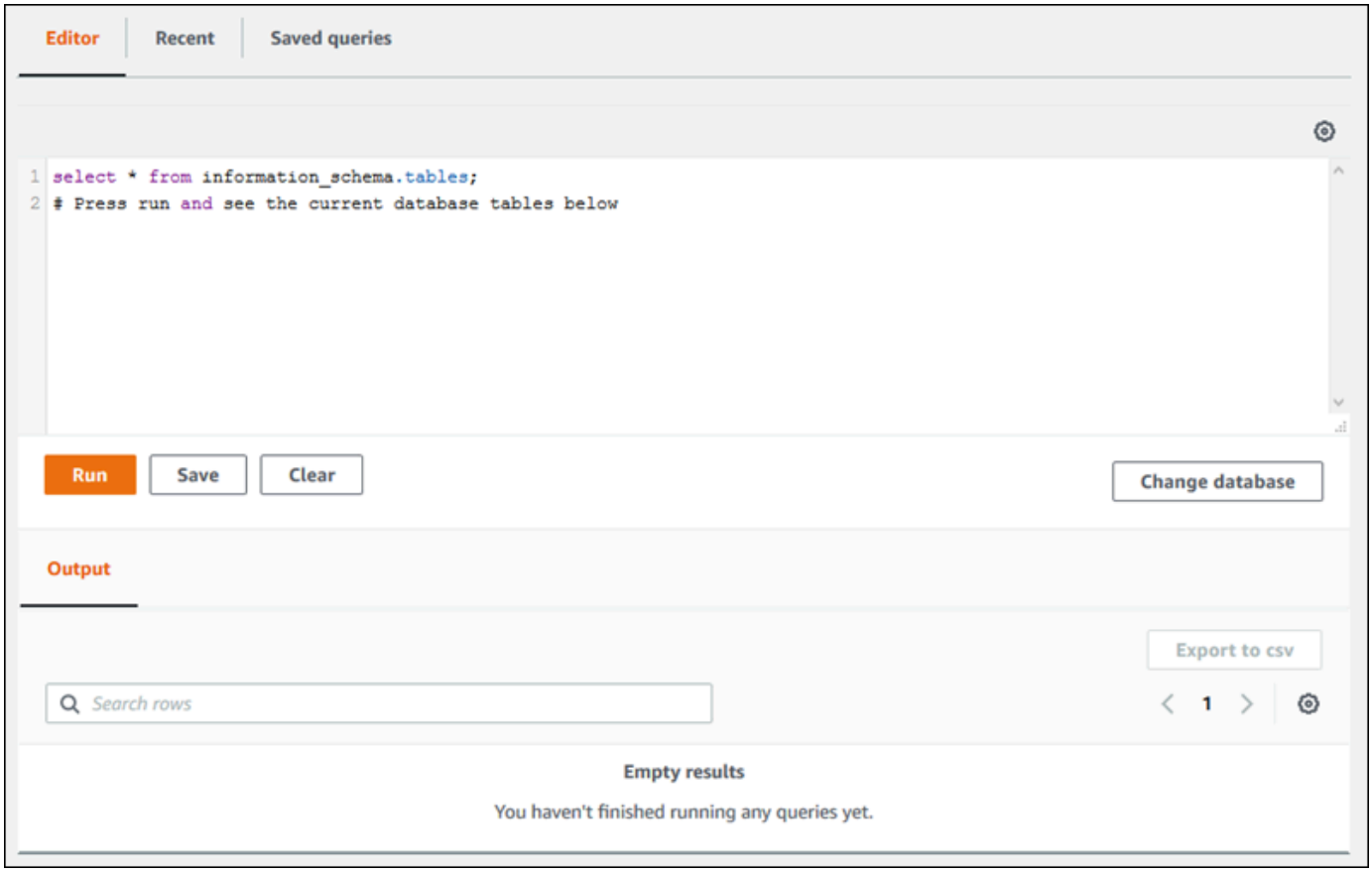
   c. [Enter database password (データベースパスワードの入力)] に、選択したデータベースユーザーのパスワードを入力します。

   d. 最後のボックスに、Aurora DB クラスターに使用するデータベースまたはスキーマの名前を入力します。

   e. [Connect to database (データベースに接続)] を選択します。

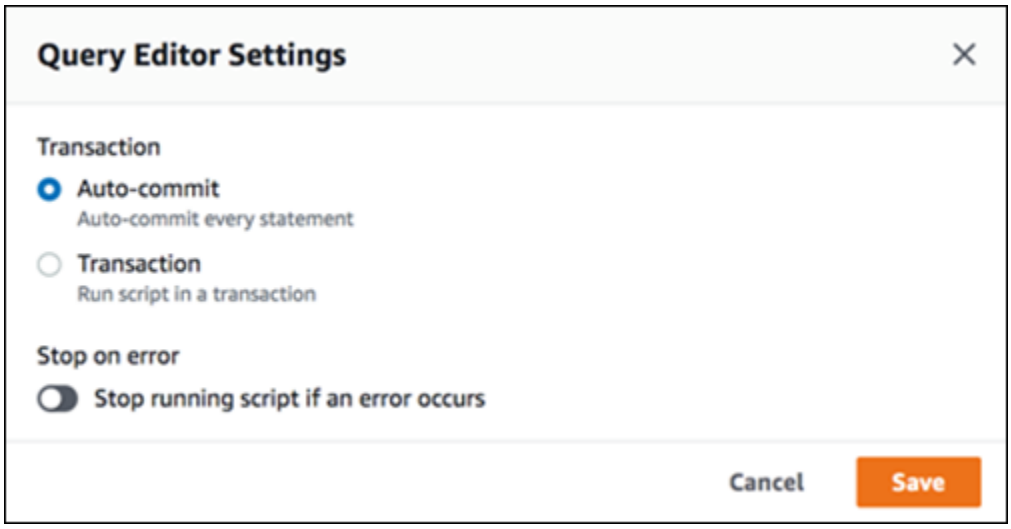
> **Note**
>
> 接続に成功すると、接続および認証情報が AWS Secrets Manager に保存されます。接続情報を再度入力する必要はありません。

7. クエリエディタで、データベースに対して実行する SQL クエリを入力します。



SQL ステートメントをそれぞれ自動的にコミットするか、トランザクションの一部としてスクリプト内で SQL ステートメントを実行することができます。この動作を制御するには、クエリウィンドウの上にある歯車アイコンを選択します。

[クエリエディタ設定] ウィンドウが表示されます。



[Auto-commit (自動コミット)] を選択した場合、SQL ステートメントはそれぞれ、自動的にコミットされます。[トランザクション] を選択すると、スクリプト内のステートメントのグループを実行できます。ステートメントは、スクリプトの終了前に明示的にコミットまたはロールバックしない限り、スクリプトの最後に自動的にコミットされます。また、[Stop on error (エラー時に停止)] を有効にして、エラー発生時にスクリプトの実行を停止するように選択することもできます。

> **Note**
>
> ステートメントのグループで、データ定義言語 (DDL) ステートメントによって前のデータ操作言語 (DML) ステートメントがコミットされる場合があります。また、COMMIT ステートメントおよび ROLLBACK ステートメントをスクリプトのステートメントのグループに含めることもできます。

[クエリエディタ設定] ウィンドウで選択したら、[保存] を選択します。

8. [実行] を選択するか、Ctrl+Enter を押すと、クエリの結果がクエリエディタに表示されます。

   クエリを実行したら、[保存] を選択し、そのクエリ結果を [Saved queries (保存したクエリ)] に保存します。

   [CSV へエクスポート] を選択して、クエリ結果をスプレッドシート形式でエクスポートします。

以前のクエリを検索、編集、再実行することができます。そのためには、[Recent (最新)] タブ、または [Saved queries (保存したクエリ)] タブを選択し、クエリテキストを選択して、[Run (実行)] を選択します。

データベースを変更するには、[Change database (データベースの変更)] を選択します。

Database Query Metadata Service (DBQMS) API リファレンス

データベースクエリメタデータサービス (dbqms) は、内部専用のサービスです。このサービスでは、Amazon RDS を含む複数の AWS サービスについて、最新および保存済みのクエリを、AWS Management Console のクエリエディタ用に提供します。

以下の DBQMS アクションがサポートされています。

トピック

- CreateFavoriteQuery
- CreateQueryHistory
- CreateTab
- DeleteFavoriteQueries

- DeleteQueryHistory
- DeleteTab
- DescribeFavoriteQueries
- DescribeQueryHistory
- DescribeTabs
- GetQueryString
- UpdateFavoriteQuery
- UpdateQueryHistory
- UpdateTab

CreateFavoriteQuery

新しいお気に入りクエリを保存します。各 ユーザーは最大 1000 個の保存済みクエリを作成できます。この制限は将来変更される可能性があります。

CreateQueryHistory

新しいクエリ履歴エントリを保存します。

CreateTab

新しいクエリタブを保存します。各 ユーザーは最大 10 個のクエリタブを作成できます。

DeleteFavoriteQueries

1 つまたは複数の保存されたクエリを削除します。

DeleteQueryHistory

クエリ履歴エントリを削除します。

DeleteTab

クエリタブエントリを削除します。

DescribeFavoriteQueries

特定のアカウントのユーザーによって作成された保存済みクエリを一覧表示します。

DescribeQueryHistory

クエリ履歴エントリを一覧表示します。

DescribeTabs

特定のアカウントでユーザーによって作成されたクエリタブを一覧表示します。

GetQueryString

クエリ ID からクエリテキスト全体を取得します。

UpdateFavoriteQuery

クエリ文字列、説明、名前、または有効期限を更新します。

UpdateQueryHistory

クエリ履歴のステータスを更新します。

UpdateTab

クエリタブ名とクエリ文字列を更新します。

Amazon Aurora 機械学習の使用

Amazon Aurora 機械学習を使用すると、必要に応じて、Aurora DB クラスターを次のいずれかの AWS 機械学習サービスと統合できます。それぞれが特定の機械学習のユースケースをサポートしています。

Amazon Bedrock

Amazon Bedrock は、AI 企業の主要な基盤モデルを API を通じて利用できるようにするフルマネージドサービスであり、生成 AI アプリケーションの構築とスケーリングに役立つデベロッパーツールを備えています。Amazon Bedrock では、どのサードパーティーの基盤モデルで推論を実行する場合でも料金がかかります。料金は、入力トークンと出力トークンの量、およびモデル用のプロビジョンドスループットの購入有無によって決まります。詳細については、「Amazon Bedrock ユーザーガイド」の「Amazon Bedrock とは」を参照してください。

Amazon Comprehend

Amazon Comprehend は、ドキュメントからインサイトを抽出するために使用されるマネージド型自然言語処理 (NLP) サービスです。Amazon Comprehend により、エンティティ、キーフレーズ、言語などの特徴を分析することで、ドキュメントの内容から感情を推測できます。詳細については、「Amazon Comprehend デベロッパーガイド」の「Amazon Comprehend とは」を参照してください。

SageMaker

Amazon SageMaker は、フルマネージド型の機械学習サービスです。データサイエンティストやデベロッパーは Amazon SageMaker を使用して、不正検出や製品レコメンデーションなど、さまざまな推論タスク用の機械学習モデルの構築、トレーニング、テストを行っています。機械学習モデルが本番環境で使用できるようになったら、Amazon SageMaker のホスト環境にデプロイできます。詳細については、「Amazon SageMaker デベロッパーガイド」の「Amazon SageMaker とは」を参照してください。

Amazon Comprehend を Aurora DB クラスターで使用すると、SageMaker を使用するよりも事前設定が少なくて済みます。AWS 機械学習が初めての場合は、Amazon Comprehend から始めることをお勧めします。

トピック

- Aurora MySQL で Amazon Aurora 機械学習を使用する

- Aurora PostgreSQL で Amazon Aurora 機械学習を使用する

Aurora MySQL で Amazon Aurora 機械学習を使用する

Aurora MySQL DB クラスターで Amazon Aurora 機械学習を使用することで、必要に応じて Amazon Bedrock、Amazon Comprehend, or Amazon SageMaker のいずれかを使用できます。それぞれが異なる機械学習のユースケースをサポートしています。

目次

Aurora 機械学習 を Aurora MySQL で使用するための要件

AWS 機械学習サービスは、お客様の本番環境で設定し、実行するマネージドサービスです。Aurora 機械学習は、Amazon Comprehend、SageMaker および Amazon Bedrock の統合をサポートしています。Aurora MySQL DB クラスターを設定して Aurora 機械学習を使用開始する前に、次の要件と前提条件を理解していることを確認してください。

- 機械学習サービスは、お使いの Aurora MySQL DB クラスターと同じ AWS リージョン で実行する必要があります。別のリージョンにある Aurora MySQL DB クラスターからの機械学習サービスは使用できません。
- Aurora MySQL DB クラスターが Amazon Bedrock、Amazon Comprehend または SageMaker サービスとは異なる仮想パブリッククラウド (VPC) にある場合、VPC のセキュリティグループが対象の Aurora 機械学習サービスへのアウトバウンド接続を許可する必要があります。詳細については、「Amazon VPC ユーザーガイド」の「セキュリティグループを使用して AWS リソースへのトラフィックを制御する」を参照してください。
- そのクラスターで Aurora 機械学習を使用する場合は、Aurora MySQL の下位バージョンを実行している Aurora クラスターをサポートされている上位バージョンにアップグレードできます。詳細については、「Amazon Aurora MySQL のデータベースエンジンの更新」を参照してください。
- Aurora MySQL DB クラスターは、カスタム DB クラスターパラメータグループを使用する必要があります。使用する各 Aurora 機械学習サービスの設定プロセスの最後で、サービス用に作成された関連する IAM ロールの Amazon リソースネーム (ARN) を追加します。事前に Aurora MySQL 用のカスタム DB クラスターパラメータグループを作成し、それを使用するように Aurora MySQL DB クラスターを設定することで、設定プロセスの最後に変更できるようにしておくことをお勧めします。
- SageMaker の場合:
  - 推論に使用する機械学習コンポーネントが設定され、使用できる状態になっている必要があります。Aurora MySQL DB クラスターの設定プロセス時に、SageMaker エンドポイントの ARN を用意する必要があります。SageMaker と連携してモデルを準備し、その他のタスクを処理するのは、チームのデータサイエンティストが最も適任だと考えられます。Amazon SageMaker を使い始めるには、「Amazon SageMaker の使用を開始する」を参照してください。推論とエンドポイントの詳細については、「リアルタイム推論」を参照してください。
  - SageMaker を独自のトレーニングデータで使用するには、Aurora 機械学習用に、Amazon S3 バケット を Aurora MySQL の設定の一部として設定する必要があります。これを行うには、SageMaker の統合設定の場合と同じ一般的なプロセスに従います。このオプションの設定プロセスの概要については、「SageMaker に Amazon S3 を使用するように Aurora MySQL DB クラスターを設定する (オプション)」を参照してください。

- Aurora グローバルデータベースでは、Aurora グローバルデータベースを構成するすべての AWS リージョン で使用する Aurora 機械学習サービスを設定します。例えば、Aurora グローバルデータベースで Aurora 機械学習を SageMaker で使用する場合は、すべての AWS リージョン にある Aurora MySQL DB クラスターごとに次のことを行います。
  - 同じ SageMaker トレーニングモデルとエンドポイントで、Amazon SageMaker サービスを設定します。これらには同じ名前を使用する必要があります。
  - 「Aurora 機械学習を使用するように Aurora MySQL DB クラスターを設定する」の説明のとおりに IAM ロールを作成します。
  - すべての AWS リージョン にある 各 Aurora MySQL DB クラスターのパラメータグループに、IAM ロールの ARN を追加します。

  これらのタスクでは、Aurora グローバルデータベースを構成するすべての AWS リージョン にある Aurora MySQL のバージョンで Aurora 機械学習を利用できる必要があります。

リージョンとバージョンの可用性

利用できる機能とそのサポートは、各 Aurora データベースエンジンの特定のバージョン、および AWS リージョン によって異なります。

- Amazon Comprehend と Amazon SageMaker で Aurora MySQL を利用できるバージョンとリージョンの詳細については、「Aurora MySQL を使用した Aurora Machine Learning」を参照してください。
- Amazon Bedrock は Aurora MySQL バージョン 3.06 以降でのみサポートされています。

  Amazon Bedrock で利用できるリージョンについては、「Amazon Bedrock ユーザーガイド」の「Amazon Bedrock のサポート対象モデル」を参照してください。

Aurora MySQL で Aurora 機械学習を使用する場合にサポートされている機能と制限事項

Aurora MySQL を Aurora 機械学習と合わせて使用する場合、次の制限事項が適用されます。

- Aurora 機械学習の拡張機能は、ベクトルインターフェイスをサポートしていません。
- Aurora 機械学習の統合は、トリガーで使用した場合はサポートされません。
- Aurora 機械学習の関数はバイナリロギング (バイナリログ) レプリケーションと互換性がありません。

  - 設定 `--binlog-format=STATEMENT` は、Aurora Machine Learning 関数の呼び出しに対して例外をスローします。
  - Aurora の機械学習関数はすべて非決定的であり、非決定的なストアド関数がこの binlog 形式と互換性がないためです。

  詳細については、MySQL ドキュメントの「バイナリログ形式」を参照してください。
- 常に生成される列を持つテーブルを呼び出すストアド関数はサポートされていません。これはすべての Aurora MySQL ストアド関数に適用されます。このカラムタイプの詳細については、MySQL ドキュメントの「CREATE TABLE と生成された列」を参照してください。
- Amazon Bedrock の関数は `RETURNS JSON` をサポートしていません。必要に応じて、`TEXT` から `JSON` に変換するのに `CONVERT` または `CAST` を使用できます。
- Amazon Bedrock はバッチリクエストをサポートしていません。
- Aurora MySQL は、`ContentType` の `text/csv` 値を介して、カンマ区切り値 (CSV) 形式を読み書きする SageMaker エンドポイントをサポートしています。この形式は以下の組み込みの SageMaker アルゴリズムで受け入れられています。
  - 線形学習
  - ランダムカットフォレスト
  - XGBoost

  これらのアルゴリズムの詳細については、「Amazon SageMaker デベロッパーガイド」の「アルゴリズムの選択」を参照してください。

Aurora 機械学習を使用するように Aurora MySQL DB クラスターを設定する

以下のトピックでは、これらの Aurora 機械学習サービスごとに個別の設定手順を参照できます。

トピック

- Amazon Bedrock を使用するための Aurora MySQL DB クラスターのセットアップ
- Amazon Comprehend を使用するように Aurora MySQL DB クラスターを設定する
- SageMaker を使用するように Aurora MySQL DB クラスターを設定する
  - SageMaker に Amazon S3 を使用するように Aurora MySQL DB クラスターを設定する (オプション)
- データベースユーザーに Aurora 機械学習へのアクセスを許可する

- Amazon Bedrock の関数へのアクセスの付与
- Amazon Comprehend 関数へのアクセスを許可する
- SageMaker 関数へのアクセス権の付与

Amazon Bedrock を使用するための Aurora MySQL DB クラスターのセットアップ

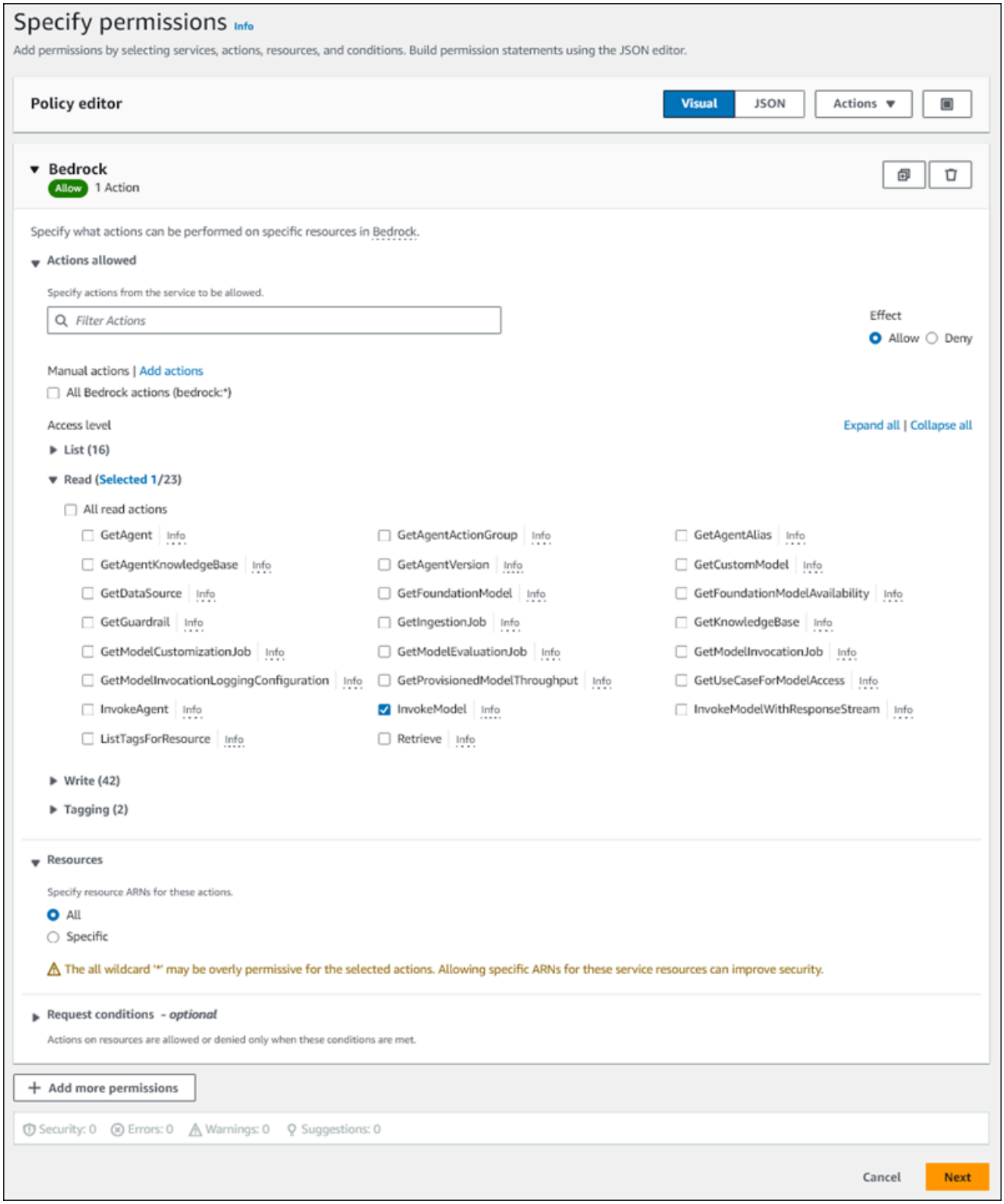
Aurora 機械学習では、Aurora MySQL DB クラスターが Amazon Bedrock サービスにアクセスして使用できるように、AWS Identity and Access Management (IAM) ロールとポリシーに依存します。以下の手順では、DB クラスターを Amazon Bedrock と統合できるように IAM アクセス許可ポリシーとロールを作成します。

IAM ポリシーを作成するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで、[ポリシー] を選択します。
3. [ポリシーの作成] を選択します。
4. [アクセス許可を指定] ページの [サービスの選択] で、[Bedrock] を選択します。

   Amazon Bedrock のアクセス許可が表示されます。
5. [読み取り] を展開し、[InvokeModel] を選択します。
6. [リソース] で [すべて] を選択します。

   [アクセス許可を指定] ページは次の図のようになるはずです。



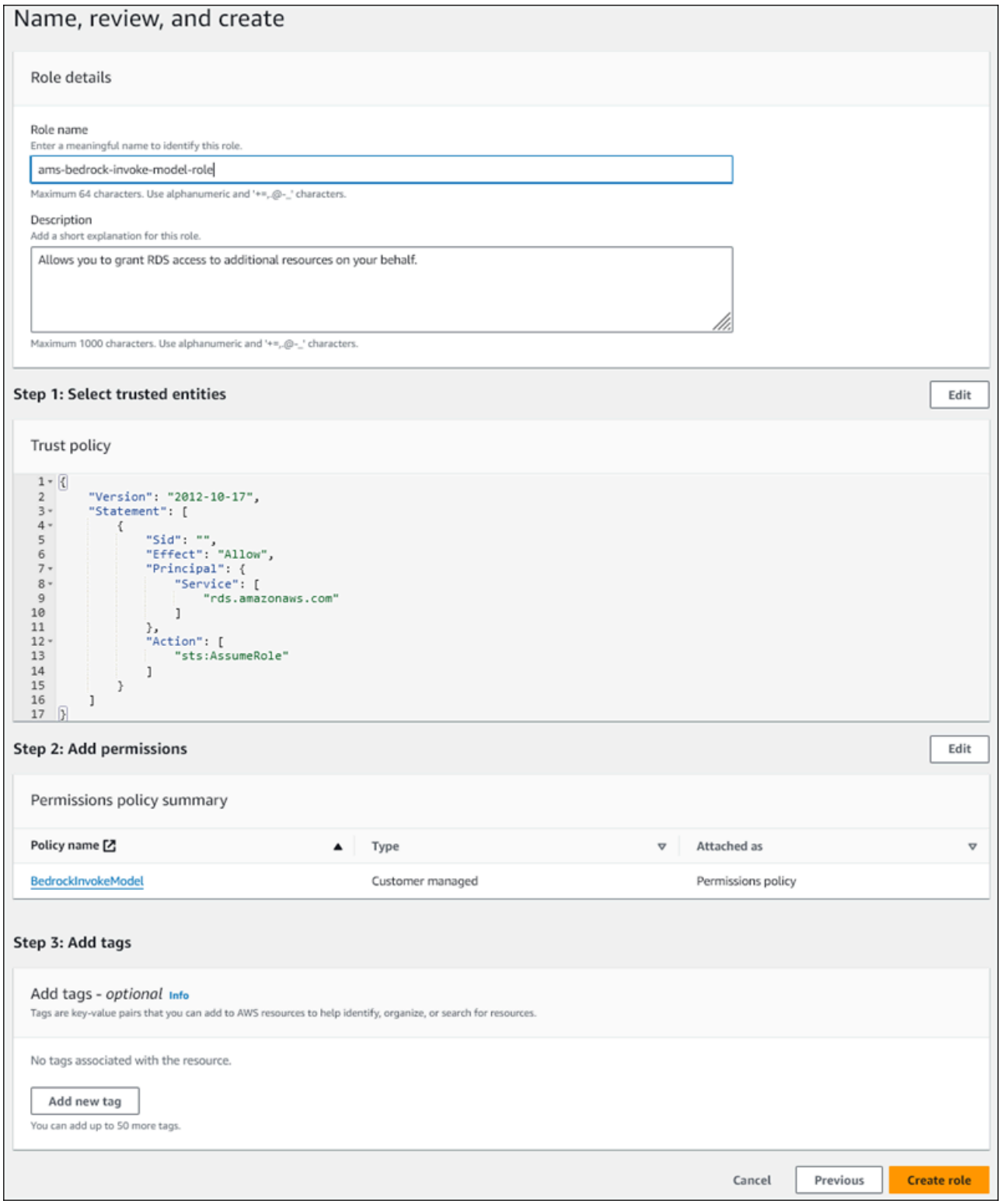
7. [Next] を選択します。
8. [レビューして作成] ページで、**BedrockInvokeModel** など、ポリシーに名前を入力します。
9. ポリシーを確認してから、[ポリシーを作成] を選択します。

次に、Amazon Bedrock のアクセス許可ポリシーを使用する IAM ロールを作成します。

IAM ロールを作成するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで [ロール] を選択します。
3. [ロールの作成] を選択します。
4. [信頼されたエンティティを選択] ページの [ユースケース] で [RDS] を選択します。
5. [RDS - データベースにロールを追加] を選択し、[次へ] を選択します。
6. [アクセス許可を追加] ページの [アクセス許可ポリシー] で、作成した IAM ポリシーを選択し、[次へ] を選択します。
7. [名前、確認および作成] ページで、**ams-bedrock-invoke-model-role** など、ロールに名前を入力します。

   ロールは以下の図ようになるはずです。



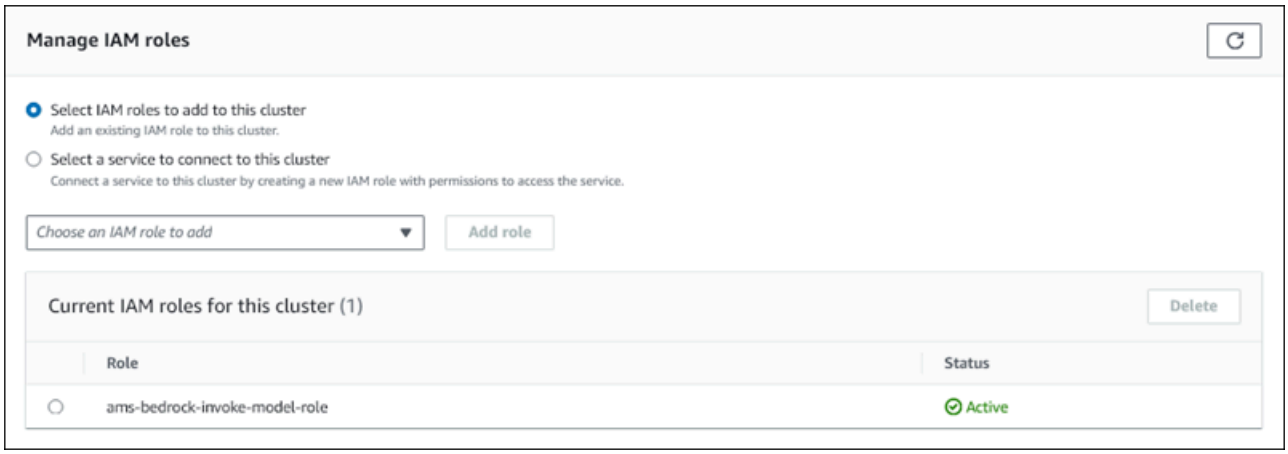
8. ロールを確認し、[ロールを作成] を選択します。

次に、Amazon Bedrock の IAM ロールを DB クラスターに関連付けます。

IAM ロールを DB クラスターと関連付けるには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインから [Databases (データベース)] を選択します。
3. Amazon Bedrock サービスに接続する Aurora MySQL DB クラスターを選択します。
4. [Connectivity & security (接続とセキュリティ)] タブを選択します。
5. [IAM ロールの管理] セクションで、[このクラスターに追加する IAM を選択] を選択します。
6. 作成した IAM を選択し、[ロールを追加] を選択します。

   IAM ロールは DB クラスターに関連付けられます。最初は [保留中] というステータスで、次に [アクティブ] になります。プロセスが完了すると、このクラスターの現在の IAM ロールリストにロールが表示されます。



Aurora MySQL DB クラスターに関連付けられているカスタム DB クラスターのパラメータグループの `aws_default_bedrock_role` パラメータにこの IAM ロールの ARN を追加する必要があります。Aurora MySQL DB クラスターでカスタム DB クラスターパラメータグループを使用していない場合は、Aurora MySQL DB クラスターで使用するパラメータグループを作成して、統合を完了させる必要があります。詳細については、「DB クラスターパラメータグループを使用する」を参照してください。

DB クラスターパラメータを設定するには

1. Amazon RDS コンソールで、Aurora MySQL DB クラスターの [Configuration] (設定) タブを開きます。

2. クラスター用に設定された DB クラスターパラメータグループを検索します。リンクを選択してカスタム DB クラスターパラメータグループを開き、[編集] を選択します。
3. DB クラスターパラメータグループから `aws_default_bedrock_role` パラメータを検索します。
4. [値] フィールドで、IAM ロールの ARN を入力します。
5. [Save changes] (変更の保存) を選択して設定を保存します。
6. このパラメータ設定を有効にするために、Aurora MySQL DB クラスターのプライマリインスタンスを再起動します。

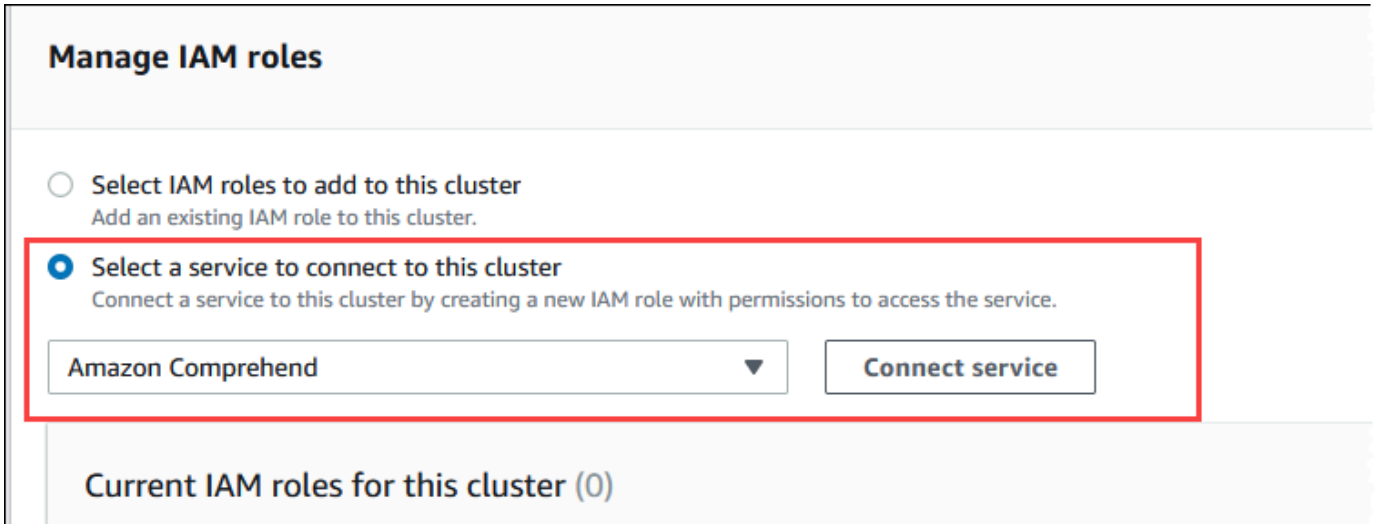
Amazon Bedrock の IAM 統合が完了しました。データベースユーザーに Aurora 機械学習へのアクセスを許可する で Amazon Bedrock を使用するために Aurora MySQL DB クラスターのセットアップを継続します。

Amazon Comprehend を使用するように Aurora MySQL DB クラスターを設定する

Aurora 機械学習では、Aurora MySQL DB クラスターが Amazon Comprehend サービスにアクセスして使用できるように、AWS Identity and Access Management ロールとポリシーに依存します。以下の手順では、クラスターで Amazon Comprehend を使用できるように、IAM ロールとポリシーを自動的に作成します。

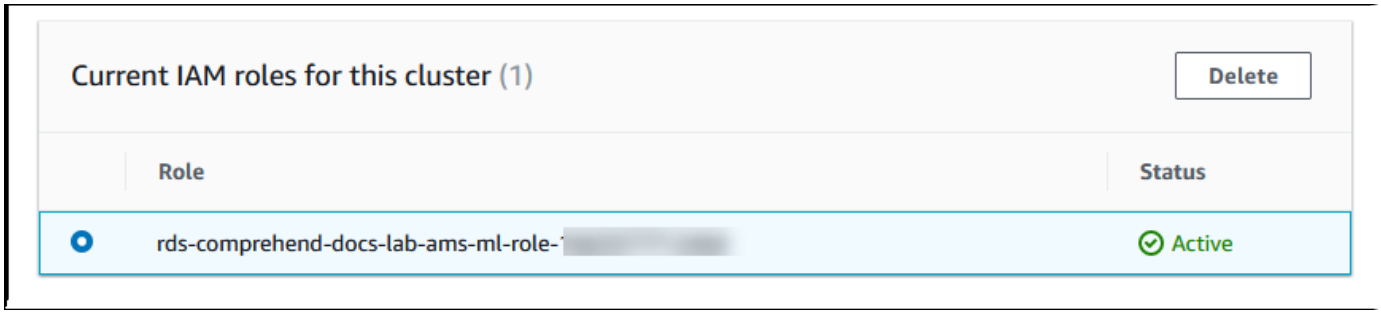
Amazon Comprehend を使用するように Aurora MySQL DB クラスターを設定には

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインから [Databases (データベース)] を選択します。
3. Amazon Comprehend サービスに接続する Aurora MySQL DB クラスターを選択します。
4. [Connectivity & security (接続とセキュリティ)] タブを選択します。
5. [IAM ロールの管理] セクションまでスクロールして、[このクラスターに接続するサービスを選択] を選択します。
6. メニューから [Amazon Comprehend]を選択し、[サービスを接続] を選択します。



7. [Connect cluster to Amazon Comprehend] (クラスターを Amazon Comprehend に接続する) ダイアログには、追加情報は必要ありません。ただし、Aurora と Amazon Comprehend の統合が現在プレビュー中であることを通知するメッセージが表示される場合があります。次に進む前に、そのメッセージを必ずお読みください。続行しない場合は、[キャンセル] を選択できます。
8. [Connect service] (サービスに接続する) を選択して統合プロセスを完了します。

   Aurora は IAM ロールを作成します。また、Aurora MySQL DB クラスターが Amazon Comprehend サービスを使用することを許可するポリシーを作成し、そのポリシーをロールにアタッチします。プロセスが完了すると、次の画像に示すように、このクラスターの現在の IAM ロールリストにロールが表示されます。



   Aurora MySQL DB クラスターに関連付けられているカスタム DB クラスターのパラメータグループの `aws_default_comprehend_role` パラメータにこの IAM ロールの ARN を追加する必要があります。Aurora MySQL DB クラスターでカスタム DB クラスターパラメータグループを使用していない場合は、Aurora MySQL DB クラスターで使用するパラメータグループを作成して、統合を完了させる必要があります。(詳しくは、「DB クラスターパラメータグループを使用する」を参照してください。)

カスタム DB クラスターパラメータグループを作成し、Aurora MySQL DB クラスターに関連付けると、以下の手順の実行を継続できます。

クラスターがカスタム DB クラスターパラメータグループを使用している場合は、以下のようにします。

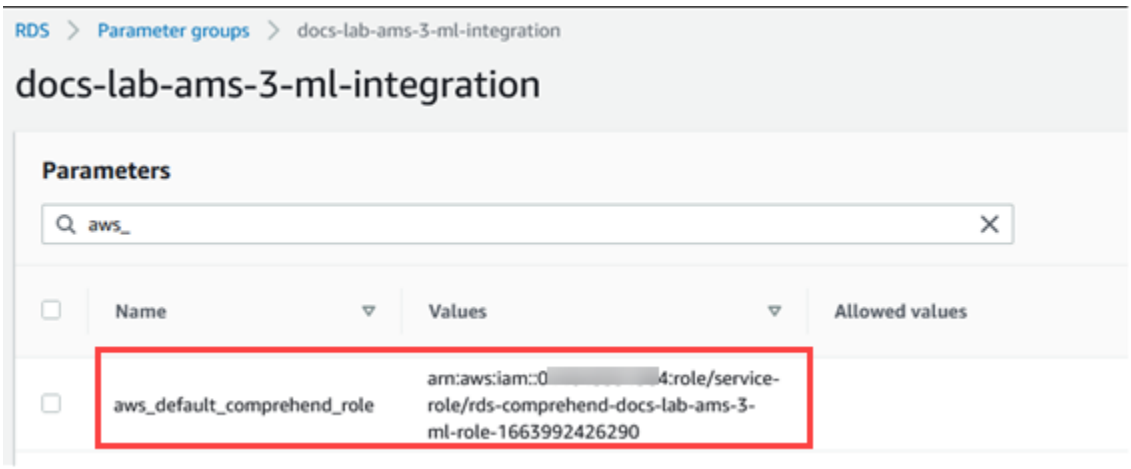
a. Amazon RDS コンソールで、Aurora MySQL DB クラスターの [Configuration] (設定) タブを開きます。

b. クラスター用に設定された DB クラスターパラメータグループを検索します。リンクを選択してカスタム DB クラスターパラメータグループを開き、[編集] を選択します。

c. DB クラスターパラメータグループから `aws_default_comprehend_role` パラメータを検索します。

d. [値] フィールドで、IAM ロールの ARN を入力します。

e. [Save changes] (変更の保存) を選択して設定を保存します。次の画像に、その例が示されています。

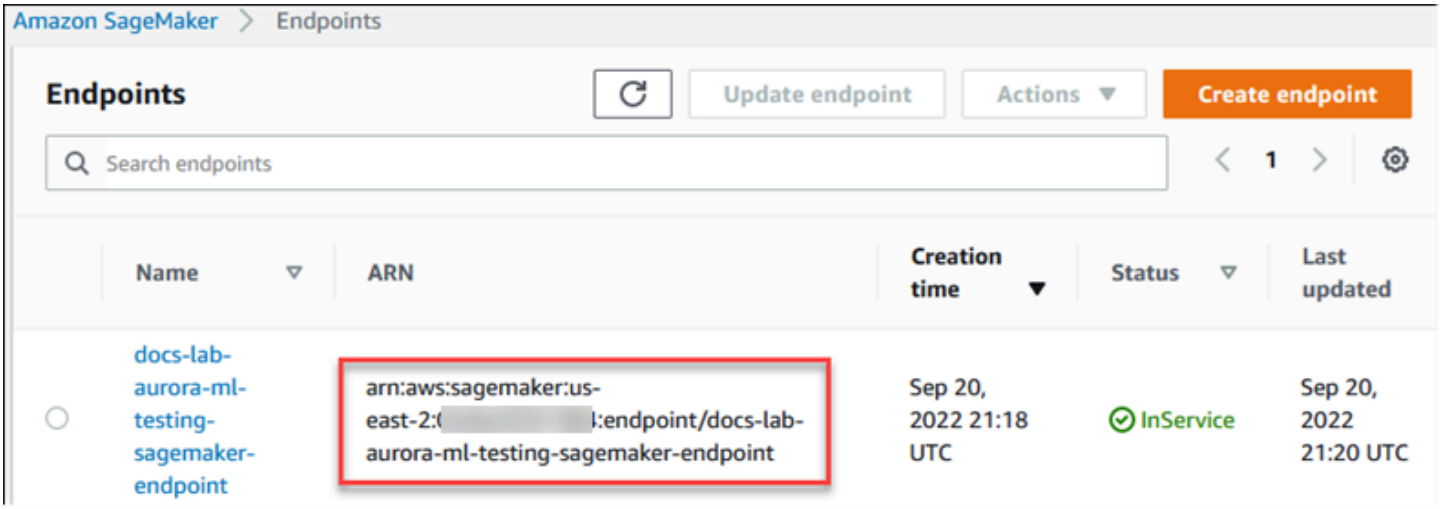


このパラメータ設定を有効にするために、Aurora MySQL DB クラスターのプライマリインスタンスを再起動します。

Amazon Comprehend の IAM 統合が完了しました。適切なデータベースユーザーにアクセス権を付与して、Amazon Comprehend と連携するように Aurora MySQL DB クラスターの設定を続けてください。

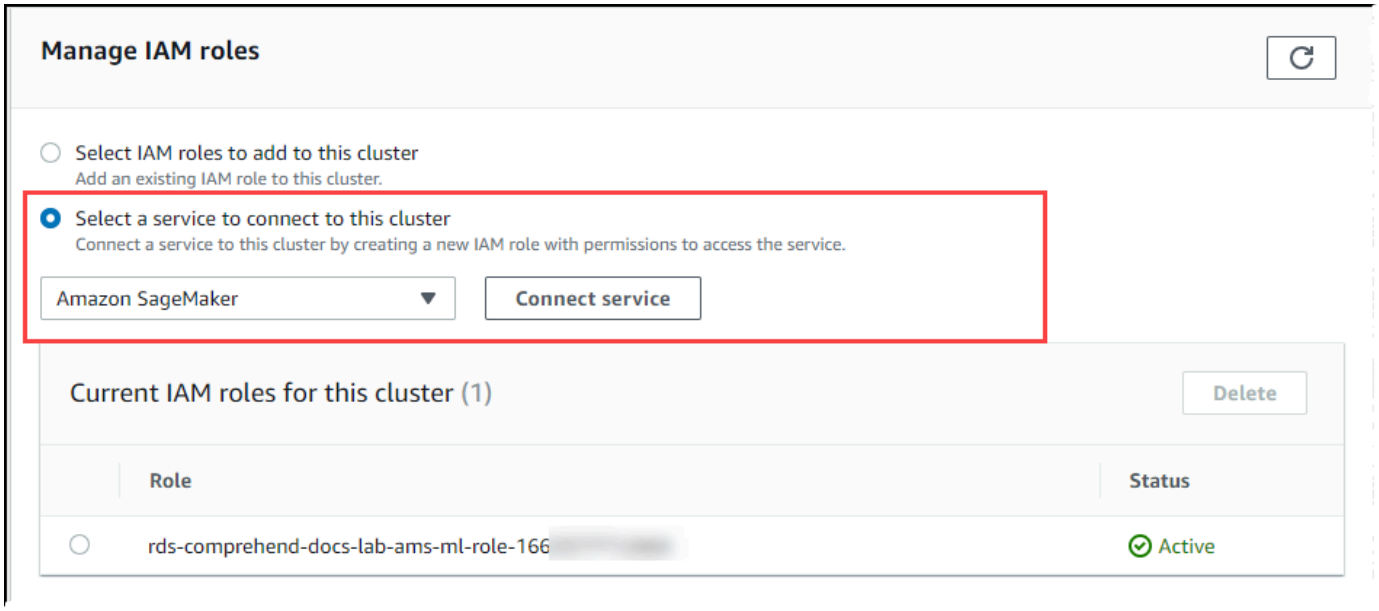
SageMaker を使用するように Aurora MySQL DB クラスターを設定する

以下の手順では、Aurora MySQL DB クラスターで SageMaker を使用できるように、IAM ロールとポリシーを自動的に作成します。この手順を実行する前に、SageMaker エンドポイントが使用可能であることを確認し、必要なときに入力できるようにしてください。通常、チームのデータサイエンティストによって Aurora MySQL DB クラスターから使用できるエンドポイントを作成する作業を行います。このようなエンドポイントは SageMaker コンソールにあります。ナビゲーションペインで [Inference] (推論) メニューを開き、[Endpoints] (エンドポイント) を選択します。次の画像に、その例が示されています。

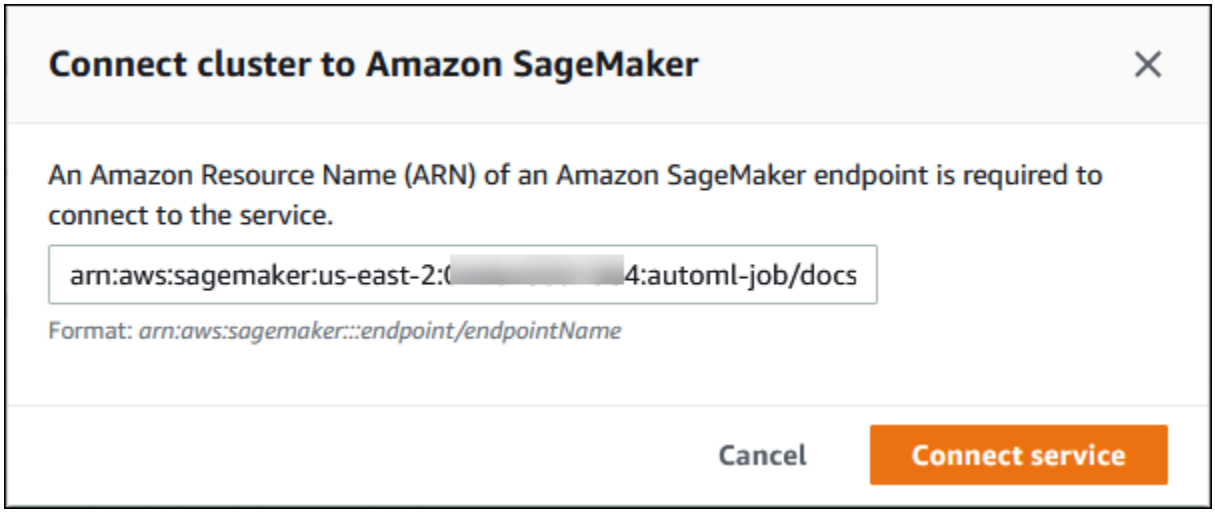


SageMaker を使用するように Aurora MySQL DB クラスターを設定するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. Amazon RDS のナビゲーションメニューから [Databases] (データベース) を選択して、SageMaker サービスに接続する Aurora MySQL DB クラスターを選択します。
3. [Connectivity & security (接続とセキュリティ)] タブを選択します。
4. [Manage IAM roles] (IAM ロールの管理) セクションまでスクロールして、[Select a service to connect to this cluster] (このクラスターに接続するサービスの選択) を選択します。セレクタから [SageMaker] を選択します。



5. [Connect service (サービスの接続)] を選択します。
6. [クラスターを SageMaker に接続する] ダイアログで、SageMaker エンドポイントの ARN を入力します。



7. Aurora は IAM ロールを作成します。また、Aurora MySQL DB クラスターが SageMaker サービスを使用することを許可するポリシーを作成し、そのポリシーをロールにアタッチします。プロセスが完了すると、このクラスターの現在の IAM ロールリストにロールが表示されます。
8. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
9. AWS Identity and Access Management ナビゲーションメニューの [Access management] (アクセス管理) セクションから [Roles] (ロール) を選択します。
10. リストの中からロールを検索します。その名前は、次のパターンを使用しています。

```
rds-sagemaker-your-cluster-name-role-auto-generated-digits
```

11. ロールの [Summary] (概要) ページを開いて ARN を検索します。ARN をメモするか、コピーウィジェットでコピーします。
12. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
13. Aurora MySQL DB クラスターを選択して、その [Configuration] (設定) タブを選択します。
14. DB クラスターパラメータグループを検索し、リンクを選択してカスタム DB クラスターパラメータグループを開きます。`aws_default_sagemaker_role` パラメータを検索し、[Value] (値) フィールドに IAM ロールの ARN を入力して、[Save] (保存) で設定を保存します。
15. このパラメータ設定を有効にするために、Aurora MySQL DB クラスターのプライマリインスタンスを再起動します。

IAM の設定が完了しました。適切なデータベースユーザーにアクセス権を付与して、SageMaker と連携するように Aurora MySQL DB クラスターの設定を続けてください。

構築済みの SageMaker コンポーネントを使用せず、SageMaker モデルをトレーニングに使用する場合は、次の「SageMaker に Amazon S3 を使用するように Aurora MySQL DB クラスターを設定する (オプション)」の説明のように Amazon S3 バケットを Aurora MySQL DB クラスターに追加する必要もあります。

SageMaker に Amazon S3 を使用するように Aurora MySQL DB クラスターを設定する (オプション)

SageMaker が提供する構築済みコンポーネントを使用せず、独自のモデルで SageMaker を使用するには、Aurora MySQL DB クラスターが使用する Amazon S3 バケットを設定する必要があります。Amazon S3 バケットの作成の詳細については、Amazon Simple Storage Service ユーザーガイドの「バケットの作成」を参照してください。

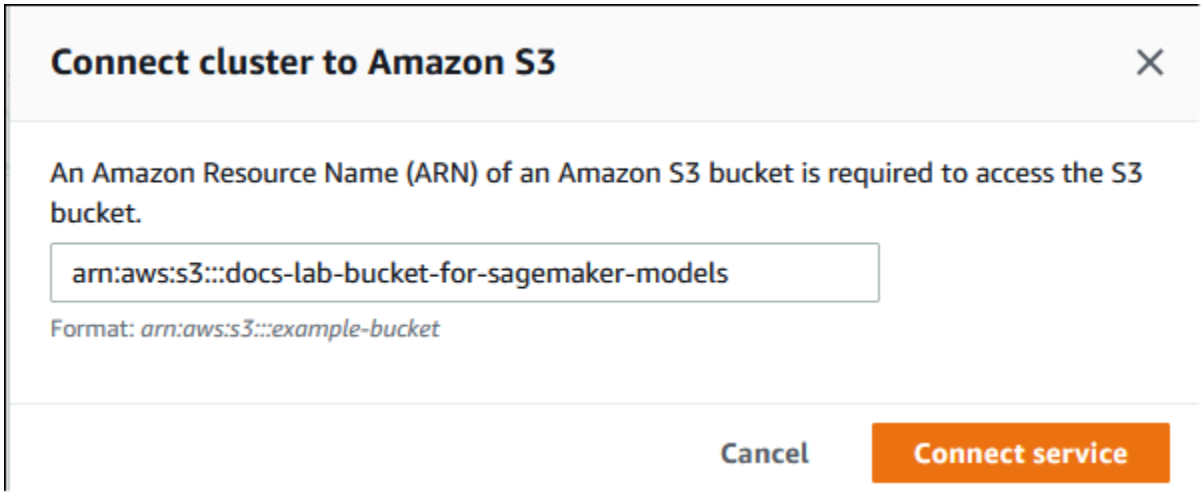
SageMaker に Amazon S3 バケットを使用するように Aurora MySQL DB クラスターを設定するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. Amazon RDS のナビゲーションメニューから [Databases] (データベース) を選択して、SageMaker サービスに接続する Aurora MySQL DB クラスターを選択します。
3. [Connectivity & security (接続とセキュリティ)] タブを選択します。

4. [Manage IAM roles] (IAM ロールの管理) セクションまでスクロールして、[Select a service to connect to this cluster] (このクラスターに接続するサービスの選択) を選択します。セレクタから [Amazon S3] を選択します。

5. [Connect service (サービスの接続)] を選択します。

6. [クラスターを Amazon S3 に接続する] ダイアログで、以下の画像のように Amazon S3 バケットの ARN を入力します。



7. [Connect service] (サービスに接続する) を選択してこのプロセスを完了します。

SageMaker で Amazon S3 バケットを使用する方法の詳細については、「Amazon SageMaker デベロッパーガイド」の「トレーニングデータセットのアップロードと出力データの保存に Amazon S3 バケットを指定する」を参照してください。SageMaker の使用方法の詳細については、「Amazon SageMaker デベロッパーガイド」の「Amazon SageMaker ノートブックインスタンスの使用開始」を参照してください。

データベースユーザーに Aurora 機械学習へのアクセスを許可する

データベースユーザーには、Aurora 機械学習の関数を呼び出すアクセス許可を付与する必要があります。権限の付与方法は、以下の説明のように、Aurora MySQL DB クラスターに使用する MySQL のバージョンによって異なります。その方法は、Aurora MySQL DB クラスターが使用する MySQL のバージョンによって異なります。

- Aurora MySQL バージョン 3 (MySQL 8.0 互換) の場合、データベースユーザーに適切なデータベースロールを付与する必要があります。詳細については、「MySQL 8.0 リファレンスマニュアル」の「ロールの使用」を参照してください。

- Aurora MySQL バージョン 2 (MySQL 5.7 互換) の場合、データベースユーザーに権限が付与されます。詳細は、「MySQL 5.7 リファレンスマニュアル」の「アクセスコントロールおよびアカウントマネジメント」を参照してください。

次の表は、データベースユーザーが機械学習の関数を使用するために必要なロールと権限を示しています。

Aurora MySQL バージョン 3 (ロール)	Aurora MySQL バージョン 2 (権限)
AWS_BEDROCK_ACCESS	–
AWS_COMPREHEND_ACCESS	INVOKE COMPREHEND
AWS_SAGEMAKER_ACCESS	INVOKE SAGEMAKER

Amazon Bedrock の関数へのアクセスの付与

データベースユーザーが Amazon Bedrock の関数にアクセスできるようにするには、次の SQL ステートメントを使用します。

```
GRANT AWS_BEDROCK_ACCESS TO user@domain-or-ip-address;
```

データベースユーザーには、Amazon Bedrock と連携するために作成した関数の EXECUTE アクセス許可も付与する必要があります。

```
GRANT EXECUTE ON FUNCTION database_name.function_name TO user@domain-or-ip-address;
```

最後に、データベースユーザーは各自のロールを AWS_BEDROCK_ACCESS に設定する必要があります。

```
SET ROLE AWS_BEDROCK_ACCESS;
```

これで Amazon Bedrock の関数が使用可能になりました。

Amazon Comprehend 関数へのアクセスを許可する

データベースユーザーが Amazon Comprehend 関数にアクセスできるようにするには、Aurora MySQL バージョンに適合した適切なステートメントを使用します。

- Aurora MySQL バージョン 3 (MYSQL 8.0 互換)

```
GRANT AWS_COMPREHEND_ACCESS TO user@domain-or-ip-address;
```

- Aurora MySQL バージョン 2 (MYSQL 5.7 互換)

```
GRANT INVOKE COMPREHEND ON *.* TO user@domain-or-ip-address;
```

Amazon Comprehend 関数が使用可能になりました。使用例については、「Aurora MySQL DB クラスターで Amazon Comprehend を使用する」を参照してください。

SageMaker 関数へのアクセス権の付与

データベースユーザーが SageMaker 関数にアクセスできるようにするには、Aurora MySQL バージョンに適合した適切なステートメントを使用します。

- Aurora MySQL バージョン 3 (MYSQL 8.0 互換)

```
GRANT AWS_SAGEMAKER_ACCESS TO user@domain-or-ip-address;
```

- Aurora MySQL バージョン 2 (MYSQL 5.7 互換)

```
GRANT INVOKE SAGEMAKER ON *.* TO user@domain-or-ip-address;
```

データベースユーザーには、SageMaker と連携するために作成した関数の EXECUTE 権限も付与する必要があります。SageMaker エンドポイントのサービスを呼び出すために、db1.anomoly_score と db2.company_forecasts という 2 つの関数を作成したとします。以下の例に示すように EXECUTE 権限を付与します。

```
GRANT EXECUTE ON FUNCTION db1.anomaly_score TO user1@domain-or-ip-address1;
GRANT EXECUTE ON FUNCTION db2.company_forecasts TO user2@domain-or-ip-address2;
```

これで SageMaker 関数が使用できるようになりました。使用例については、「Aurora MySQL DB クラスターで SageMaker を使用する」を参照してください。

Aurora MySQL DB クラスターでの Amazon Bedrock の使用

Amazon Bedrock を使用するには、モデルを呼び出すユーザー定義関数 (UDF) を Aurora MySQL データベースに作成します。詳細については、「Amazon Bedrock ユーザーガイド」の「Amazon Bedrock でサポートされているモデル」を参照してください。

UDF は次の構文を使用します。

```
CREATE FUNCTION function_name (argument type)
        [DEFINER = user]
    RETURNS mysql_data_type
    [SQL SECURITY {DEFINER | INVOKER}]
    ALIAS AWS_BEDROCK_INVOKE_MODEL
    MODEL ID 'model_id'
    [CONTENT_TYPE 'content_type']
    [ACCEPT 'content_type']
    [TIMEOUT_MS timeout_in_milliseconds];
```

- Amazon Bedrock の関数は RETURNS JSON をサポートしていません。必要に応じて、TEXT から JSON に変換するのに CONVERT または CAST を使用できます。
- CONTENT_TYPE またはACCEPT を指定しなかった場合、デフォルトは application/json です。
- TIMEOUT_MS を指定しなかった場合、aurora_ml_inference_timeout の値が使用されます。

例えば、次の UDF は Amazon Titan Text Express モデルを呼び出します。

```
CREATE FUNCTION invoke_titan (request_body TEXT)
    RETURNS TEXT
    ALIAS AWS_BEDROCK_INVOKE_MODEL
    MODEL ID 'amazon.titan-text-express-v1'
    CONTENT_TYPE 'application/json'
    ACCEPT 'application/json';
```

DB ユーザーがこの関数を使用できるようにするには、次の SQL コマンドを使用します。

```
GRANT EXECUTE ON FUNCTION database_name.invoke_titan TO user@domain-or-ip-address;
```

これで、以下の例に示されているように、ユーザーは他の関数と同様に invoke_titan を呼び出すことができます。Amazon Titan のテキストモデルに従って、リクエスト本文をフォーマットしてください。

```
CREATE TABLE prompts (request varchar(1024));
INSERT INTO prompts VALUES (
'{
    "inputText": "Generate synthetic data for daily product sales in various categories
 - include row number, product name, category, date of sale and price. Produce output
 in JSON format. Count records and ensure there are no more than 5.",
    "textGenerationConfig": {
        "maxTokenCount": 1024,
        "stopSequences": [],
        "temperature":0,
        "topP":1
    }
}');

SELECT invoke_titan(request) FROM prompts;

{"inputTextTokenCount":44,"results":[{"tokenCount":296,"outputText":"
```tabular-data-json
{
    "rows": [
        {
            "Row Number": "1",
            "Product Name": "T-Shirt",
            "Category": "Clothing",
            "Date of Sale": "2024-01-01",
            "Price": "$20"
        },
        {
            "Row Number": "2",
            "Product Name": "Jeans",
            "Category": "Clothing",
            "Date of Sale": "2024-01-02",
            "Price": "$30"
        },
        {
            "Row Number": "3",
            "Product Name": "Hat",
            "Category": "Accessories",
            "Date of Sale": "2024-01-03",
```
```

```
            "Price": "$15"
        },
        {
            "Row Number": "4",
            "Product Name": "Watch",
            "Category": "Accessories",
            "Date of Sale": "2024-01-04",
            "Price": "$40"
        },
        {
            "Row Number": "5",
            "Product Name": "Phone Case",
            "Category": "Accessories",
            "Date of Sale": "2024-01-05",
            "Price": "$25"
        }
    ]
}
```","completionReason":"FINISH"}]}

使用する他のモデルについては、リクエスト本文をそのモデルに合わせて適切にフォーマットしてください。詳細については、「Amazon Bedrock ユーザーガイド」の「基盤モデルの推論パラメータ」を参照してください。

Aurora MySQL DB クラスターで Amazon Comprehend を使用する

Aurora MySQL の場合、Aurora 機械学習には Amazon Comprehend とテキストデータを操作するために、次のような 2 つの組み込み関数が用意されています。分析するテキスト (`input_data`) と言語 (`language_code`) を指定します。

aws_comprehend_detect_sentiment

この関数は、テキストの感情的な姿勢がポジティブ、中立、ネガティブ、または混合かを識別します。この関数のリファレンスドキュメントは次のとおりです。

```
aws_comprehend_detect_sentiment(
  input_text,
  language_code
  [,max_batch_size]
)
```

詳細については、「Amazon Comprehend デベロッパーガイド」の「感情」を参照してください。

aws_comprehend_detect_sentiment_confidence

この関数は、特定のテキストについて検出された感情の信頼度を測定します。aws_comprehend_detect_sentiment 関数によってテキストに割り当てられた感情の信頼度を示す値 (タイプ double) を返します。信頼度は 0 ~ 1 の間の統計的メトリクスです。信頼度が高いほど、その結果に与える重みが増えます。関数のドキュメントの概要は次のとおりです。

```
aws_comprehend_detect_sentiment_confidence(
  input_text,
  language_code
  [,max_batch_size]
)
```

いずれの関数 (aws_comprehend_detect_sentiment_confidence、aws_comprehend_detect_sentiment) において、指定がない場合は max_batch_size にはデフォルト値である 25 を使用します。バッチサイズは常に 0 より大きい必要があります。max_batch_size を使用すると、Amazon Comprehend 関数呼び出しのパフォーマンスを調整しやすくなります。バッチサイズを大きくすると、Aurora MySQL DB クラスターでのメモリ使用量が増えるため、パフォーマンスが加速します。(詳しくは、「Aurora MySQL で Aurora 機械学習を使用した場合のパフォーマンスに関する考慮事項」を参照してください。)

Amazon Comprehend の感情検出関数のパラメータと戻り値のタイプについては、「DetectSentiment」を参照してください。

Example 例: Amazon Comprehend 関数を使用する簡単なクエリ

ここでは、この 2 つの関数を呼び出して、サポートチームに対する顧客満足度を確認する簡単なクエリの例を示します。ヘルプをリクエストするたびに顧客からのフィードバックを保存するデータベーステーブル (support) があるとします。このクエリ例では、両方の組み込み関数をテーブルの feedback 列のテキストに適用し、結果を出力します。この関数によって返される信頼値は、0.0 ~ 1.0 の間の倍数です。出力の読みやすさを向上させるため、このクエリでは小数点以下を 6 桁に丸めて結果を出力します。比較しやすいように、このクエリでは信頼度が最も高い結果から降順で結果をソートします。

```
SELECT feedback AS 'Customer feedback',
```

```
    aws_comprehend_detect_sentiment(feedback, 'en') AS Sentiment,
    ROUND(aws_comprehend_detect_sentiment_confidence(feedback, 'en'), 6)
    AS Confidence FROM support
    ORDER BY Confidence DESC;
+-----------------------------------------------------------+-----------+------------+
| Customer feedback                                         | Sentiment | Confidence |
+-----------------------------------------------------------+-----------+------------+
| Thank you for the excellent customer support!             | POSITIVE  |   0.999771 |
| The latest version of this product stinks!                | NEGATIVE  |   0.999184 |
| Your support team is just awesome! I am blown away.       | POSITIVE  |   0.997774 |
| Your product is too complex, but your support is great.   | MIXED     |   0.957958 |
| Your support tech helped me in fifteen minutes.           | POSITIVE  |   0.949491 |
| My problem was never resolved!                            | NEGATIVE  |   0.920644 |
| When will the new version of this product be released?    | NEUTRAL   |   0.902706 |
| I cannot stand that chatbot.                              | NEGATIVE  |   0.895219 |
| Your support tech talked down to me.                      | NEGATIVE  |   0.868598 |
| It took me way too long to get a real person.             | NEGATIVE  |   0.481805 |
+-----------------------------------------------------------+-----------+------------+
 10 rows in set (0.1898 sec)
```

Example 例: 特定の信頼度を超えるテキストの平均感情を判別する。

一般的な Amazon Comprehend クエリは、感情が特定の値であり、かつ信頼レベルが特定の数値より大きい行を検索します。例えば、次のクエリは、データベース内のドキュメントの平均感情を判別する方法を示しています。クエリでは、評価の信頼度が 80% 以上のドキュメントのみが考慮されます。

```
SELECT AVG(CASE aws_comprehend_detect_sentiment(productTable.document, 'en')
  WHEN 'POSITIVE' THEN 1.0
  WHEN 'NEGATIVE' THEN -1.0
  ELSE 0.0 END) AS avg_sentiment, COUNT(*) AS total
FROM productTable
WHERE productTable.productCode = 1302 AND
    aws_comprehend_detect_sentiment_confidence(productTable.document, 'en') >= 0.80;
```

Aurora MySQL DB クラスターで SageMaker を使用する

Aurora MySQL DB クラスターから SageMaker 関数を使用するには、SageMaker エンドポイントとその推論機能への呼び出しを埋め込むストアド関数を作成する必要があります。そのためには、Aurora MySQL DB クラスターの他の処理タスクと同じように MySQL の `CREATE FUNCTION` を使用します。

推論のために SageMaker にデプロイされたモデルを使用するには、ストアド関数の MySQL データ定義言語 (DDL) ステートメントを使用して、ユーザー定義関数を作成します。各ストアド関数は、モデルをホストする SageMaker エンドポイントを表します。そのような関数を定義するときには、モデルへの入力パラメータ、呼び出す特定の SageMaker エンドポイント、および戻り値の型を指定します。この関数は、モデルを入力パラメータに適用した後、SageMaker エンドポイントによって計算された推論を返します。

すべての Aurora 機械学習ストアド関数が、数値型または VARCHAR を返します。BIT 以外の数値型を使用できます。JSON、BLOB、TEXT、DATE などの型は使用できません。

以下の例は、SageMaker を操作するための CREATE FUNCTION 構文を示しています。

```
CREATE FUNCTION function_name (
     arg1 type1,
     arg2 type2, ...)
     [DEFINER = user]
  RETURNS mysql_type
  [SQL SECURITY { DEFINER | INVOKER } ]
  ALIAS AWS_SAGEMAKER_INVOKE_ENDPOINT
  ENDPOINT NAME 'endpoint_name'
  [MAX_BATCH_SIZE max_batch_size];
```

これは通常の CREATE FUNCTION DDL ステートメントの拡張です。SageMaker 関数を定義する CREATE FUNCTION ステートメントでは、関数本体を指定しません。代わりに、関数本体が通常実行されるキーワード ALIAS を指定します。Aurora 機械学習は現在、この拡張構文の aws_sagemaker_invoke_endpoint のみをサポートしています。endpoint_name パラメータを指定する必要があります。SageMaker エンドポイントは、モデルごとに異なる特性を持つことができます。

> **Note**
>
> CREATE FUNCTION の詳細については、「MySQL 8.0 リファレンスマニュアルの」の「CREATE PROCEDURE および CREATE FUNCTION ステートメント」を参照してください。

max_batch_size パラメータはオプションです。デフォルトでは、最大バッチサイズは 10,000 です。このパラメータを関数で使用すると、SageMaker へのバッチリクエストで処理される入力の最大数を制限できます。max_batch_size パラメータを使用すると、大きすぎる入力によって引き起

こされるエラーを回避したり、SageMaker からレスポンスが返される時間を短縮したりすることができます。このパラメータは、SageMaker リクエスト処理に使用される内部バッファのサイズに影響します。max_batch_size に指定する値が大きすぎると、DB インスタンスでかなりの規模のメモリオーバーヘッドが発生する可能性があります。

MANIFEST の設定はデフォルト値 OFF のままにすることをお勧めします。MANIFEST ON オプションを使用できますが、一部の SageMaker 機能では、このオプションでエクスポートされた CSV を直接使用できません。マニフェスト形式は、SageMaker から期待されるマニフェスト形式と互換性がありません。

SageMaker モデルごとに個別のストアド関数を作成します。エンドポイントが特定のモデルに関連付けられ、各モデルは異なるパラメータを受け入れるため、このように関数をモデルにマッピングする必要があります。モデル入力とモデル出力タイプに SQL タイプを使用すると、AWS サービス間でデータをやり取りするタイプ変換エラーを回避できます。モデルを適用できるユーザーを制御できます。また、ランタイム特性を制御するには、最大バッチサイズを表すパラメータを指定します。

現在、すべての Aurora 機械学習関数に NOT DETERMINISTIC プロパティがあります。このプロパティを明示的に指定しなかった場合は、Aurora によって自動的に NOT DETERMINISTIC に設定されます。この要件は、データベースに通知せずに SageMaker モデルを変更できることで実現しています。その場合、Aurora 機械学習関数を呼び出すと、単一のトランザクション内の同じ入力に対して異なる結果が返される可能性があります。

CONTAINS SQL ステートメントで、特性 NO SQL、READS SQL DATA、MODIFIES SQL DATA、または CREATE FUNCTION を使用することはできません。

以下は、SageMaker エンドポイントを呼び出して異常を検出する使用例です。SageMaker エンドポイント random-cut-forest-model があります。対応するモデルは、random-cut-forest アルゴリズムによって既にトレーニングされています。モデルは、入力ごとに異常スコアを返します。この例は、スコアが平均スコアから 3 スタンダード偏差 (約 99.9 パーセンタイル) より大きいデータポイントを示しています。

```
CREATE FUNCTION anomaly_score(value real) returns real
  alias aws_sagemaker_invoke_endpoint endpoint name 'random-cut-forest-model-demo';

set @score_cutoff = (select avg(anomaly_score(value)) + 3 * std(anomaly_score(value))
 from nyc_taxi);

select *, anomaly_detection(value) score from nyc_taxi
  where anomaly_detection(value) > @score_cutoff;
```

文字列を返す SageMaker 関数の文字セット要件

文字列値を返す SageMaker 関数の戻り値の型として、文字セット utf8mb4 を指定することをお勧めします。それが実用的でない場合は、文字セット utf8mb4 で表される値を保持できるよう、戻り値の型に十分な長さの文字列長を使用します。次の例は、関数の文字セット utf8mb4 を宣言する方法を示しています。

```
CREATE FUNCTION my_ml_func(...) RETURNS VARCHAR(5) CHARSET utf8mb4 ALIAS ...
```

現在、文字列を返す各 SageMaker 関数は、戻り値に文字セット utf8mb4 を使用します。SageMaker 関数がその戻り値の型に対して暗黙的または明示的に異なる文字セットを宣言している場合でも、戻り値はこの文字セットを使用します。SageMaker 関数が戻り値に別の文字セットを宣言している場合、長さが十分でないテーブル列に格納すると、返されたデータが暗黙的に切り捨てられる可能性があります。例えば、DISTINCT 句を含むクエリは、テンポラリテーブルを作成します。したがって、クエリ実行中に文字列が内部的に処理される方法が原因で、SageMaker 関数の結果が切り捨てられる場合があります。

SageMaker モデルトレーニング用のデータを Amazon S3 にエクスポートする (高度)

提供されているいくつかのアルゴリズムを使用して、Aurora 機械学習と SageMaker を使い始めることをお勧めします。また、チームのデータサイエンティストによって SQL コードで使用できる SageMaker エンドポイントを提供することをお勧めします。以下に、独自の Amazon S3 バケットを独自の SageMaker モデルと Aurora MySQL DB クラスターで使用する方法についての最低限必要な情報を示します。

機械学習は、トレーニングと推論という 2 つの主要な手順で構成されます。SageMaker モデルをトレーニングするには、データを Amazon S3 バケットにエクスポートします。Amazon S3 バケットは、モデルをデプロイする前にトレーニングする目的で SageMaker ノートブックインスタンスによって使用されます。SELECT INTO OUTFILE S3 ステートメントを使用すると、Aurora MySQL DB クラスターからデータをクエリし、Amazon S3 バケットに保存されているテキストファイルに直接保存できます。これにより、ノートブックインスタンスは、トレーニングのために Amazon S3 バケットからデータを消費します。

Aurora 機械学習は、Aurora MySQL の既存の SELECT INTO OUTFILE 構文を拡張して、データを CSV 形式にエクスポートします。生成された CSV ファイルは、トレーニング目的で、この形式を必要とするモデルで直接使用できます。

```
SELECT * INTO OUTFILE S3 's3_uri' [FORMAT {CSV|TEXT} [HEADER]] FROM table_name;
```

このエクステンションは、スタンダードの CSV 形式をサポートしています。

- TEXT 形式は、既存の MySQL エクスポート形式と同じです。これがデフォルトの形式です。
- CSV 形式は、RFC-4180 の仕様に従って新しく導入された形式です。
- オプションのキーワード HEADER を指定すると、出力ファイルに 1 つのヘッダー行が含まれます。ヘッダー行のラベルは、SELECT ステートメントの列名に対応します。
- キーワード CSV および HEADER を引き続き識別子として使用できます。

SELECT INTO の拡張構文と文法は次のとおりです。

```
INTO OUTFILE S3 's3_uri'
[CHARACTER SET charset_name]
[FORMAT {CSV|TEXT} [HEADER]]
[{FIELDS | COLUMNS}
  [TERMINATED BY 'string']
  [[OPTIONALLY] ENCLOSED BY 'char']
  [ESCAPED BY 'char']
]
[LINES
  [STARTING BY 'string']
  [TERMINATED BY 'string']
]
```

Aurora MySQL で Aurora 機械学習を使用した場合のパフォーマンスに関する考慮事項

Amazon Bedrock、Amazon Comprehend と SageMaker サービスは、Aurora 機械学習の関数によって呼び出された際には、ほとんどの作業を行います。つまり、これらのリソースを必要に応じて個別にスケーリングできることになります。Aurora MySQL DB クラスターでは、関数呼び出しを最大限効率的に行うことができます。以下に、Aurora 機械学習を使用する際に注意すべきパフォーマンスに関する考慮事項をいくつか示します。

モデルとプロンプト

Amazon Bedrock を使用する場合のパフォーマンスは、使用するモデルとプロンプトに大きく依存します。ユースケースに最適なモデルとプロンプトを選択します。

クエリキャッシュ

Aurora MySQL クエリキャッシュは、Aurora 機械学習関数では機能しません。Aurora MySQL は、Aurora 機械学習関数を呼び出す SQL ステートメントのクエリ結果を、クエリキャッシュに保存しません。

Aurora 機械学習関数呼び出しのバッチ最適化

Aurora クラスターの影響を受ける可能性がある Aurora 機械学習のパフォーマンスの主な側面は、Aurora 機械学習ストアド関数を呼び出すためのバッチモード設定です。通常、機械学習関数はかなりのオーバーヘッドを必要とするため、行ごとに外部サービスを個別に呼び出すことは現実的ではありません。Aurora 機械学習では、多くの行の外部 Aurora 機械学習サービスへの呼び出しを 1 つのバッチに結合することで、このオーバーヘッドを最小限に抑えることができます。Aurora 機械学習は、すべての入力行に対する応答を受け取り、その応答を一度に 1 行ずつ実行中のクエリに返送します。この最適化により、結果を変更せずに Aurora クエリのスループットとレイテンシーが改善されます。

SageMaker エンドポイントに接続されている Aurora ストアド関数を作成するときに、バッチサイズパラメータを定義します。このパラメータは、基礎となる SageMaker の呼び出しごとに転送される行数に影響します。多数の行を処理するクエリの場合、行ごとに個別の SageMaker 呼び出しを行うオーバーヘッドが大規模化する可能性があります。ストアドプロシージャによって処理されるデータセットが大きいほど、バッチサイズを大きくすることができます。

バッチモードの最適化を SageMaker 関数に適用できる場合は、`EXPLAIN PLAN` ステートメントによって生成されたクエリプランを確認することで判断できます。この場合、実行計画の `extra` 列には `Batched machine learning` が含まれます。次の例は、バッチモードを使用する SageMaker 関数の呼び出しを示しています。

```
mysql> CREATE FUNCTION anomaly_score(val real) returns real alias
 aws_sagemaker_invoke_endpoint endpoint name 'my-rcf-model-20191126';
Query OK, 0 rows affected (0.01 sec)

mysql> explain select timestamp, value, anomaly_score(value) from nyc_taxi;
+----+-------------+----------+------------+------+---------------+------+---------
+------+------+----------+--------------------------+
| id | select_type | table    | partitions | type | possible_keys | key  | key_len |
 ref  | rows | filtered | Extra                    |
+----+-------------+----------+------------+------+---------------+------+---------
+------+------+----------+--------------------------+
|  1 | SIMPLE      | nyc_taxi | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL |   48 |   100.00 | Batched machine learning |
```

```
+----+-------------+----------+------------+------+---------------+------+---------
+------+------+----------+--------------------------+
1 row in set, 1 warning (0.01 sec)
```

組み込みの Amazon Comprehend 関数の 1 つを呼び出す場合、バッチサイズを制御するには、オプションの max_batch_size パラメータを指定します。このパラメータは、各バッチで処理される input_text 値の最大数を制限します。複数の項目を一度に送信することにより、Aurora と Amazon Comprehend 間の往復回数を減らします。バッチサイズの制限は、LIMIT 句を使用したクエリなどで役立ちます。max_batch_size に小さい値を使用することで、入力テキストより Amazon Comprehend を呼び出す回数が増えるのを回避できます。

Aurora 機械学習関数を評価するためのバッチ最適化は、以下の場合に適用されます。

- 選択リストまたは SELECT ステートメントの WHERE 句に含まれる関数呼び出し。
- INSERT および REPLACE ステートメントの VALUES リストに含まれる関数呼び出し。
- UPDATE ステートメントの SET 値の SageMaker 関数。

```
INSERT INTO MY_TABLE (col1, col2, col3) VALUES
  (ML_FUNC(1), ML_FUNC(2), ML_FUNC(3)),
  (ML_FUNC(4), ML_FUNC(5), ML_FUNC(6));
UPDATE MY_TABLE SET col1 = ML_FUNC(col2), SET col3 = ML_FUNC(col4) WHERE ...;
```

Aurora 機械学習のモニタリング

次の例に示すように、複数のグローバル変数をクエリすることで、Aurora 機械学習のバッチオペレーションをモニタリングできます。

```
show status like 'Aurora_ml%';
```

このステータス変数をリセットするには、FLUSH STATUS ステートメントを使用します。したがって、可変の最後のリセット以降のすべての数値は、合計、平均などを表します。

Aurora_ml_logical_request_cnt

前回のステータスリセット以降、DB インスタンスが Aurora 機械学習サービスに送信されると評価した論理リクエストの数。バッチ処理が使用されているかどうかによっては、この値が Aurora_ml_actual_request_cnt より高くなることがあります。

`Aurora_ml_logical_response_cnt`

DB インスタンスのユーザーが実行するすべてのクエリで、Aurora MySQL が Aurora 機械学習サービスから受け取るレスポンスの集計カウント。

`Aurora_ml_actual_request_cnt`

DB インスタンスのユーザーによって実行されたすべてのクエリで、Aurora MySQL が Aurora 機械学習サービスに対して行ったリクエストの集計カウント。

`Aurora_ml_actual_response_cnt`

DB インスタンスのユーザーが実行するすべてのクエリで、Aurora MySQL が Aurora 機械学習サービスから受け取るレスポンスの集計カウント。

`Aurora_ml_cache_hit_cnt`

DB インスタンスのユーザーが実行したすべてのクエリで、Aurora MySQL が Aurora 機械学習サービスから受け取る内部キャッシュヒットの集計カウント。

`Aurora_ml_retry_request_cnt`

前回のステータスリセット以降、DB インスタンスが Aurora 機械学習サービスに送信した再試行リクエストの数。

`Aurora_ml_single_request_cnt`

DB インスタンスのユーザーが実行するすべてのクエリで、非バッチモードで評価される Aurora 機械学習の関数の集計カウント。

Aurora 機械学習機能から呼び出される SageMaker オペレーションのパフォーマンスのモニタリングについては、Amazon SageMaker のモニタリングを参照してください。

Aurora PostgreSQL で Amazon Aurora 機械学習を使用する

Aurora PostgreSQL DB クラスターで Amazon Aurora 機械学習を使用することで、必要に応じて Amazon Comprehend、Amazon SageMaker または Amazon Bedrock のいずれかを使用できます。これらのサービスが特定の機械学習のユースケースをサポートしています。

Aurora 機械学習は、特定の AWS リージョン と Aurora PostgreSQL の一部のバージョンでのみサポートされています。Aurora 機械学習を設定する前に、Aurora PostgreSQL で利用できるバージョ

ンとリージョンを確認してください。詳細については、「Aurora PostgreSQL を使用した Aurora Machine Learning」を参照してください。

トピック

- Aurora 機械学習 を Aurora PostgreSQL で使用するための要件
- Aurora PostgreSQL で Aurora 機械学習を使用する場合にサポートされている機能と制限事項
- Aurora 機械学習を使用するように PostgreSQL DB クラスターを設定する
- Aurora PostgreSQL DB クラスターでの Amazon Bedrock の使用
- Aurora PostgreSQL DB クラスターで Amazon Comprehend を使用する
- Aurora PostgreSQL DB クラスターで SageMaker を使用する
- SageMaker モデルトレーニング用のデータを Amazon S3 にエクスポートする (高度)
- Aurora PostgreSQL で Aurora 機械学習を使用した場合のパフォーマンスに関する考慮事項
- Aurora 機械学習のモニタリング

Aurora 機械学習 を Aurora PostgreSQL で使用するための要件

AWS 機械学習サービスは、お客様の本番環境で設定し、実行するマネージドサービスです。Aurora 機械学習は、Amazon Comprehend、SageMaker および Amazon Bedrock の統合をサポートしています。Aurora PostgreSQL DB クラスターを設定して Aurora 機械学習を使用開始する前に、次の要件と前提条件を理解していることを確認してください。

- Amazon Comprehend、SageMaker および Amazon Bedrock のサービスは、お使いの Aurora PostgreSQL DB クラスターと同じ AWS リージョン で実行する必要があります。別のリージョンにある Aurora PostgreSQL DB クラスターから、Amazon Comprehend、SageMaker または および Amazon Bedrock サービスは使用できません。
- Aurora PostgreSQL DB クラスターが Amazon VPC サービスに基づく Amazon Comprehend や SageMaker サービスとは異なる仮想パブリッククラウド (VPC) にある場合、VPC のセキュリティグループが対象の Aurora 機械学習サービスへのアウトバウンド接続を許可する必要があります。詳細については、「Amazon Aurora MySQL から他の AWS のサービスへのネットワーク通信の有効化」を参照してください。
- SageMaker では、推論に使用する機械学習コンポーネントが設定され、使用できる状態になっている必要があります。Aurora PostgreSQL DB クラスターの設定プロセス時に、SageMaker エンドポイントの Amazon リソースネーム (ARN) を用意する必要があります。SageMaker と連携してモデルを準備し、その他のタスクを処理するのは、チームのデータサイエンティストが最も適任だ

と考えられます。Amazon SageMaker を使い始めるには、「Amazon SageMaker の使用を開始する」を参照してください。推論とエンドポイントの詳細については、「リアルタイム推論」を参照してください。

- Amazon Bedrock の場合、Aurora PostgreSQL DB クラスターの設定プロセス中に推論に使用する Bedrock モデルのモデル ID が必要です。チームのデータサイエンティストは、Bedrock と連携して使用するモデルを決定し、必要に応じて微調整して、他のタスクを処理するのが最も最適と考えられます。Amazon Bedrock の使用を開始するには、「Bedrock のセットアップ方法」を参照してください。
- Amazon Bedrock ユーザーがモデルを使用するには、まずそのモデルへのアクセスをリクエストする必要があります。テキスト、チャット、イメージを生成するための新しいモデルを追加する場合は、Amazon Bedrock のモデルへのアクセスをリクエストする必要があります。詳細については、「モデルアクセス」を参照してください。

Aurora PostgreSQL で Aurora 機械学習を使用する場合にサポートされている機能と制限事項

Aurora 機械学習は、`ContentType` の `text/csv` 値を介して、カンマ区切り値 (CSV) 形式を読み書きできる SageMaker エンドポイントをサポートしています。現在、この形式を受け入れている組み込みの SageMaker アルゴリズムは以下のとおりです。

- 線形学習
- ランダムカットフォレスト
- XGBoost

これらのアルゴリズムの詳細については、「Amazon SageMaker デベロッパーガイド」の「アルゴリズムの選択」を参照してください。

Amazon Bedrock を Aurora 機械学習と合わせて使用する場合、次の制限事項が適用されます。

- ユーザー定義関数 (UDF) は、Amazon Bedrock を操作するためのネイティブな方法を提供します。UDF には特定のリクエストまたはレスポンス要件がないため、どのモデルも使用できます。
- UDF を使用して必要なワークフローを構築できます。例えば、`pg_cron` などの基本プリミティブを組み合わせて、クエリの実行、データの取得、推論の生成、テーブルへの書き込みを行ってクエリを直接処理できます。
- UDF はバッチ呼び出しまたは並列呼び出しをサポートしていません。

- Aurora 機械学習の拡張機能は、ベクトルインターフェイスをサポートしていません。拡張機能の一部として、関数を使用してモデルのレスポンスの埋め込みを `float8[]` 形式で出力し、それらの埋め込みを Aurora に保存できます。`float8[]` の使用の詳細については、「Aurora PostgreSQL DB クラスターでの Amazon Bedrock の使用」を参照してください。

Aurora 機械学習を使用するように PostgreSQL DB クラスターを設定する

Aurora 機械学習を Aurora PostgreSQL DB クラスターと連携させるには、使用するサービスごとに AWS Identity and Access Management (IAM) ロールを作成する必要があります。IAM ロールによって、Aurora PostgreSQL DB クラスターがクラスターに代わって Aurora 機械学習サービスを使用することができます。また、Aurora 機械学習拡張機能のインストールが必要です。以下のトピックでは、これらの Aurora 機械学習サービスごとの設定手順を参照できます。

トピック

- Amazon Bedrock を使用するように Aurora PostgreSQL を設定する
- Amazon Comprehend を使用するように Aurora PostgreSQL を設定する
- Amazon SageMaker を使用するように Aurora PostgreSQL を設定する
  - SageMaker で Amazon S3 を使用するように Aurora PostgreSQL を設定する (高度)
- Aurora 機械学習拡張拡張のインストール

Amazon Bedrock を使用するように Aurora PostgreSQL を設定する

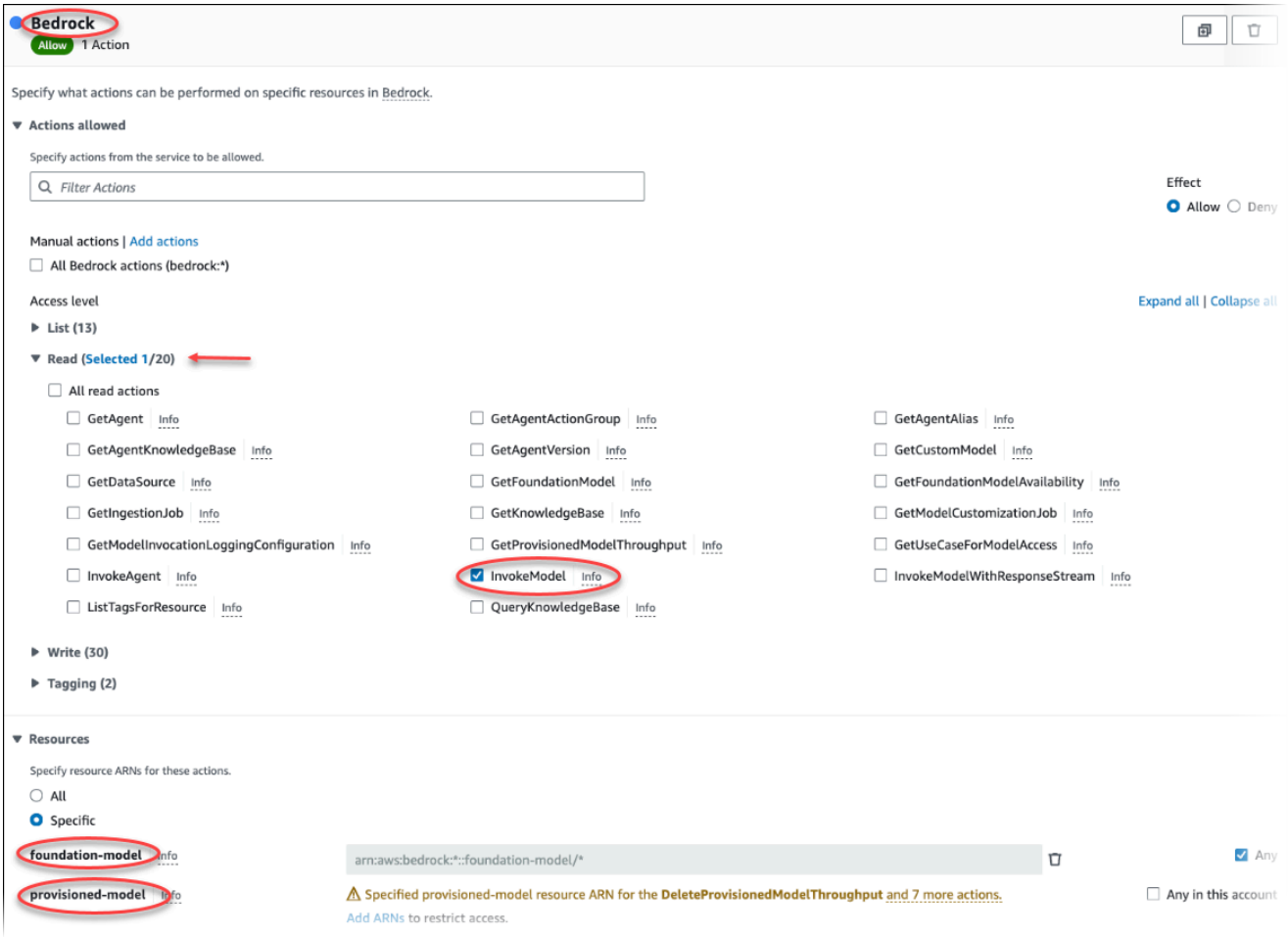
以下の手順では、まず、クラスターに代わって Amazon Bedrock を使用するアクセス権限を Aurora PostgreSQL に付与する IAM ロールとポリシーを作成します。次に、Aurora PostgreSQL DB クラスターが Amazon Bedrock の操作に使用する IAM ロールにポリシーをアタッチします。説明を簡単にするため、この手順では AWS Management Console を使用してすべてのタスクを完了します。

Amazon Bedrock を使用するように Aurora PostgreSQL DB クラスターを設定には

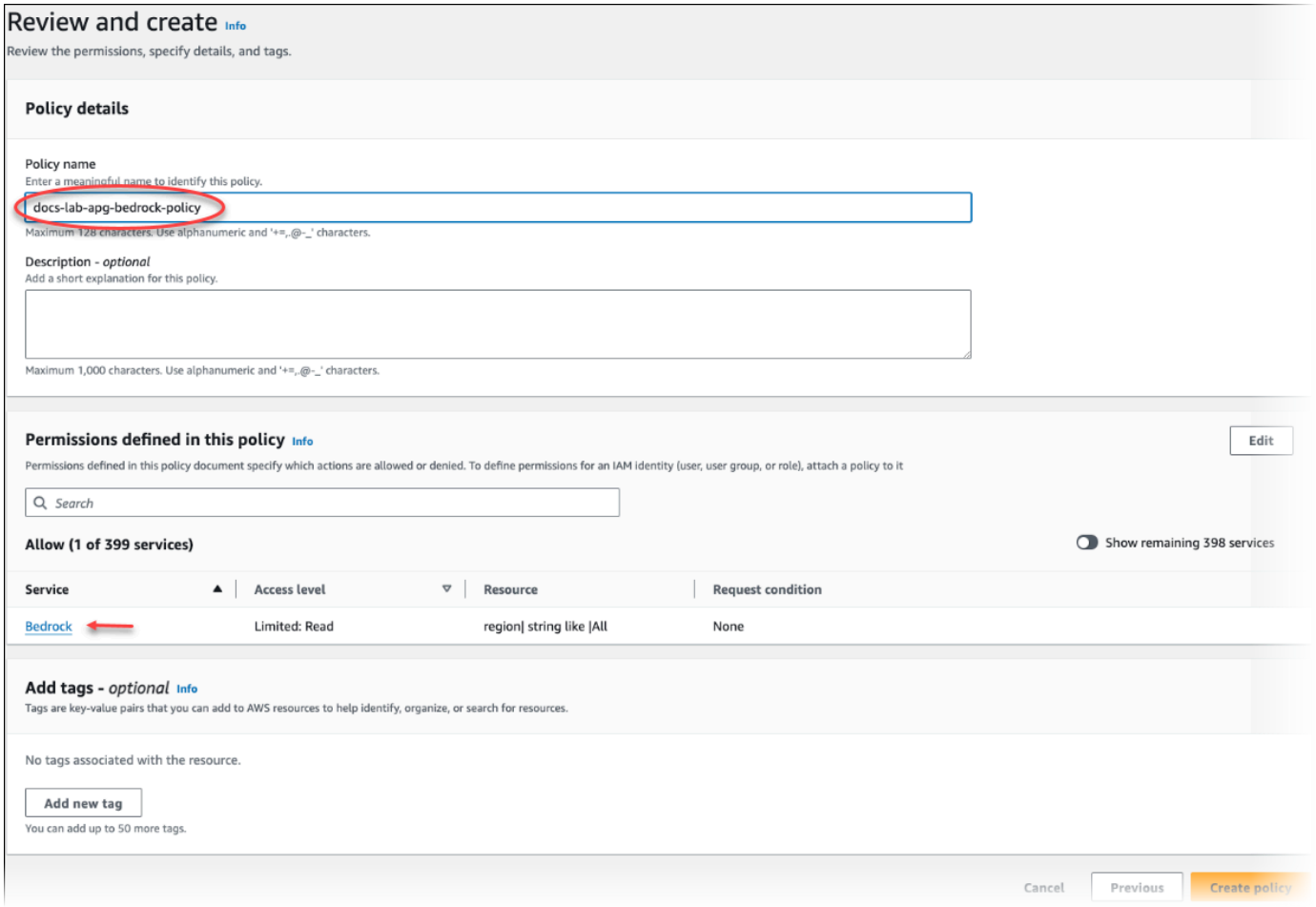
1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
3. AWS Identity and Access Management (IAM) コンソールメニューで [Policies] (ポリシー) ([Access management] (アクセス管理) の下) を選択します。

a. [Create policy] を選択します。ビジュアルエディタのページで [サービス] を選択し、[サービスの選択] フィールドに [Bedrock] と入力します。読み取りアクセスレベルを拡張します。Amazon Bedrock の読み取り設定から InvokeModel を選択します。

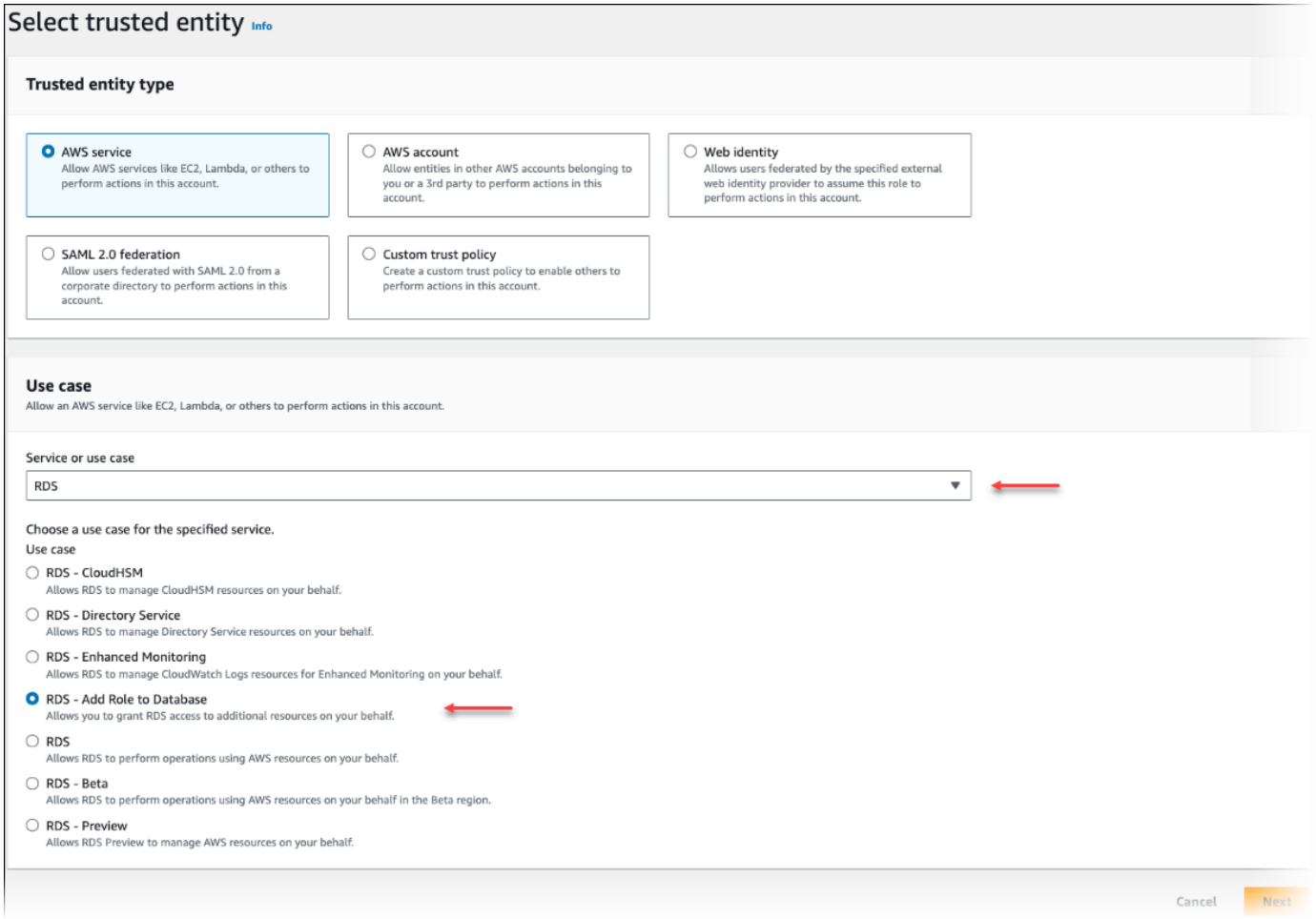
b. ポリシーを介して読み取りアクセスを許可する [基礎/プロビジョニング済み] モデルを選択します。



4. [Next: Tags] (次へ: タグ) を選択し、任意のタグを定義します (これはオプションです)。[次へ: レビュー] を選択します。画像のように ポリシーの名前と説明を入力します。

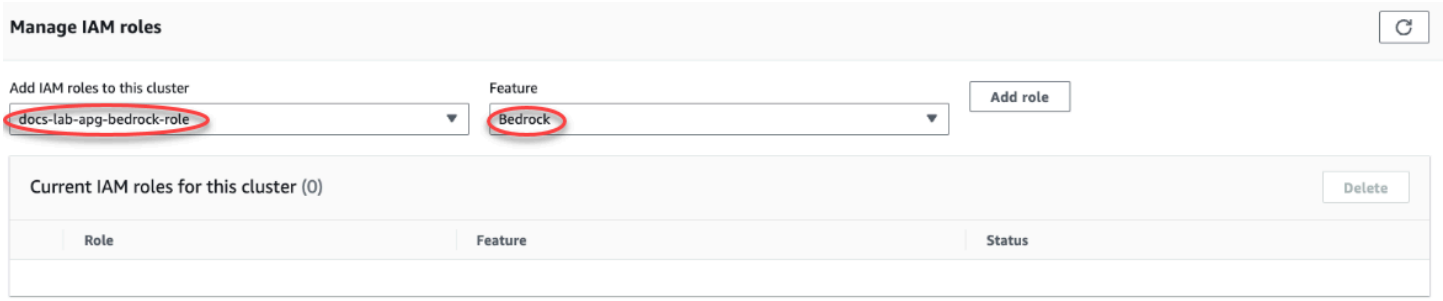


5. [Create policy] を選択します。ポリシーが保存されると、コンソールにアラートが表示されます。これはポリシーのリストで確認できます。
6. IAM コンソールで、[Roles] (ロール) ([Access management] (アクセス管理) の下) を選択します。
7. [ロールの作成] を選択します。
8. [Select trusted entity] (信頼されたエンティティの選択) ページで、[AWS service] (AWS サービス) タイルを選択し、[RDS] を選択してセレクタを開きます。
9. [RDS – ロールをデータベースに追加する] を選択します。



10. [Next] を選択します。[Add permissions] (権限の追加) ページで、前の手順で作成したポリシーをリストから検索して選択します。[Next] を選択します。
11. [Next: Review] (次へ: 確認)。IAM ロールの名前と説明を入力します。
12. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
13. Aurora PostgreSQL DB クラスターがある AWS リージョン に移動します。
14. ナビゲーションペインで、[データベース] を選択して、Bedrock で使用する Aurora PostgreSQL DB クラスターを選択します。
15. [Connectivity & security] (接続とセキュリティ) タブを選択し、ページをスクロールして [Manage IAM roles] (IAM ロールの管理) セクションを見つけます。[Add IAM roles to this cluster] (IAM ロールをこのクラスターに追加する) セレクタから、前の手順で作成したロールを選択します。[機能] セレクタで [Bedrock] を選択し、[ロールを追加] を選択します。

ロール (およびそのポリシー) は、Aurora PostgreSQL DB クラスターに関連付けられています。プロセスが完了すると、次に示すように、このクラスターの現在の IAM ロールリストにロールが表示されます。



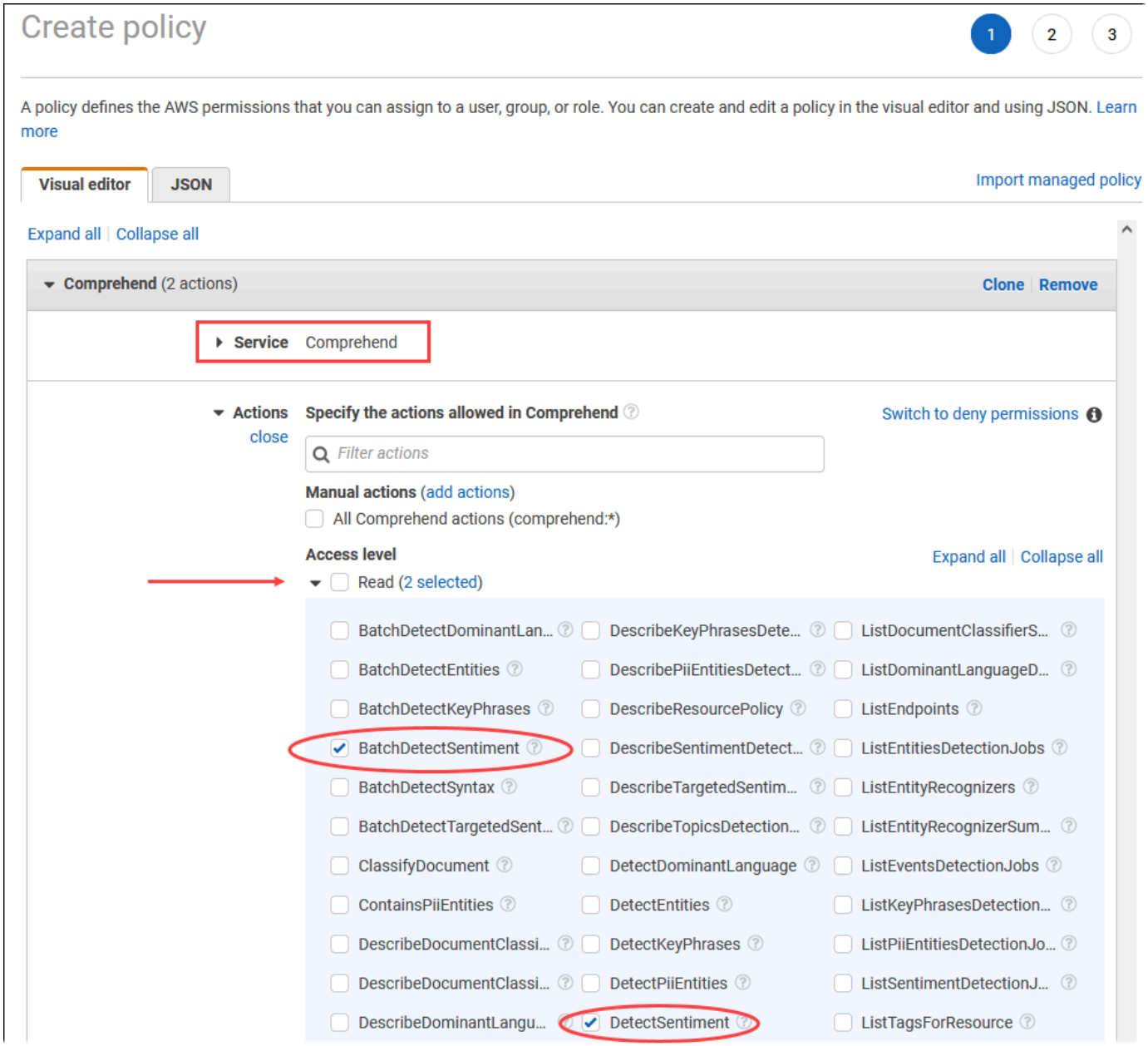
Amazon Bedrock の IAM 設定が完了しました。「Aurora 機械学習拡張拡張のインストール」の説明のように、Aurora PostgreSQL を Aurora 機械学習と連携するように設定を継続するには、拡張機能をインストールします

Amazon Comprehend を使用するように Aurora PostgreSQL を設定する

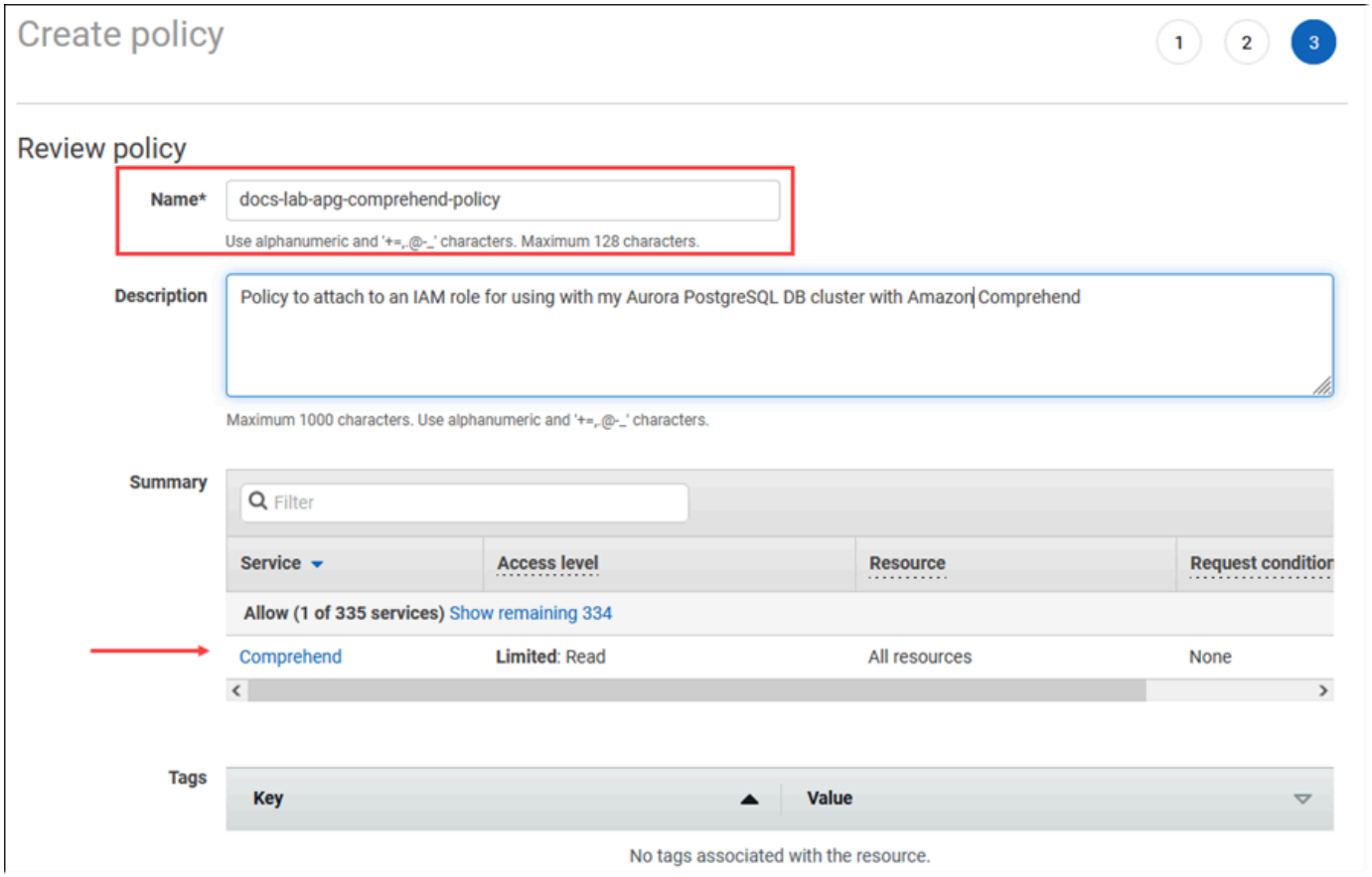
以下の手順では、まず、クラスターに代わって Amazon Comprehend を使用するアクセス権限を Aurora PostgreSQL に付与する IAM ロールとポリシーを作成します。次に、Aurora PostgreSQL DB クラスターが Amazon Comprehend と連携するために使用する IAM ロールにポリシーをアタッチします。わかりやすくするために、この手順では AWS Management Console を使用してすべてのタスクを完了します。

Amazon Comprehend を使用するように Aurora PostgreSQL DB クラスターを設定には

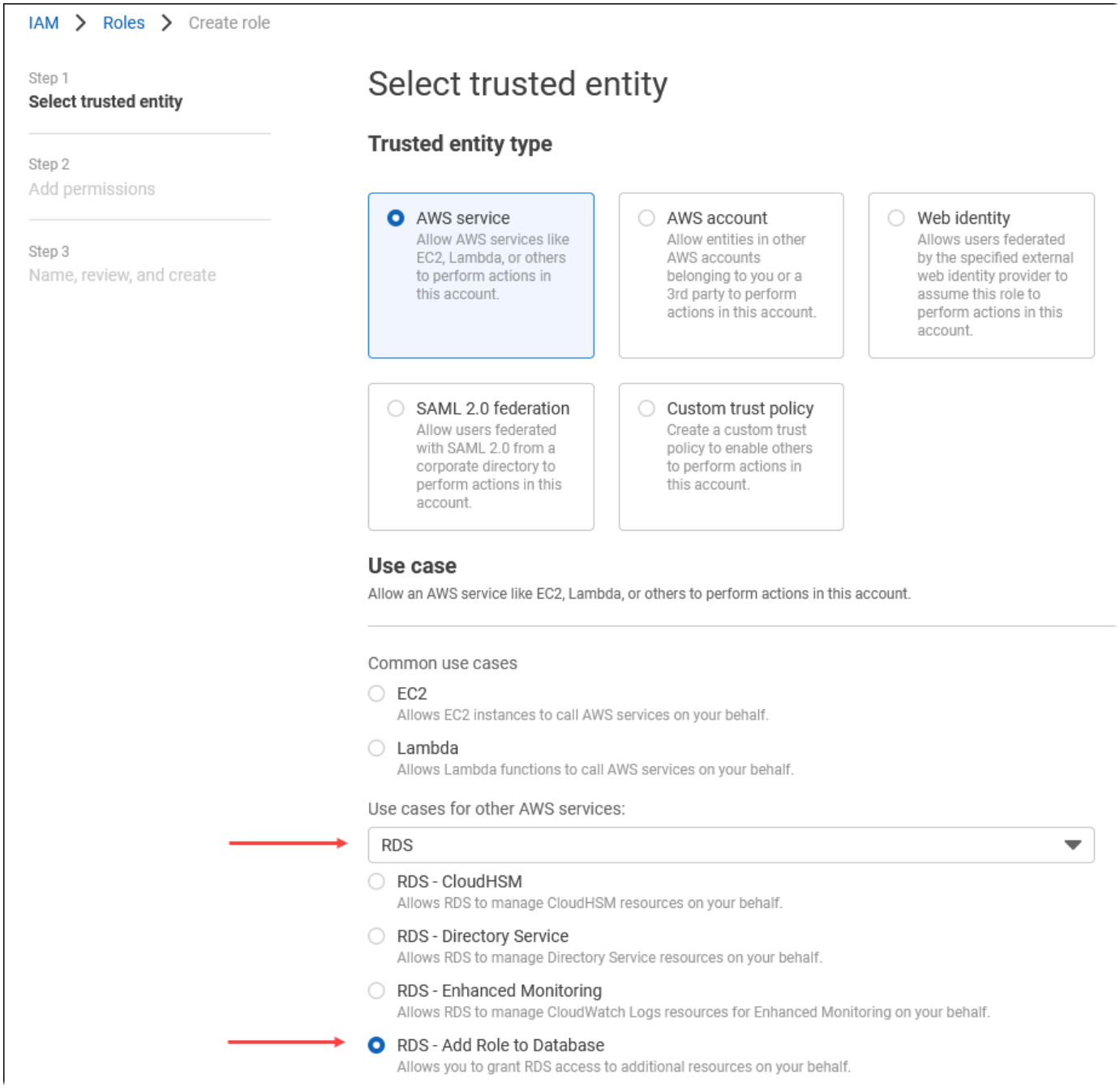
1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
3. AWS Identity and Access Management (IAM) コンソールメニューで [Policies] (ポリシー) ([Access management] (アクセス管理) の下) を選択します。



4. [Create policy] を選択します。ビジュアルエディタのページで[Service] (サービス) を選択し、[Select a service] (サービス選択) フィールドに [Comprehend] と入力します。読み取りアクセスレベルを拡張します。Amazon Comprehend の読み取り設定から BatchDetectSentiment と DetectSentiment を選択します。
5. [Next: Tags] (次へ: タグ) を選択し、任意のタグを定義します (これはオプションです)。[次へ: レビュー] を選択します。画像のように ポリシーの名前と説明を入力します。



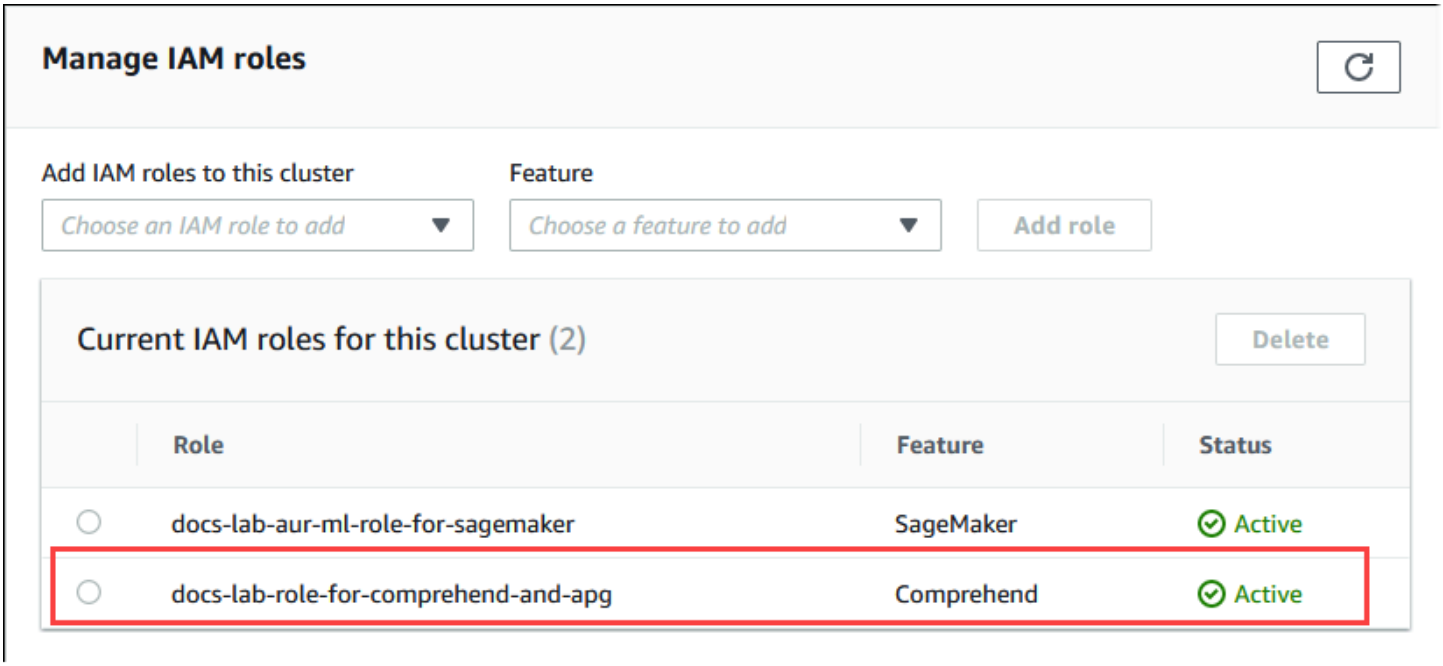
6. [Create policy] を選択します。ポリシーが保存されると、コンソールにアラートが表示されます。これはポリシーのリストで確認できます。
7. IAM コンソールで、[Roles] (ロール) ([Access management] (アクセス管理) の下) を選択します。
8. [ロールの作成] を選択します。
9. [Select trusted entity] (信頼されたエンティティの選択) ページで、[AWS service] (AWS サービス) タイルを選択し、[RDS] を選択してセレクタを開きます。
10. [RDS – Add Role to Database] (RDS – ロールをデータベースに追加する) を選択します。



11. [Next] を選択します。[Add permissions] (権限の追加) ページで、前の手順で作成したポリシーをリストから検索して選択します。[Next] (次へ) を選択します。
12. [Next: Review] (次へ: 確認)。IAM ロールの名前と説明を入力します。
13. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
14. Aurora PostgreSQL DB クラスターがある AWS リージョン に移動します。
15. ナビゲーションペインで、[Databases] (データベース) を選択して、Amazon Comprehend で使用する Aurora PostgreSQL DB クラスターを選択します。

16. [Connectivity & security] (接続とセキュリティ) タブを選択し、ページをスクロールして [Manage IAM roles] (IAM ロールの管理) セクションを見つけます。[Add IAM roles to this cluster] (IAM ロールをこのクラスターに追加する) セレクタから、前の手順で作成したロールを選択します。[機能] セレクタで [Comprehend] を選択し、[ロールを追加] を選択します。

ロール (およびそのポリシー) は、Aurora PostgreSQL DB クラスターに関連付けられています。プロセスが完了すると、次に示すように、このクラスターの現在の IAM ロールリストにロールが表示されます。



Amazon Comprehend の IAM 設定が完了しました。「Aurora 機械学習拡張拡張のインストール」の説明のように、Aurora PostgreSQL を Aurora 機械学習と連携するように設定を継続するには、拡張機能をインストールします

Amazon SageMaker を使用するように Aurora PostgreSQL を設定する

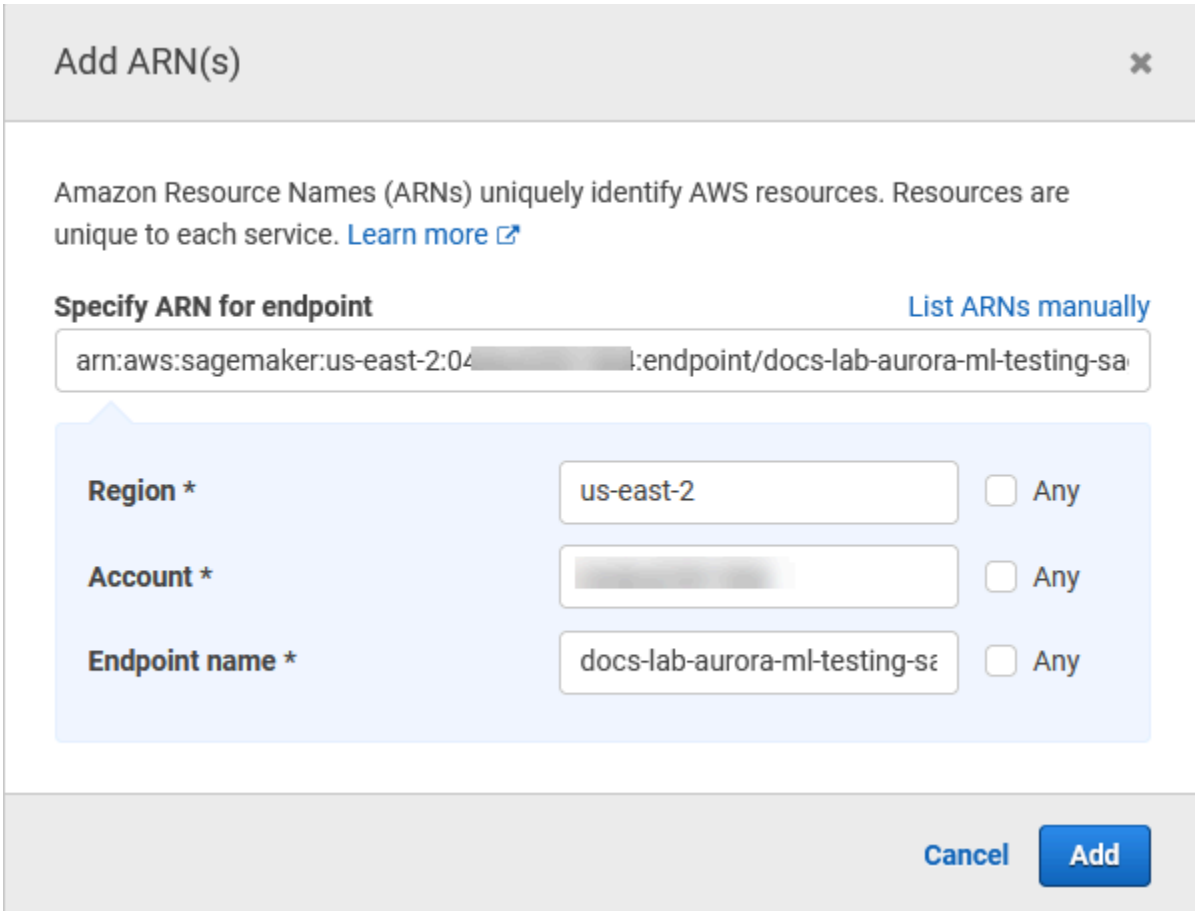
Aurora PostgreSQL DB クラスターの IAM ポリシーとロールを作成する前に、SageMaker モデルの設定とエンドポイントを利用できるようにする必要があります。

SageMaker を使用するように Aurora PostgreSQL DB クラスターを設定するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. AWS Identity and Access Management (IAM) コンソールメニューで [Policies] (ポリシー) ([Access management] (アクセス管理) の下) を選択し、次に [Create policy] (ポリシーの作成) を

選択します。ビジュアルエディタで、[Service] (サービス) に [SageMaker] を選択します。アクションについては、読み取りセレクタ (アクセスレベルの下) を開き、[InvokeEndpoint] を選択します。これを実行すると、警告アイコンが表示されます。

3. リソースセレクタを開き、InvokeEndpoint アクションのエンドポイントリソース ARN の指定で、[Add ARN to restrict access] (アクセスを制限する ARN を追加する) リンクを選択します。
4. SageMaker リソースの AWS リージョン とエンドポイントの名前を入力します。AWS アカウントは事前に入力されています。



5. [Add] (追加) を選択して保存します。[Next: Tags] (次へ: タグ) と [Next: Review] (次へ: 確認) を選択して、ポリシー作成プロセスの最後のページに進みます。
6. ポリシーの名前と説明を入力して、[Create policy] (ポリシーの作成) を選択します。ポリシーが作成され、ポリシーのリストに追加されます。これが発生すると、コンソールにアラートが表示されます。
7. IAM コンソールで、[Roles] (ロール) を選択します。
8. [ロールの作成] を選択します。
9. [Select trusted entity] (信頼されたエンティティの選択) ページで、[AWS service] (AWS サービス) タイルを選択し、[RDS] を選択してセレクタを開きます。

10. [RDS – Add Role to Database] (RDS – ロールをデータベースに追加する) を選択します。
11. [Next] を選択します。[Add permissions] (権限の追加) ページで、前の手順で作成したポリシーをリストから検索して選択します。[Next] (次へ) を選択します。
12. [Next: Review] (次へ: 確認)。IAM ロールの名前と説明を入力します。
13. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
14. Aurora PostgreSQL DB クラスターがある AWS リージョン に移動します。
15. ナビゲーションペインで、[Databases] (データベース) を選択して、SageMaker で使用する Aurora PostgreSQL DB クラスターを選択します。
16. [Connectivity & security] (接続とセキュリティ) タブを選択し、ページをスクロールして [Manage IAM roles] (IAM ロールの管理) セクションを見つけます。[Add IAM roles to this cluster] (IAM ロールをこのクラスターに追加する) セレクタから、前の手順で作成したロールを選択します。[Feature] (機能) セレクタで [SageMaker] を選択し、[Add role] (役割を追加) を選択します。

ロール (およびそのポリシー) は、Aurora PostgreSQL DB クラスターに関連付けられています。プロセスが完了すると、このクラスターの現在の IAM ロールリストにロールが表示されます。

SageMaker の IAM 設定が完了しました。「Aurora 機械学習拡張拡張のインストール」の説明のように、Aurora PostgreSQL を Aurora 機械学習と連携するように設定を継続するには、拡張機能をインストールします。

SageMaker で Amazon S3 を使用するように Aurora PostgreSQL を設定する (高度)

SageMaker が提供する構築済みコンポーネントを使用せず、独自のモデルで SageMaker を使用するには、Aurora PostgreSQL DB クラスターが使用する Amazon Simple Storage Service (Amazon S3) バケットを設定する必要があります。これは高度なトピックであり、この「Amazon Aurora ユーザーガイド」にはすべては記載されていません。一般的な手順は、SageMaker のサポートを統合する場合と同じで、次のようになります。

1. Amazon S3 の IAM ポリシーとロールを作成します。
2. Aurora PostgreSQL DB クラスターの [Connectivity & security] (接続とセキュリティ) タブに、IAM ロールと Amazon S3 のインポートまたはエクスポートを機能として追加します。
3. Aurora DB クラスターのパラメータグループに、ロールの ARN を追加します。

基本的な使用に関する情報については、「SageMaker モデルトレーニング用のデータを Amazon S3 にエクスポートする (高度)」を参照してください。

Aurora 機械学習拡張拡張のインストール

Aurora 機械学習の拡張機能 `aws_ml 1.0` には、Amazon Comprehend の呼び出しに使用できる 2 つの関数があり、SageMaker サービスと `aws_ml 2.0` には、Amazon Bedrock サービスの呼び出しに使用できる 2 つの追加関数があります。Aurora PostgreSQL DB クラスターにこれらの拡張機能をインストールすると、その機能の管理者ロールも作成されます。

> **Note**
>
> これらの機能の使用については、「Aurora 機械学習を使用するように PostgreSQL DB クラスターを設定する」の説明のように、Aurora 機械学習サービス ((Amazon Comprehend、SageMaker、Amazon Bedrock) の IAM 設定完了の有無で異なります。

- aws_comprehend.detect_sentiment — この関数を使用して、Aurora PostgreSQL DB クラスターのデータベースに保存されているテキストに感情分析を適用します。
- aws_sagemaker.invoke_endpoint — SQL コードでこの関数を使用して、クラスターから SageMaker エンドポイントと通信します。
- aws_bedrock.invoke_model – SQL コードでこの関数を使用して、クラスターから Bedrock モデルと通信します。この関数のレスポンスは TEXT の形式になるため、モデルが JSON 本文の形式で応答する場合、この関数の出力は文字列の形式でエンドユーザーに中継されます。
- aws_bedrock.invoke_model_get_embeddings – SQL コードでこの関数を使用して、JSON レスポンス内で出力埋め込みを返す Bedrock Models を呼び出します。これは、json-key に直接関連付けられた埋め込みを抽出して、セルフマネージドワークフローでのレスポンスを効率化する場合に利用できます。

Aurora 機械学習拡張機能を使用するように PostgreSQL DB クラスターにインストールするには

- `psql` を使用して、Aurora PostgreSQL DB クラスターのライターインスタンスに接続します。`aws_ml` 拡張機能をインストールする対象のデータベースに接続します。

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=labdb
```

```
labdb=> CREATE EXTENSION IF NOT EXISTS aws_ml CASCADE;
NOTICE:  installing required extension "aws_commons"
```

```
CREATE EXTENSION
labdb=>
```

また、aws_ml 拡張機能をインストールすると、次のように aws_ml 管理者ロールと 2 つの新しいスキーマが作成されます。

- aws_comprehend — Amazon Comprehend サービスのスキーマと detect_sentiment 関数のソース (aws_comprehend.detect_sentiment)。
- aws_sagemaker — SageMaker サービスのスキーマと invoke_endpoint 関数のソース (aws_sagemaker.invoke_endpoint)。
- aws_bedrock — Amazon Bedrock サービスのスキーマと invoke_model(aws_bedrock.invoke_model) および invoke_model_get_embeddings(aws_bedrock.invoke_model_get_embeddings) 関数のソース。

この rds_superuser ロールには aws_ml 管理者ロールが付与され、これら 2 つの Aurora 機械学習スキーマの OWNER で構成されています。他のデータベースユーザーが Aurora 機械学習機能にアクセスできるようにするには、rds_superuser では Aurora 機械学習機能の EXECUTE 権限を付与する必要があります。デフォルトでは、2 つの Aurora 機械学習スキーマの関数に対して、PUBLIC から EXECUTE 権限が取り消されます。

マルチテナントデータベース構成では、保護する特定の Aurora 機械学習スキーマで REVOKE USAGE を使用することで、テナントが Aurora 機械学習関数にアクセスするのを防ぐことができます。

Aurora PostgreSQL DB クラスターでの Amazon Bedrock の使用

Aurora PostgreSQL の場合、Aurora 機械学習では、テキストデータを扱うために次のような Amazon Bedrock 関数が用意されています。この機能は、aws_ml 2.0 拡張機能をインストールしてすべての設定手順を完了した後にのみ使用できます。詳細については、「Aurora 機械学習を使用するように PostgreSQL DB クラスターを設定する」を参照してください。

aws_bedrock.invoke_model

この関数は、JSON でフォーマットされたテキストを入力として受け取り、Amazon Bedrock でホストされているさまざまなモデルに対して処理し、モデルから JSON テキストレスポンスを返します。このレスポンスには、テキスト、画像、埋め込みが含まれる場合があります。関数のドキュメントの概要は次のとおりです。

```
aws_bedrock.invoke_model(
   IN model_id      varchar,
   IN content_type  text,
   IN accept_type   text,
   IN model_input   text,
   OUT model_output varchar)
```

この関数の入力と出力は次のとおりです。

- `model_id` - モデルの識別子。
- `content_type` - Bedrock のモデルへのリクエストのタイプ。
- `accept_type` - Bedrock のモデルに期待されるレスポンスのタイプ。通常、ほとんどのモデルのアプリケーション/JSON。
- `model_input` - プロンプト。content_type で指定された形式のモデルへの特定の入力セット。モデルが受け入れるリクエスト形式/構造の詳細については、「基盤モデルの推論パラメータ」を参照してください。
- `model_output` - Bedrock モデルのテキストとしての出力。

次の例は、invoke_model を使用して Bedrock の Anthropic Claude 2 モデルを呼び出す方法を示しています。

Example 例: Amazon Bedrock の関数を使用する簡単なクエリ

```
SELECT aws_bedrock.invoke_model (
    model_id    := 'anthropic.claude-v2',
    content_type:= 'application/json',
    accept_type := 'application/json',
    model_input := '{"prompt": "\n\nHuman: You are a helpful assistant that answers
 questions directly and only using the information provided in the context below.
\nDescribe the answer
    in detail.\n\nContext: %s \n\nQuestion: %s \n
\nAssistant:","max_tokens_to_sample":4096,"temperature":0.5,"top_k":250,"top_p":0.5,"stop_seque
[]}'
);
```

aws_bedrock.invoke_model_get_embeddings

モデル出力は、場合によってはベクトル埋め込みを指すことがあります。レスポンスはモデルによって異なるため、別の関数 invoke_model_get_embeddings を使用できます。これは invoke_model とまったく同じように機能しますが、適切な json-key を指定して埋め込みを出力します。

```
aws_bedrock.invoke_model_get_embeddings(
   IN model_id      varchar,
   IN content_type  text,
   IN json_key      text,
   IN model_input   text,
   OUT model_output float8[])
```

この関数の入力と出力は次のとおりです。

- `model_id` - モデルの識別子。
- `content_type` - Bedrock のモデルへのリクエストのタイプ。ここで、accept_type はデフォルト値の `application/json` に設定されます。
- `model_input` - プロンプト。content_type で指定された形式のモデルへの特定の入力セット。モデルが受け入れるリクエスト形式/構造の詳細については、「基盤モデルの推論パラメータ」を参照してください。
- `json_key` – 埋め込みの抽出元のフィールドへの参照。これは、埋め込みモデルが変更されると異なる場合があります。
- `model_output` – 16 ビットの小数を持つ埋め込みの配列としての Bedrock モデルの出力。

次の例は、Titan Embeddings G1 - Text embedding モデルを使用して「PostgreSQL I/O monitoring views」というフレーズの埋め込みを生成する方法を示しています。

Example 例: Amazon Bedrock 関数を使用する簡単なクエリ

```
SELECT aws_bedrock.invoke_model_get_embeddings(
   model_id      := 'amazon.titan-embed-text-v1',
   content_type  := 'application/json',
   json_key      := 'embedding',
   model_input   := '{ "inputText": "PostgreSQL I/O monitoring views"}') AS embedding;
```

Aurora PostgreSQL DB クラスターで Amazon Comprehend を使用する

Aurora PostgreSQL の場合、Aurora 機械学習では、テキストデータを扱うために次のような Amazon Comprehend 関数が用意されています。この機能は、aws_ml 拡張機能をインストールしてすべての設定手順を完了した後にのみ使用できます。(詳しくは、「Aurora 機械学習を使用するように PostgreSQL DB クラスターを設定する」を参照してください。)

aws_comprehend.detect_sentiment

この関数は、テキストを入力として受け取り、そのテキストの感情的な姿勢がポジティブ、ネガティブ、中立、または混合のいずれであるかを評価します。この感情を、信頼度とともに評価用に出力します。関数のドキュメントの概要は次のとおりです。

```
aws_comprehend.detect_sentiment(
   IN input_text varchar,
   IN language_code varchar,
   IN max_rows_per_batch int,
   OUT sentiment varchar,
   OUT confidence real)
```

この関数の入力と出力は次のとおりです。

- input_text — 評価して、感情 (ネガティブ、ポジティブ、中立、混合) を割り当てるテキスト。
- language_code — 必要に応じて、地域サブタグ付きの 2 文字の ISO 639-1 識別子、または ISO 639-2 の 3 文字のコードを使用して識別される input_text 言語。例えば、en は英語のコード、zh は中国語 (簡体字) のコードです。詳細については、「Amazon Comprehend デベロッパーガイド」の「サポートしている言語」を参照してください。
- max_rows_per_batch – バッチモード処理のバッチあたりの最大行数。(詳しくは、「バッチモードと Aurora 機械学習関数の理解」を参照してください。)
- sentiment— 入力テキストの感情で、POSITIVE (ポジティブ)、NEGATIVE (ネガティブ)、NEUTRAL (中立) 、MIXED (混合) として識別されます。
- confidence – 指定された sentiment の値の精度に対する信頼度。値の範囲は 0.0 ~ 1.0 です。

以下では、この関数の使用方法について説明しています。

Example 例: Amazon Comprehend 関数を使用する簡単なクエリ

ここでは、この関数を呼び出し、サポートチームに対する顧客満足度を評価する簡単なクエリの例を示します。ヘルプをリクエストするたびに顧客からのフィードバックを保存するデータベーステーブル (support) があるとします。このクエリ例では、テーブルの feedback 列のテキストに aws_comprehend.detect_sentiment 関数を適用し、感情と、その感情に対する信頼度を出力します。また、このクエリは結果を降順で出力します。

```
SELECT feedback, s.sentiment,s.confidence
    FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
    ORDER BY s.confidence DESC;
 feedback                          | sentiment | confidence
 ------------------------------------------------------------+-----------+------------
 Thank you for the excellent customer support!               | POSITIVE  |   0.999771
 The latest version of this product stinks!                  | NEGATIVE  |   0.999184
 Your support team is just awesome! I am blown away.         | POSITIVE  |   0.997774
 Your product is too complex, but your support is great.     | MIXED     |   0.957958
 Your support tech helped me in fifteen minutes.             | POSITIVE  |   0.949491
 My problem was never resolved!                              | NEGATIVE  |   0.920644
 When will the new version of this product be released?      | NEUTRAL   |   0.902706
 I cannot stand that chatbot.                                | NEGATIVE  |   0.895219
 Your support tech talked down to me.                        | NEGATIVE  |   0.868598
 It took me way too long to get a real person.               | NEGATIVE  |   0.481805

 (10 rows)
```

テーブル行ごとに感情の検出が複数回発生しないように、分析の結果をマテリアライズできます。対象の行でこれを実行します。例えば、臨床医のメモは、フランス語 (fr) のものだけが感情検出関数を使用するように更新されています。

```
UPDATE clinician_notes
SET sentiment = (aws_comprehend.detect_sentiment (french_notes, 'fr')).sentiment,
    confidence = (aws_comprehend.detect_sentiment (french_notes, 'fr')).confidence
WHERE
    clinician_notes.french_notes IS NOT NULL AND
    LENGTH(TRIM(clinician_notes.french_notes)) > 0 AND
    clinician_notes.sentiment IS NULL;
```

関数呼び出しの最適化の詳細については、「Aurora PostgreSQL で Aurora 機械学習を使用した場合のパフォーマンスに関する考慮事項」を参照してください。

Aurora PostgreSQL DB クラスターで SageMaker を使用する

「Amazon SageMaker を使用するように Aurora PostgreSQL を設定する」の説明にあるように、SageMaker 環境を設定して Aurora PostgreSQL と統合すると、aws_sagemaker.invoke_endpoint 関数を使用してオペレーションを呼び出すことができます。aws_sagemaker.invoke_endpoint 関数は、同じ AWS リージョン のモデルエンドポイントにのみ接続します。複数の AWS リージョン にデータベースインスタンスのレプリカがある場合は、必ず各 SageMaker モデルをすべての AWS リージョン に設定し、デプロイします。

aws_sagemaker.invoke_endpoint への呼び出しは、Aurora PostgreSQL DB クラスターを SageMaker サービスに関連付けるために設定した IAM ロールと設定プロセス時に指定したエンドポイントを使用して認証されます。SageMaker モデルエンドポイントは個別のアカウントをスコープとし、パブリックではありません。endpoint_name の URL には、アカウント ID は含まれません。SageMaker は、データベースインスタンスの SageMaker IAM ロールによって提供される認証トークンからアカウント ID を決定します。

aws_sagemaker.invoke_endpoint

この関数は、SageMaker エンドポイントを入力として、バッチとして処理すべき行数を受け取ります。また、SageMaker モデルエンドポイントに必要な複数のパラメータを入力として受け取ります。この関数のリファレンスドキュメントは次のとおりです。

```
aws_sagemaker.invoke_endpoint(
  IN endpoint_name varchar,
  IN max_rows_per_batch int,
  VARIADIC model_input "any",
  OUT model_output varchar
  )
```

この関数の入力と出力は次のとおりです。

- endpoint_name – AWS リージョン に依存しないエンドポイントの URL。
- max_rows_per_batch – バッチモード処理のバッチあたりの最大行数。(詳しくは、「バッチモードと Aurora 機械学習関数の理解」を参照してください。)
- model_input – モデルの 1 つまたは複数の入力パラメータ。これらは、SageMaker モデルに必要なあらゆるデータ型にすることができます。PostgreSQL では、1 つの関数に対して最大 100 の入力パラメータを指定できます。配列のデータ型は 1 次元でなければなりませんが、SageMaker

モデルで期待される多数の要素を含めることができます。SageMaker モデルへの入力の数は、SageMaker の 6 MB のメッセージサイズ制限によってのみ制限されます。

- `model_output` – SageMaker モデルの、テキストとしての出力。

SageMaker モデルを呼び出すユーザー定義関数の作成

SageMaker の各モデルに対して `aws_sagemaker.invoke_endpoint` を呼び出す個別のユーザー定義関数を作成します。ユーザー定義関数は、モデルをホストする SageMaker エンドポイントを表します。この `aws_sagemaker.invoke_endpoint` 関数は、ユーザー定義関数内で実行されます。ユーザー定義関数には、次のような多くの利点があります。

- SageMaker モデルには、すべての SageMaker モデルを `aws_sagemaker.invoke_endpoint` とのみ呼ぶ代わりに、独自の名前を付けることができます。
- モデルエンドポイントの URL は、SQL アプリケーションコード内の 1 か所だけで指定できます。
- 各 Aurora 機械学習関数に対する `EXECUTE` 権限は、個別に制御できます。
- SQL タイプを使用して、モデルの入力および出力タイプを宣言することができます。SQL は、SageMaker モデルに渡される引数の数と型を強制し、必要に応じて型変換を実行します。SQL タイプを使用すると、`SQL NULL` も SageMaker モデルで期待される適切なデフォルト値に変換されます。
- 初期の数行を少し速く返す場合は、最大バッチサイズを小さくすることができます。

ユーザー定義関数を指定するには、SQL データ定義言語 (DDL) ステートメント `CREATE FUNCTION` を使用します。関数を定義するときは、以下を指定します。

- モデルへの入力パラメータ。
- 呼び出す特定の SageMaker エンドポイント。
- 戻り型。

このユーザー定義関数は、入力パラメータでモデルを実行した後、SageMaker エンドポイントによって計算された推論を返します。次の例では、2 つの入力パラメータを持つ SageMaker モデルのユーザー定義関数を作成します。

```
CREATE FUNCTION classify_event (IN arg1 INT, IN arg2 DATE, OUT category INT)
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
        'sagemaker_model_endpoint_name', NULL,
```

```
        arg1, arg2                          -- model inputs are separate arguments
        )::INT                              -- cast the output to INT
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

次の点に注意してください。

- aws_sagemaker.invoke_endpoint 関数入力には、任意のデータ型の 1 つ以上のパラメータを指定できます。
- この例では、INT 出力型を使用します。ある varchar 型から別の型に出力をキャストする場合は、INTEGER、REAL、FLOAT、または NUMERIC などのPostgreSQL組み込みスカラー型にキャストする必要があります。このような型の詳細については、PostgreSQL ドキュメントの「データ型」を参照してください。
- パラレルクエリ処理を有効にするには、PARALLEL SAFE を指定します。詳細については、「パラレルクエリ処理によるレスポンス時間の向上」を参照してください。
- 関数を実行するためのコストを見積もるには COST 5000 を指定します。関数の推定実行コストを示す正の数を cpu_operator_cost の単位で使用します 。

配列を入力として SageMaker モデルに渡す

この aws_sagemaker.invoke_endpoint 関数は、PostgreSQL 関数の上限である、最大 100 個の入力パラメータを持つことができます。SageMaker モデルが同じ型のパラメータを 100 個以上必要とする場合は、モデルパラメータを配列として渡します。

次の例では、SageMaker リグレッションモデルへの入力として配列を渡すユーザー関数を定義します。出力は値に REAL キャストされます。

```
CREATE FUNCTION regression_model (params REAL[], OUT estimate REAL)
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
      'sagemaker_model_endpoint_name',
      NULL,
      params
      )::REAL
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

SageMaker モデルの呼び出し時にバッチサイズを指定する

次の例では、バッチサイズのデフォルトを NULL に設定する SageMaker モデルのユーザー定義関数を作成します。この関数では、呼び出し時に異なるバッチサイズを指定することもできます。

```
CREATE FUNCTION classify_event (
    IN event_type INT, IN event_day DATE, IN amount REAL, -- model inputs
    max_rows_per_batch INT DEFAULT NULL,  -- optional batch size limit
    OUT category INT)                     -- model output
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
        'sagemaker_model_endpoint_name', max_rows_per_batch,
        event_type, event_day, COALESCE(amount, 0.0)
        )::INT              -- casts output to type INT
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

次の点に注意してください。

- オプションの max_rows_per_batch パラメータを使用すると、バッチモード関数呼び出しの行数を制御できます。NULL の値を使用すると、クエリオプティマイザは最大バッチサイズを自動的に選択します。詳細については、「バッチモードと Aurora 機械学習関数の理解」を参照してください。
- デフォルトでは、パラメータの値として NULL を渡すと、SageMaker に渡す前に空の文字列に変換されます。この例では、入力のタイプが異なります。
- テキスト以外の入力、または空の文字列以外の値をデフォルトにする必要があるテキスト入力がある場合は、COALESCE ステートメントを使用します。COALESCE を使用して、aws_sagemaker.invoke_endpoint への呼び出しで NULL を目的の NULL 置換値に変換します 。この例の amount パラメータでは、NULL 値が 0.0 に変換されます。

複数の出力を持つ SageMaker モデルの呼び出し

次の例では、複数の出力を返す SageMaker モデルのユーザー定義関数を作成します。関数は、aws_sagemaker.invoke_endpoint 関数の出力を対応するデータ型にキャストする必要があります。例えば、組み込み PostgreSQL ポイント型を (x,y) ペアまたはユーザー定義のコンポジット型に使用できます。

このユーザー定義関数は、出力にコンポジット型を使用して複数の出力を返すモデルから値を返します。

```
CREATE TYPE company_forecasts AS (
    six_month_estimated_return real,
    one_year_bankruptcy_probability float);
CREATE FUNCTION analyze_company (
```

```
    IN free_cash_flow NUMERIC(18, 6),
    IN debt NUMERIC(18,6),
    IN max_rows_per_batch INT DEFAULT NULL,
    OUT prediction company_forecasts)
AS $$
    SELECT (aws_sagemaker.invoke_endpoint('endpt_name',
       max_rows_per_batch,free_cash_flow, debt))::company_forecasts;


$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

コンポジット型の場合、モデル出力に表示されるのと同じ順序でフィールドを使用し、`aws_sagemaker.invoke_endpoint` の出力をコンポジット型にキャストします。呼び出し元は、名前または PostgreSQL「*」表記で個々のフィールドを抽出することができます。

SageMaker モデルトレーニング用のデータを Amazon S3 にエクスポートする (高度)

独自のモデルをトレーニングするよりも、提供されているアルゴリズムや例を使用して Aurora 機械学習と SageMaker に慣れることをお勧めします。詳細については、「Amazon SageMaker の開始方法」を参照してください。

SageMaker モデルをトレーニングするには、データを Amazon S3 バケットにエクスポートします。Amazon S3 バケットは、デプロイ前にモデルをトレーニングするために SageMaker によって使用されます。Aurora PostgreSQL DB クラスターからデータをクエリし、Amazon S3 バケットに保存されているテキストファイルに直接保存できます。その後、SageMaker は、トレーニングのために Amazon S3 バケットからデータを消費します。SageMaker モデルトレーニングの詳細については、「Amazon SageMaker でモデルをトレーニングする」を参照してください。

> **Note**
>
> SageMaker モデルトレーニングまたはバッチスコアリング用に Amazon S3 バケットを作成する場合は、Amazon S3 バケット名に `sagemaker` を使用してください。詳細については、「Amazon SageMaker デベロッパーガイド」の「トレーニングデータセットのアップロードと出力データの保存に Amazon S3 バケットを指定する」を参照してください。

データのエクスポートの詳細については、「Aurora PostgreSQL DB クラスターから Amazon S3 へのデータのエクスポート」を参照してください。

Aurora PostgreSQL で Aurora 機械学習を使用した場合のパフォーマンスに関する考慮事項

Amazon Comprehend と SageMaker のサービスは、Aurora の機械学習関数によって呼び出された際には、ほとんどの作業を行います。つまり、これらのリソースを必要に応じて個別にスケーリングできることになります。Aurora PostgreSQL DB クラスターでは、関数呼び出しを最大限効率的に行うことができます。以下に、Aurora PostgreSQL から Aurora 機械学習を使用する際に注意すべきパフォーマンスに関する考慮事項をいくつか示します。

トピック

バッチモードと Aurora 機械学習関数の理解

通常、PostgreSQL は関数を一度に 1 行ずつ実行します。Aurora 機械学習は、このオーバーヘッドを削減するため、バッチモード実行と呼ばれるアプローチを使用して、多くの行のための外部 Aurora 機械学習サービスの呼び出しをバッチ処理に結合します。バッチモードでは、Aurora 機械学習は入力行のバッチに対する応答を受け取り、その応答を一度に 1 行ずつ実行中のクエリに返送します。この最適化により、PostgreSQL クエリオプティマイザを制限せずに、Aurora クエリのスループットが向上します。

関数が SELECT リスト、WHERE 句、または HAVING 句から参照される場合、Aurora は自動的にバッチモードを使用します。トップレベルの単純な CASE 表現は、バッチモードの実行の対象であることに注意してください。最上位レベルの検索 CASE 表現は、初期の WHEN 句がバッチモード関数呼び出しを伴う単純な述語である場合にも、バッチモード実行の対象となります。

ユーザー定義関数は LANGUAGE SQL 関数で、PARALLEL SAFE と COST 5000 を指定する必要があります。

SELECT ステートメントから FROM 句への関数の移行

通常、バッチモード実行の対象となる aws_ml 関数は、Aurora によって自動的に FROM 句に移行されます。

対象のバッチモード関数から FROM 句への移行は、クエリごとのレベルで手動で調べることができます。これを行うには、EXPLAIN ステートメント (および ANALYZE および VERBOSE) を使用

し、各バッチモード Function Scan の下に「バッチ処理」情報を見つけます。クエリを実行せずに EXPLAIN (VERBOSE 付き) を使用することもできます。次に、関数への呼び出しが、元のステートメントで指定されていないネストされたループ結合の下に Function Scan と表示されるかどうかを観察します。

次の例では、プランにネストされたループ結合演算子によって、Aurora が anomaly_score 関数を移行したことを示しています。この関数を SELECT リストから、バッチモード実行の対象である、FROM 句に移行しました。

```
EXPLAIN (VERBOSE, COSTS false)
SELECT anomaly_score(ts.R.description) from ts.R;
                          QUERY PLAN
---------------------------------------------------------------
 Nested Loop
   Output: anomaly_score((r.description)::text)
   ->  Seq Scan on ts.r
         Output: r.id, r.description, r.score
   ->  Function Scan on public.anomaly_score
         Output: anomaly_score.anomaly_score
         Function Call: anomaly_score((r.description)::text)
```

バッチモードの実行を無効にするには、apg_enable_function_migration パラメータを false に設定します。これにより、SELECT から aws_ml 句への FROM 関数の移行が防止されます。以下にその方法を示します。

```
SET apg_enable_function_migration = false;
```

apg_enable_function_migration パラメータは、クエリプラン管理の Aurora PostgreSQL apg_plan_mgmt エクステンションによって認識される Grand Unified Configuration (GUC) パラメータです。セッションで関数の移行を無効にするには、クエリプラン管理を使用して、結果のプランを approved プランとして保存します。実行時に、クエリプラン管理によって、approved プランが apg_enable_function_migration 設定で適用されます。この強制は、apg_enable_function_migration GUC パラメータの設定に関係なく発生します。詳細については、「Aurora PostgreSQL のクエリ実行計画の管理」を参照してください。

max_rows_per_batch パラメータを使用する

aws_comprehend.detect_sentiment 関数と aws_sagemaker.invoke_endpoint 関数の両方に max_rows_per_batch パラメータがあります。このパラメータには、Aurora 機械学習サービス

に送信できる行数を指定します。関数で処理されるデータセットが大きいほど、バッチサイズを大きくすることができます。

バッチモード関数は、 Aurora 機械学習関数呼び出しのコストを多数の行に分散させる行のバッチを構築することにより、効率を向上させます。ただし、SELECT 句が原因で LIMIT ステートメントが早期終了した場合、クエリが使用する行よりも多くの行にわたってバッチを構築できます。この方法では、AWS アカウントに追加料金が発生する可能性があります。バッチモード実行の利点を得て、大きすぎるバッチの作成を避けるには、関数呼び出しで max_rows_per_batch パラメータに小さい値を使用します。

バッチモードの実行を使用するクエリの EXPLAIN (VERBOSE、ANALYZE) を実行すると、ネストされたループ結合の下にある FunctionScan 演算子が表示されます。EXPLAIN によって報告されるループ回数は、FunctionScan 演算子から行がフェッチされた回数に等しくなります。ステートメントが LIMIT 句を使用する場合、フェッチの数は一貫しています。バッチのサイズを最適化するには、max_rows_per_batch パラメータをこの値に設定します。ただし、バッチモード関数が WHERE 句または HAVING 句の述語で参照されている場合、事前にフェッチの数を知ることができない可能性があります。この場合、ループをガイドラインとして使用し、max_rows_per_batch で実験して、パフォーマンスを最適化する設定を見つけます。

バッチモード実行の検証

関数がバッチモードで実行されたかどうかを確認するには、EXPLAIN ANALYZE を使用します。バッチモードの実行が使用された場合、クエリプランは「バッチ処理」セクションに情報を含めます。

```
EXPLAIN ANALYZE SELECT user-defined-function();
 Batch Processing: num batches=1 avg/min/max batch size=3333.000/3333.000/3333.000
                                 avg/min/max batch call time=146.273/146.273/146.273
```

この例では、3,333 行を含む 1 つのバッチがあり、処理に 146.273 ミリ秒かかりました。「バッチ処理」セクションには、次の項目が表示されます。

- この関数スキャン操作に対して存在したバッチ数
- バッチサイズの平均、最小および最大
- バッチ実行時間の平均、最小および最大

通常、最終バッチは残りのバッチよりも小さく、多くの場合、平均よりかなり小さい最小バッチサイズになります。

初期の数行をより速く返すには、max_rows_per_batch パラメータを小さい値に設定します。

ユーザー定義関数で LIMIT を使用するときに ML サービスへのバッチモード呼び出し回数を減らすには、max_rows_per_batch パラメータの値を小さくします。

パラレルクエリ処理によるレスポンス時間の向上

多数の行からできるだけ早く結果を得るために、パラレルクエリ処理とバッチモード処理を組み合わせることができます。SELECT、CREATE TABLE AS SELECT、および CREATE MATERIALIZED VIEW ステートメントに対してパラレルクエリ処理を使用できます。

> **Note**
>
> PostgreSQL はまだデータ操作言語 (DML) ステートメントのパラレルクエリをサポートしていません。

パラレルクエリ処理は、データベース内と ML サービス内の両方で行われます。データベースのインスタンスクラスのコア数によって、クエリの実行中に使用できるパラレル性の程度が制限されます。データベースサーバーは、パラレルワーカーのセット間でタスクをパーティショニングするパラレルクエリ実行プランを構築できます。その後、これらのワーカーのそれぞれは、数万行 (または各サービスで許可される行数) を含むバッチリクエストを構築できます。

すべてのパラレルワーカーからのバッチ結合されたリクエストは、SageMaker のエンドポイントに送信されます。エンドポイントがサポートできる並列性の程度は、エンドポイントをサポートするインスタンスの数とタイプによって制限されます。K 度の並列性には、少なくとも K 個のコアを持つデータベースインスタンスクラスが必要です。また、十分に高性能なインスタンスクラスの初期インスタンスを K 個の持つようにモデルの SageMaker エンドポイントを設定する必要があります。

パラレルクエリ処理を利用するには、渡す予定のデータを含むテーブルの parallel_workers 格納パラメータを設定します。parallel_workers は、aws_comprehend.detect_sentiment などのバッチモード関数に設定します。オプティマイザがパラレルクエリプランを選択した場合、AWS ML サービスはバッチとパラレルの両方で呼び出すことができます。

aws_comprehend.detect_sentiment 関数で以下のパラメータを使用すると、四方向パラレル処理でプランを取得できます。次の 2 つのパラメータのいずれかを変更した場合、データベースインスタンスを再起動して変更を反映させる必要があります。

```
-- SET max_worker_processes to 8;  -- default value is 8
-- SET max_parallel_workers to 8;  -- not greater than max_worker_processes
SET max_parallel_workers_per_gather to 4;  -- not greater than max_parallel_workers

-- You can set the parallel_workers storage parameter on the table that the data
-- for the Aurora machine learning function is coming from in order to manually
 override the degree of
-- parallelism that would otherwise be chosen by the query optimizer
--
ALTER TABLE yourTable SET (parallel_workers = 4);

-- Example query to exploit both batch-mode execution and parallel query
EXPLAIN (verbose, analyze, buffers, hashes)
SELECT aws_comprehend.detect_sentiment(description, 'en')).*
FROM yourTable
WHERE id < 100;
```

パラレルクエリの制御情報の詳細については、PostgreSQL ドキュメントの「パラレルプラン」を参照してください。

マテリアライズドビューとマテリアライズド列の使用

SageMaker や Amazon Comprehend などの AWS のサービスをデータベースから呼び出すと、それらのサービスの料金ポリシーに基づきアカウントが課金されます。アカウントへの請求を最小限に抑えるために、AWS サービスを呼び出した結果をマテリアライズド列に生成して、AWS のサービスが入力行ごとに複数回呼び出されないようにすることができます。必要に応じて、materializedAt タイムスタンプカラムを追加して、カラムがマテリアライズされた時刻を記録できます。

通常の単一行 INSERT ステートメントのレイテンシーは、通常、バッチモード関数を呼び出すレイテンシーよりもはるかに短くなります。したがって、アプリケーションが実行するすべての単一行 INSERT に対してバッチモード関数を呼び出すと、アプリケーションのレイテンシー要件を満たすことができない場合があります。AWS のサービスをマテリアライズド列に呼び出した結果をマテリアライズするには、通常、高性能アプリケーションはマテリアライズド列に値を設定する必要があります。これを行うために、大量の行のバッチを同時に処理する UPDATE ステートメントを定期的に発行します。

UPDATE では、実行中のアプリケーションに影響を与える可能性のある行レベルのロックが行われます。したがって、SELECT ... FOR UPDATE SKIP LOCKED を使用するか、MATERIALIZED VIEW を使用する必要があります。

大量の行をリアルタイムで操作する分析クエリは、バッチモードのマテリアライズとリアルタイム処理を組み合わせることができます。これを行うために、これらのクエリは、事前マテリアライズド結果の UNION ALL をアセンブルし、まだマテリアライズド結果がない行をクエリします。場合によっては、このような UNION ALL が複数の場所で必要になるか、またはサードパーティーのアプリケーションによってクエリが生成されます。その場合は、VIEW を作成し、UNION ALL 操作をカプセル化して、この詳細が SQL アプリケーションの残りの部分に表示されないようにすることができます。

マテリアライズドビューを使用すると、スナップショットで任意の SELECT ステートメントの結果をマテリアライズできます。また、これを使用して、将来いつでもマテリアライズドビューを更新することもできます。現在、PostgreSQL は増分更新をサポートしていないため、マテリアライズドビューが更新されるたびにマテリアライズドビューが完全に再計算されます。

CONCURRENTLY オプションを使用してマテリアライズドビューをリフレッシュできます。このオプションを使用すると、排他ロックを取らずにマテリアライズドビューの内容が更新されます。これにより、SQL アプリケーションはマテリアライズドビューの更新中にマテリアライズドビューから読み取ることができます。

Aurora 機械学習のモニタリング

カスタム DB クラスターパラメータグループの track_functions パラメータを all に設定することで、aws_ml 関数をモニタリングできます。デフォルトでは、このパラメータは pl に設定されており、プロシージャ言語関数のみが追跡されることになります。これを all に変更すると、aws_ml 関数も追跡されます。詳細については、PostgreSQL のドキュメントで「ランタイム統計」を参照してください。

Aurora 機械学習機能から呼び出される SageMaker オペレーションのパフォーマンスのモニタリングについては、「Amazon SageMaker デベロッパーガイド」の「Amazon SageMaker のモニタリング」を参照してください。

track_functions を all に設定すると、pg_stat_user_functions ビューをクエリして、Aurora 機械学習サービスを呼び出すために定義および使用した関数に関する統計を取得できます。各関数について、ビューに calls、total_time、self_time の数が表示されます。

aws_sagemaker.invoke_endpoint 関数と aws_comprehend.detect_sentiment 関数の統計を表示するには、次のクエリを使用してスキーマ名で結果をフィルタリングできます。

```
SELECT * FROM pg_stat_user_functions
    WHERE schemaname
```

```
    LIKE 'aws_%';
```

統計を消去するには、以下の手順を実行してください。

```
SELECT pg_stat_reset();
```

PostgreSQL pg_proc システムカタログにクエリすることで、aws_sagemaker.invoke_endpoint 関数を呼び出す SQL 関数の名前を取得できます。このカタログには、関数、プロシージャなどに関する情報が保存されています。詳細については、PostgreSQL ドキュメントの「pg_proc」を参照してください。次は、ソース (prosrc) に invoke_endpoint というテキストが含まれている関数 (proname) の名前を取得するためにテーブルをクエリする例を示しています。

```
SELECT proname FROM pg_proc WHERE prosrc LIKE '%invoke_endpoint%';
```

AWS SDK を使用した Aurora のコード例

以下は、AWS Software Development Kit (SDK) で Aurora を使用する方法を説明するコード例です。

アクションはより大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。アクションは個々のサービス機能を呼び出す方法を示していますが、関連するシナリオやサービス間の例ではアクションのコンテキストが確認できます。

「シナリオ」は、同じサービス内で複数の関数を呼び出して、特定のタスクを実行する方法を示すコード例です。

クロスサービスの例は、複数の AWS のサービス で動作するサンプルアプリケーションです。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

開始方法

Hello Aurora

次のコード例は、Aurora の使用を開始する方法を示しています。

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
using Amazon.RDS;
using Amazon.RDS.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
```

```
namespace AuroraActions;

public static class HelloAurora
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the
        // Amazon Relational Database Service (Amazon RDS).
        // Use your AWS profile name, or leave it blank to use the default
 profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRDS>()
            ).Build();

        // Now the client is available for injection. Fetching it directly here
 for example purposes only.
        var rdsClient = host.Services.GetRequiredService<IAmazonRDS>();

        // You can use await and any of the async methods to get a response.
        var response = await rdsClient.DescribeDBClustersAsync(new
 DescribeDBClustersRequest { IncludeShared = true });
        Console.WriteLine($"Hello Amazon RDS Aurora! Let's list some clusters in
 this account:");
        foreach (var cluster in response.DBClusters)
        {
            Console.WriteLine($"\tCluster: database: {cluster.DatabaseName}
 identifier: {cluster.DBClusterIdentifier}.");
        }
    }
}
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeDBClusters」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

CMakeLists.txt CMake ファイルのコード。

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS rds)

# Set this project's name.
project("hello_aurora")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.
```

```
    # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
                                    # and set the proper subdirectory to the
 executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_aurora.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_aurora.cpp ソースファイルのコード。

```
#include <aws/core/Aws.h>
#include <aws/rds/RDSClient.h>
#include <aws/rds/model/DescribeDBClustersRequest.h>
#include <iostream>

/*
 *  A "Hello Aurora" starter application which initializes an Amazon Relational
 Database Service (Amazon RDS) client
 *  and describes the Amazon Aurora (Aurora) clusters.
 *
 *  main function
 *
 *  Usage: 'hello_aurora'
 *
 */
int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
```

```
        Aws::RDS::RDSClient rdsClient(clientConfig);

        Aws::String marker; // Used for pagination.
        std::vector<Aws::String> clusterIds;
        do {
            Aws::RDS::Model::DescribeDBClustersRequest request;

            Aws::RDS::Model::DescribeDBClustersOutcome outcome =
                    rdsClient.DescribeDBClusters(request);

            if (outcome.IsSuccess()) {
                for (auto &cluster: outcome.GetResult().GetDBClusters()) {
                    clusterIds.push_back(cluster.GetDBClusterIdentifier());
                }
                marker = outcome.GetResult().GetMarker();
            } else {
                result = 1;
                std::cerr << "Error with Aurora::GDescribeDBClusters. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                break;
            }
        } while (!marker.empty());

        std::cout << clusterIds.size() << " Aurora clusters found." << std::endl;
        for (auto &clusterId: clusterIds) {
            std::cout << "  clusterId " << clusterId << std::endl;
        }
    }

    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「DescribeDBClusters」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/rds"
)

// main uses the AWS SDK for Go V2 to create an Amazon Aurora client and list up
 to 20
// DB clusters in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 sdkConfig, err := config.LoadDefaultConfig(context.TODO())
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 auroraClient := rds.NewFromConfig(sdkConfig)
 const maxClusters = 20
 fmt.Printf("Let's list up to %v DB clusters.\n", maxClusters)
 output, err := auroraClient.DescribeDBClusters(context.TODO(),
  &rds.DescribeDBClustersInput{MaxRecords: aws.Int32(maxClusters)})
 if err != nil {
  fmt.Printf("Couldn't list DB clusters: %v\n", err)
```

```
  return
 }
 if len(output.DBClusters) == 0 {
  fmt.Println("No DB clusters found.")
 } else {
  for _, cluster := range output.DBClusters {
   fmt.Printf("DB cluster %v has database %v.\n", *cluster.DBClusterIdentifier,
    *cluster.DatabaseName)
  }
 }
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「DescribeDBClusters」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rds.RdsClient;
import software.amazon.awssdk.services.rds.paginators.DescribeDBClustersIterable;

public class DescribeDbClusters {
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        describeClusters(rdsClient);
        rdsClient.close();
    }
```

```
    public static void describeClusters(RdsClient rdsClient) {
        DescribeDBClustersIterable clustersIterable =
 rdsClient.describeDBClustersPaginator();
        clustersIterable.stream()
                .flatMap(r -> r.dbClusters().stream())
                .forEach(cluster -> System.out
                        .println("Database name: " + cluster.databaseName() + "
 Arn = " + cluster.dbClusterArn()));
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DescribeDBClusters」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_sdk_rds::Client;

#[derive(Debug)]
struct Error(String);
impl std::fmt::Display for Error {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "{}", self.0)
    }
}
impl std::error::Error for Error {}

#[tokio::main]
async fn main() -> Result<(), Error> {
```

```
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::from_env().load().await;
    let client = Client::new(&sdk_config);

    let describe_db_clusters_output = client
        .describe_db_clusters()
        .send()
        .await
        .map_err(|e| Error(e.to_string()))?;
    println!(
        "Found {} clusters:",
        describe_db_clusters_output.db_clusters().len()
    );
    for cluster in describe_db_clusters_output.db_clusters() {
        let name = cluster.database_name().unwrap_or("Unknown");
        let engine = cluster.engine().unwrap_or("Unknown");
        let id = cluster.db_cluster_identifier().unwrap_or("Unknown");
        let class = cluster.db_cluster_instance_class().unwrap_or("Unknown");
        println!("\tDatabase: {name}",);
        println!("\t  Engine: {engine}",);
        println!("\t      ID: {id}",);
        println!("\tInstance: {class}",);
    }

    Ok(())
}
```

- API の詳細については、AWS SDK for Rust API リファレンスの「DescribeDBClusters」を参照してください。

コードの例

- AWS SDK を使用した Aurora 向けアクション
  - AWS SDK または CLI で CreateDBCluster を使用する
  - AWS SDK または CLI で CreateDBClusterParameterGroup を使用する
  - AWS SDK または CLI で CreateDBClusterSnapshot を使用する
  - AWS SDK または CLI で CreateDBInstance を使用する
  - AWS SDK または CLI で DeleteDBCluster を使用する
  - AWS SDK または CLI で DeleteDBClusterParameterGroup を使用する

AWS SDK を使用した Aurora 向けアクション

以下は、AWS SDK を使用して個々の Aurora アクションを実行する方法を説明するコード例です。これらは Aurora API を呼び出すもので、コンテキスト内で実行する必要がある大規模なプログラムからのコード抜粋です。それぞれの例には、GitHub へのリンクがあり、そこにはコードの設定と実行に関する説明が記載されています。

以下の例には、最も一般的に使用されるアクションのみ含まれています。詳細な一覧については、「Amazon Aurora API Reference」(Amazon Aurora API リファレンス) を参照してください。

例

- AWS SDK または CLI で DescribeDBClusterParameterGroups を使用する
- AWS SDK または CLI で DescribeDBClusterParameters を使用する
- AWS SDK または CLI で DescribeDBClusterSnapshots を使用する
- AWS SDK または CLI で DescribeDBClusters を使用する
- AWS SDK または CLI で DescribeDBEngineVersions を使用する
- AWS SDK または CLI で DescribeDBInstances を使用する
- AWS SDK または CLI で DescribeOrderableDBInstanceOptions を使用する
- AWS SDK または CLI で ModifyDBClusterParameterGroup を使用する

AWS SDK または CLI で **CreateDBCluster** を使用する

以下のコード例は、CreateDBCluster の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- DB クラスターの開始方法

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
/// <summary>
/// Create a new cluster and database.
/// </summary>
/// <param name="dbName">The name of the new database.</param>
/// <param name="clusterIdentifier">The identifier of the cluster.</param>
/// <param name="parameterGroupName">The name of the parameter group.</param>
/// <param name="dbEngine">The engine to use for the new cluster.</param>
/// <param name="dbEngineVersion">The version of the engine to use.</param>
/// <param name="adminName">The admin username.</param>
```

```
    /// <param name="adminPassword">The primary admin password.</param>
    /// <returns>The cluster object.</returns>
    public async Task<DBCluster> CreateDBClusterWithAdminAsync(
        string dbName,
        string clusterIdentifier,
        string parameterGroupName,
        string dbEngine,
        string dbEngineVersion,
        string adminName,
        string adminPassword)
    {
        var request = new CreateDBClusterRequest
        {
            DatabaseName = dbName,
            DBClusterIdentifier = clusterIdentifier,
            DBClusterParameterGroupName = parameterGroupName,
            Engine = dbEngine,
            EngineVersion = dbEngineVersion,
            MasterUsername = adminName,
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「CreateDBCluster」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
```

```
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::CreateDBClusterRequest request;
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetEngine(engineName);
        request.SetEngineVersion(engineVersionName);
        request.SetMasterUsername(administratorName);
        request.SetMasterUserPassword(administratorPassword);

        Aws::RDS::Model::CreateDBClusterOutcome outcome =
                client.CreateDBCluster(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBCluster. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
            return false;
        }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「CreateDBCluster」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateDbCluster creates a DB cluster that is configured to use the specified
 parameter group.
// The newly created DB cluster contains a database that uses the specified
 engine and
// engine version.
func (clusters *DbClusters) CreateDbCluster(clusterName string,
 parameterGroupName string,
 dbName string, dbEngine string, dbEngineVersion string, adminName string,
 adminPassword string) (
 *types.DBCluster, error) {

 output, err := clusters.AuroraClient.CreateDBCluster(context.TODO(),
 &rds.CreateDBClusterInput{
  DBClusterIdentifier:         aws.String(clusterName),
  Engine:                      aws.String(dbEngine),
  DBClusterParameterGroupName: aws.String(parameterGroupName),
  DatabaseName:                aws.String(dbName),
  EngineVersion:               aws.String(dbEngineVersion),
  MasterUserPassword:          aws.String(adminPassword),
  MasterUsername:              aws.String(adminName),
 })
 if err != nil {
  log.Printf("Couldn't create DB cluster %v: %v\n", clusterName, err)
  return nil, err
 } else {
  return output.DBCluster, err
 }
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「CreateDBCluster」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   public static String createDBCluster(RdsClient rdsClient, String
dbParameterGroupFamily, String dbName,
           String dbClusterIdentifier, String userName, String password) {
       try {
           CreateDbClusterRequest clusterRequest =
CreateDbClusterRequest.builder()
                   .databaseName(dbName)
                   .dbClusterIdentifier(dbClusterIdentifier)
                   .dbClusterParameterGroupName(dbParameterGroupFamily)
                   .engine("aurora-mysql")
                   .masterUsername(userName)
                   .masterUserPassword(password)
                   .build();

           CreateDbClusterResponse response =
rdsClient.createDBCluster(clusterRequest);
           return response.dbCluster().dbClusterArn();

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
       return "";
   }
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「CreateDBCluster」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun createDBCluster(dbParameterGroupFamilyVal: String?, dbName: String?,
 dbClusterIdentifierVal: String?, userName: String?, password: String?): String?
 {
    val clusterRequest = CreateDbClusterRequest {
        databaseName = dbName
        dbClusterIdentifier = dbClusterIdentifierVal
        dbClusterParameterGroupName = dbParameterGroupFamilyVal
        engine = "aurora-mysql"
        masterUsername = userName
        masterUserPassword = password
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbCluster(clusterRequest)
        return response.dbCluster?.dbClusterArn
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「CreateDBCluster」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_db_cluster(
        self,
        cluster_name,
        parameter_group_name,
        db_name,
        db_engine,
        db_engine_version,
        admin_name,
        admin_password,
    ):
        """
        Creates a DB cluster that is configured to use the specified parameter
 group.
        The newly created DB cluster contains a database that uses the specified
 engine and
        engine version.

        :param cluster_name: The name of the DB cluster to create.
        :param parameter_group_name: The name of the parameter group to associate
 with
                                     the DB cluster.
        :param db_name: The name of the database to create.
        :param db_engine: The database engine of the database that is created,
 such as MySql.
```

```
        :param db_engine_version: The version of the database engine.
        :param admin_name: The user name of the database administrator.
        :param admin_password: The password of the database administrator.
        :return: The newly created DB cluster.
        """
        try:
            response = self.rds_client.create_db_cluster(
                DatabaseName=db_name,
                DBClusterIdentifier=cluster_name,
                DBClusterParameterGroupName=parameter_group_name,
                Engine=db_engine,
                EngineVersion=db_engine_version,
                MasterUsername=admin_name,
                MasterUserPassword=admin_password,
            )
            cluster = response["DBCluster"]
        except ClientError as err:
            logger.error(
                "Couldn't create database %s. Here's why: %s: %s",
                db_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return cluster
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「CreateDBCluster」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_ENGINE,
               self.engine_version.as_deref().expect("engine version"),
               self.username.as_deref().expect("username"),
               self.password
                   .replace(SecretString::new("".to_string()))
                   .expect("password"),
           )
           .await;
       if let Err(err) = create_db_cluster {
           return Err(ScenarioError::new(
               "Failed to create DB Cluster with cluster group",
               &err,
           ));
       }

       self.db_cluster_identifier = create_db_cluster
           .unwrap()
```

```
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
```

```
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
```

```
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn create_db_cluster(
        &self,
        name: &str,
        parameter_group: &str,
        engine: &str,
        version: &str,
        username: &str,
        password: SecretString,
    ) -> Result<CreateDbClusterOutput, SdkError<CreateDBClusterError>> {
        self.inner
            .create_db_cluster()
            .db_cluster_identifier(name)
            .db_cluster_parameter_group_name(parameter_group)
            .engine(engine)
            .engine_version(version)
            .master_username(username)
            .master_user_password(password.expose_secret())
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();
```

```
    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()
```

```
.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
```

```
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
```

```
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
```

```
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
```

```
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
```

```
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「CreateDBCluster」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **CreateDBClusterParameterGroup** を使用する

以下のコード例は、CreateDBClusterParameterGroup の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- DB クラスターの開始方法

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create a custom cluster parameter group.
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
    /// <param name="description">A description for the new parameter group.</
param>
    /// <returns>The new parameter group object.</returns>
    public async Task<DBClusterParameterGroup>
 CreateCustomClusterParameterGroupAsync(
        string parameterGroupFamily,
        string groupName,
        string description)
    {
        var request = new CreateDBClusterParameterGroupRequest
```

```
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
            Description = description,
        };

        var response = await
_amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「CreateDBClusterParameterGroup」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::CreateDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetDBParameterGroupFamily(dbParameterGroupFamily);
        request.SetDescription("Example cluster parameter group.");

        Aws::RDS::Model::CreateDBClusterParameterGroupOutcome outcome =
                client.CreateDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
```

```
            std::cout << "The DB cluster parameter group was successfully
 created."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「CreateDBClusterParameterGroup」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateParameterGroup creates a DB cluster parameter group that is based on the
 specified
// parameter group family.
func (clusters *DbClusters) CreateParameterGroup(
 parameterGroupName string, parameterGroupFamily string, description string) (
 *types.DBClusterParameterGroup, error) {

 output, err :=
 clusters.AuroraClient.CreateDBClusterParameterGroup(context.TODO(),
```

```
  &rds.CreateDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   DBParameterGroupFamily:      aws.String(parameterGroupFamily),
   Description:                 aws.String(description),
  })
 if err != nil {
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBClusterParameterGroup, err
 }
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「CreateDBClusterParameterGroup」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   public static void createDBClusterParameterGroup(RdsClient rdsClient, String
dbClusterGroupName,
           String dbParameterGroupFamily) {
       try {
           CreateDbClusterParameterGroupRequest groupRequest =
CreateDbClusterParameterGroupRequest.builder()
                   .dbClusterParameterGroupName(dbClusterGroupName)
                   .dbParameterGroupFamily(dbParameterGroupFamily)
                   .description("Created by using the AWS SDK for Java")
                   .build();

           CreateDbClusterParameterGroupResponse response =
rdsClient.createDBClusterParameterGroup(groupRequest);
```

```
            System.out.println("The group name is " +
 response.dbClusterParameterGroup().dbClusterParameterGroupName());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「CreateDBClusterParameterGroup」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun createDBClusterParameterGroup(dbClusterGroupNameVal: String?,
 dbParameterGroupFamilyVal: String?) {
    val groupRequest = CreateDbClusterParameterGroupRequest {
        dbClusterParameterGroupName = dbClusterGroupNameVal
        dbParameterGroupFamily = dbParameterGroupFamilyVal
        description = "Created by using the AWS SDK for Kotlin"
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterParameterGroup(groupRequest)
        println("The group name is
 ${response.dbClusterParameterGroup?.dbClusterParameterGroupName}")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「CreateDBClusterParameterGroup」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_parameter_group(
        self, parameter_group_name, parameter_group_family, description
    ):
        """
        Creates a DB cluster parameter group that is based on the specified
 parameter group
        family.

        :param parameter_group_name: The name of the newly created parameter
 group.
        :param parameter_group_family: The family that is used as the basis of
 the new
                                       parameter group.
```

```
        :param description: A description given to the parameter group.
        :return: Data about the newly created parameter group.
        """
        try:
            response = self.rds_client.create_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                DBParameterGroupFamily=parameter_group_family,
                Description=description,
            )
        except ClientError as err:
            logger.error(
                "Couldn't create parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「CreateDBClusterParameterGroup」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   // Select an engine family and create a custom DB cluster parameter group.
rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
   pub async fn set_engine(&mut self, engine: &str, version: &str) -> Result<(),
ScenarioError> {
        self.engine_family = Some(engine.to_string());
        self.engine_version = Some(version.to_string());
```

```
        let create_db_cluster_parameter_group = self
            .rds
            .create_db_cluster_parameter_group(
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION,
                engine,
            )
            .await;

        match create_db_cluster_parameter_group {
            Ok(CreateDbClusterParameterGroupOutput {
                db_cluster_parameter_group: None,
                ..
            }) => {
                return Err(ScenarioError::with(
                    "CreateDBClusterParameterGroup had empty response",
                ));
            }
            Err(error) => {
                if error.code() == Some("DBParameterGroupAlreadyExists") {
                    info!("Cluster Parameter Group already exists, nothing to
do");
                } else {
                    return Err(ScenarioError::new(
                        "Could not create Cluster Parameter Group",
                        &error,
                    ));
                }
            }
            _ => {
                info!("Created Cluster Parameter Group");
            }
        }

        Ok(())
    }

    pub async fn create_db_cluster_parameter_group(
        &self,
        name: &str,
        description: &str,
        family: &str,
    ) -> Result<CreateDbClusterParameterGroupOutput,
SdkError<CreateDBClusterParameterGroupError>>
```

```
    {
        self.inner
            .create_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .description(description)
            .db_parameter_group_family(family)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_set_engine() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert_eq!(set_engine, Ok(()));
    assert_eq!(Some("aurora-mysql"), scenario.engine_family.as_deref());
    assert_eq!(Some("aurora-mysql8.0"), scenario.engine_version.as_deref());
}

#[tokio::test]
async fn test_scenario_set_engine_not_create() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
```

```
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _|
 Ok(CreateDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}

#[tokio::test]
async fn test_scenario_set_engine_param_group_exists() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .withf(|_, _, _| true)
        .return_once(|_, _, _| {
            Err(SdkError::service_error(

 CreateDBClusterParameterGroupError::DbParameterGroupAlreadyExistsFault(
                    DbParameterGroupAlreadyExistsFault::builder().build(),
                ),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「CreateDBClusterParameterGroup」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **CreateDBClusterSnapshot** を使用する

以下のコード例は、CreateDBClusterSnapshot の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- DB クラスターの開始方法

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
```

```
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });

        return response.DBClusterSnapshot;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「CreateDBClusterSnapshot」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    Aws::Client::ClientConfiguration clientConfig;
    // Optional: Set to the AWS Region (overrides config file).
    // clientConfig.region = "us-east-1";

Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::CreateDBClusterSnapshotRequest request;
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetDBClusterSnapshotIdentifier(snapshotID);

        Aws::RDS::Model::CreateDBClusterSnapshotOutcome outcome =
                client.CreateDBClusterSnapshot(request);

        if (outcome.IsSuccess()) {
            std::cout << "Snapshot creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBClusterSnapshot. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
```

```
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
 client);
                return false;
            }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「CreateDBClusterSnapshot」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateClusterSnapshot creates a snapshot of a DB cluster.
func (clusters *DbClusters) CreateClusterSnapshot(clusterName string,
 snapshotName string) (
 *types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.CreateDBClusterSnapshot(context.TODO(),
 &rds.CreateDBClusterSnapshotInput{
  DBClusterIdentifier:         aws.String(clusterName),
  DBClusterSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return output.DBClusterSnapshot, nil
```

```
 }
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「CreateDBClusterSnapshot」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   public static void createDBClusterSnapshot(RdsClient rdsClient, String
dbInstanceClusterIdentifier,
           String dbSnapshotIdentifier) {
       try {
           CreateDbClusterSnapshotRequest snapshotRequest =
CreateDbClusterSnapshotRequest.builder()
                   .dbClusterIdentifier(dbInstanceClusterIdentifier)
                   .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                   .build();

           CreateDbClusterSnapshotResponse response =
rdsClient.createDBClusterSnapshot(snapshotRequest);
           System.out.println("The Snapshot ARN is " +
response.dbClusterSnapshot().dbClusterSnapshotArn());

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「CreateDBClusterSnapshot」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun createDBClusterSnapshot(dbInstanceClusterIdentifier: String?,
 dbSnapshotIdentifier: String?) {
    val snapshotRequest = CreateDbClusterSnapshotRequest {
        dbClusterIdentifier = dbInstanceClusterIdentifier
        dbClusterSnapshotIdentifier = dbSnapshotIdentifier
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterSnapshot(snapshotRequest)
        println("The Snapshot ARN is
 ${response.dbClusterSnapshot?.dbClusterSnapshotArn}")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「CreateDBClusterSnapshot」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_cluster_snapshot(self, snapshot_id, cluster_id):
        """
        Creates a snapshot of a DB cluster.

        :param snapshot_id: The ID to give the created snapshot.
        :param cluster_id: The DB cluster to snapshot.
        :return: Data about the newly created snapshot.
        """
        try:
            response = self.rds_client.create_db_cluster_snapshot(
                DBClusterSnapshotIdentifier=snapshot_id,
 DBClusterIdentifier=cluster_id
            )
            snapshot = response["DBClusterSnapshot"]
        except ClientError as err:
            logger.error(
                "Couldn't create snapshot of %s. Here's why: %s: %s",
                cluster_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「CreateDBClusterSnapshot」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
```

```
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_ENGINE,
                self.engine_version.as_deref().expect("engine version"),
                self.username.as_deref().expect("username"),
                self.password
                    .replace(SecretString::new("".to_string()))
                    .expect("password"),
            )
            .await;
        if let Err(err) = create_db_cluster {
            return Err(ScenarioError::new(
                "Failed to create DB Cluster with cluster group",
                &err,
            ));
        }

        self.db_cluster_identifier = create_db_cluster
            .unwrap()
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
```

```
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }
```

```
            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn snapshot_cluster(
        &self,
        db_cluster_identifier: &str,
        snapshot_name: &str,
    ) -> Result<CreateDbClusterSnapshotOutput,
SdkError<CreateDBClusterSnapshotError>> {
        self.inner
```

```
            .create_db_cluster_snapshot()
            .db_cluster_identifier(db_cluster_identifier)
            .db_cluster_snapshot_identifier(snapshot_name)
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
```

```
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
```

```
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));
```

```
    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
```

```
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
```

```
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
```

```
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
```

```
    let _ = assertions.await;
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「CreateDBClusterSnapshot」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **CreateDBInstance** を使用する

以下のコード例は、CreateDBInstance の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- DB クラスターの開始方法

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
    /// with a particular set of properties. Use the action
DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
```

```
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
        // When creating the instance within a cluster, do not specify the name
or size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「CreateDBInstance」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
```

```
    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::CreateDBInstanceRequest request;
        request.SetDBInstanceIdentifier(DB_INSTANCE_IDENTIFIER);
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetEngine(engineName);
        request.SetDBInstanceClass(dbInstanceClass);

        Aws::RDS::Model::CreateDBInstanceOutcome outcome =
                client.CreateDBInstance(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB instance creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with RDS::CreateDBInstance. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER,
"",
                             client);
            return false;
        }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「CreateDBInstance」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateInstanceInCluster creates a database instance in an existing DB cluster.
 The first database that is
// created defaults to a read-write DB instance.
func (clusters *DbClusters) CreateInstanceInCluster(clusterName string,
 instanceName string,
 dbEngine string, dbInstanceClass string) (*types.DBInstance, error) {
 output, err := clusters.AuroraClient.CreateDBInstance(context.TODO(),
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBClusterIdentifier:  aws.String(clusterName),
  Engine:               aws.String(dbEngine),
  DBInstanceClass:      aws.String(dbInstanceClass),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
  return output.DBInstance, nil
 }
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「CreateDBInstance」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    public static String createDBInstanceCluster(RdsClient rdsClient,
            String dbInstanceIdentifier,
            String dbInstanceClusterIdentifier,
            String instanceClass) {
        try {
            CreateDbInstanceRequest instanceRequest =
CreateDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .engine("aurora-mysql")
                    .dbInstanceClass(instanceClass)
                    .build();

            CreateDbInstanceResponse response =
rdsClient.createDBInstance(instanceRequest);
            System.out.print("The status is " +
response.dbInstance().dbInstanceStatus());
            return response.dbInstance().dbInstanceArn();

        } catch (RdsException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「CreateDBInstance」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun createDBInstanceCluster(dbInstanceIdentifierVal: String?,
 dbInstanceClusterIdentifierVal: String?, instanceClassVal: String?): String? {
    val instanceRequest = CreateDbInstanceRequest {
        dbInstanceIdentifier = dbInstanceIdentifierVal
        dbClusterIdentifier = dbInstanceClusterIdentifierVal
        engine = "aurora-mysql"
        dbInstanceClass = instanceClassVal
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbInstance(instanceRequest)
        print("The status is ${response.dbInstance?.dbInstanceStatus}")
        return response.dbInstance?.dbInstanceArn
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「CreateDBInstance」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
```

```
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_instance_in_cluster(
        self, instance_id, cluster_id, db_engine, instance_class
    ):
        """
        Creates a database instance in an existing DB cluster. The first database
that is
        created defaults to a read-write DB instance.

        :param instance_id: The ID to give the newly created DB instance.
        :param cluster_id: The ID of the DB cluster where the DB instance is
created.
        :param db_engine: The database engine of a database to create in the DB
instance.
                          This must be compatible with the configured parameter
group
                          of the DB cluster.
        :param instance_class: The DB instance class for the newly created DB
instance.
        :return: Data about the newly created DB instance.
        """
        try:
            response = self.rds_client.create_db_instance(
                DBInstanceIdentifier=instance_id,
                DBClusterIdentifier=cluster_id,
                Engine=db_engine,
                DBInstanceClass=instance_class,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't create DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

```
        else:
            return db_inst
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「CreateDBInstance」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
```

```
            .create_db_cluster(
                DB_CLUSTER_IDENTIFIER,
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_ENGINE,
                self.engine_version.as_deref().expect("engine version"),
                self.username.as_deref().expect("username"),
                self.password
                    .replace(SecretString::new("".to_string()))
                    .expect("password"),
            )
            .await;
        if let Err(err) = create_db_cluster {
            return Err(ScenarioError::new(
                "Failed to create DB Cluster with cluster group",
                &err,
            ));
        }

        self.db_cluster_identifier = create_db_cluster
            .unwrap()
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
```

```
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
```

```
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn create_db_instance(
        &self,
        cluster_name: &str,
        instance_name: &str,
        instance_class: &str,
```

```
        engine: &str,
    ) -> Result<CreateDbInstanceOutput, SdkError<CreateDBInstanceError>> {
        self.inner
            .create_db_instance()
            .db_cluster_identifier(cluster_name)
            .db_instance_identifier(instance_name)
            .db_instance_class(instance_class)
            .engine(engine)
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
```

```
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });
```

```
    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });
```

```
    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
```

```
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
```

```
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
```

```
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
```

```
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「CreateDBInstance」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **DeleteDBCluster** を使用する

以下のコード例は、DeleteDBCluster の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- DB クラスターの開始方法

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>DB cluster object.</returns>
```

```
    public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
    {
        var response = await _amazonRDS.DeleteDBClusterAsync(
            new DeleteDBClusterRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                SkipFinalSnapshot = true
            });

        return response.DBCluster;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DeleteDBCluster」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DeleteDBClusterRequest request;
            request.SetDBClusterIdentifier(dbClusterIdentifier);
            request.SetSkipFinalSnapshot(true);

            Aws::RDS::Model::DeleteDBClusterOutcome outcome =
                    client.DeleteDBCluster(request);

            if (outcome.IsSuccess()) {
```

```
                std::cout << "DB cluster deletion has started."
                          << std::endl;
                clusterDeleting = true;
                std::cout
                        << "Waiting for DB cluster to delete before deleting the
 parameter group."
                        << std::endl;
                std::cout << "This may take a while." << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBCluster. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「DeleteDBCluster」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteDbCluster deletes a DB cluster without keeping a final snapshot.
func (clusters *DbClusters) DeleteDbCluster(clusterName string) error {
 _, err := clusters.AuroraClient.DeleteDBCluster(context.TODO(),
 &rds.DeleteDBClusterInput{
```

```
  DBClusterIdentifier: aws.String(clusterName),
  SkipFinalSnapshot:   true,
 })
 if err != nil {
  log.Printf("Couldn't delete DB cluster %v: %v\n", clusterName, err)
  return err
 } else {
  return nil
 }
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「DeleteDBCluster」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   public static void deleteCluster(RdsClient rdsClient, String
dbInstanceClusterIdentifier) {
       try {
           DeleteDbClusterRequest deleteDbClusterRequest =
DeleteDbClusterRequest.builder()
                   .dbClusterIdentifier(dbInstanceClusterIdentifier)
                   .skipFinalSnapshot(true)
                   .build();

           rdsClient.deleteDBCluster(deleteDbClusterRequest);
           System.out.println(dbInstanceClusterIdentifier + " was deleted!");

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
```

```
        }
    }
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「DeleteDBCluster」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun deleteCluster(dbInstanceClusterIdentifier: String) {
    val deleteDbClusterRequest = DeleteDbClusterRequest {
        dbClusterIdentifier = dbInstanceClusterIdentifier
        skipFinalSnapshot = true
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        rdsClient.deleteDbCluster(deleteDbClusterRequest)
        println("$dbInstanceClusterIdentifier was deleted!")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「DeleteDBCluster」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_db_cluster(self, cluster_name):
        """
        Deletes a DB cluster.

        :param cluster_name: The name of the DB cluster to delete.
        """
        try:
            self.rds_client.delete_db_cluster(
                DBClusterIdentifier=cluster_name, SkipFinalSnapshot=True
            )
            logger.info("Deleted DB cluster %s.", cluster_name)
        except ClientError:
            logger.exception("Couldn't delete DB cluster %s.", cluster_name)
```

```
            raise
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「DeleteDBCluster」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
```

```
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
```

```
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
```

```
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn delete_db_cluster(
        &self,
        cluster_identifier: &str,
    ) -> Result<DeleteDbClusterOutput, SdkError<DeleteDBClusterError>> {
        self.inner
            .delete_db_cluster()
            .db_cluster_identifier(cluster_identifier)
            .skip_final_snapshot(true)
```

```
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
```

```
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
```

```
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
```

```
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
```

```
        assert_matches!(errs.get(0), Some(ScenarioError {message, context: _}) if
 message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「DeleteDBCluster」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **DeleteDBClusterParameterGroup** を使用する

以下のコード例は、DeleteDBClusterParameterGroup の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- DB クラスターの開始方法

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete a particular parameter group by name.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterParameterGroupByNameAsync(string
groupName)
    {
        var request = new DeleteDBClusterParameterGroupRequest
        {
            DBClusterParameterGroupName = groupName,
        };

        var response = await
_amazonRDS.DeleteDBClusterParameterGroupAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DeleteDBClusterParameterGroup」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::DeleteDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(parameterGroupName);

        Aws::RDS::Model::DeleteDBClusterParameterGroupOutcome outcome =
                client.DeleteDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB parameter group was successfully deleted."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::DeleteDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「DeleteDBClusterParameterGroup」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
type DbClusters struct {
 AuroraClient *rds.Client
}
```

```
// DeleteParameterGroup deletes the named DB cluster parameter group.
func (clusters *DbClusters) DeleteParameterGroup(parameterGroupName string) error
 {
 _, err := clusters.AuroraClient.DeleteDBClusterParameterGroup(context.TODO(),
  &rds.DeleteDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「DeleteDBClusterParameterGroup」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   public static void deleteDBClusterGroup(RdsClient rdsClient, String
dbClusterGroupName, String clusterDBARN)
           throws InterruptedException {
       try {
           boolean isDataDel = false;
           boolean didFind;
           String instanceARN;
```

```
            // Make sure that the database has been deleted.
            while (!isDataDel) {
                DescribeDbInstancesResponse response =
rdsClient.describeDBInstances();
                List<DBInstance> instanceList = response.dbInstances();
                int listSize = instanceList.size();
                didFind = false;
                int index = 1;
                for (DBInstance instance : instanceList) {
                    instanceARN = instance.dbInstanceArn();
                    if (instanceARN.compareTo(clusterDBARN) == 0) {
                        System.out.println(clusterDBARN + " still exists");
                        didFind = true;
                    }
                    if ((index == listSize) && (!didFind)) {
                        // Went through the entire list and did not find the
database ARN.
                        isDataDel = true;
                    }
                    Thread.sleep(sleepTime * 1000);
                    index++;
                }
            }

            DeleteDbClusterParameterGroupRequest clusterParameterGroupRequest =
DeleteDbClusterParameterGroupRequest
                    .builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .build();


rdsClient.deleteDBClusterParameterGroup(clusterParameterGroupRequest);
            System.out.println(dbClusterGroupName + " was deleted.");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「DeleteDBClusterParameterGroup」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
@Throws(InterruptedException::class)
suspend fun deleteDBClusterGroup(dbClusterGroupName: String, clusterDBARN:
 String) {
    var isDataDel = false
    var didFind: Boolean
    var instanceARN: String

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        // Make sure that the database has been deleted.
        while (!isDataDel) {
            val response = rdsClient.describeDbInstances()
            val instanceList = response.dbInstances
            val listSize = instanceList?.size
            isDataDel = false
            didFind = false
            var index = 1
            if (instanceList != null) {
                for (instance in instanceList) {
                    instanceARN = instance.dbInstanceArn.toString()
                    if (instanceARN.compareTo(clusterDBARN) == 0) {
                        println("$clusterDBARN still exists")
                        didFind = true
                    }
                    if (index == listSize && !didFind) {
                        // Went through the entire list and did not find the
 database ARN.
                        isDataDel = true
                    }
                    delay(slTime * 1000)
                    index++
                }
            }
```

```
        }
        val clusterParameterGroupRequest = DeleteDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dbClusterGroupName
        }

        rdsClient.deleteDbClusterParameterGroup(clusterParameterGroupRequest)
        println("$dbClusterGroupName was deleted.")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「DeleteDBClusterParameterGroup」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)
```

```
    def delete_parameter_group(self, parameter_group_name):
        """
        Deletes a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to delete.
        :return: Data about the parameter group.
        """
        try:
            response = self.rds_client.delete_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name
            )
        except ClientError as err:
            logger.error(
                "Couldn't delete parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」で「DeleteDBClusterParameterGroup」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];
```

```
        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
```

```
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
```

```
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
```

```
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn delete_db_cluster_parameter_group(
        &self,
        name: &str,
    ) -> Result<DeleteDbClusterParameterGroupOutput,
 SdkError<DeleteDBClusterParameterGroupError>>
    {
        self.inner
            .delete_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
```

```
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
```

```
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
```

```
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });
```

```
    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.get(0), Some(ScenarioError {message, context: _}) if
 message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「DeleteDBClusterParameterGroup」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **DeleteDBInstance** を使用する

以下のコード例は、DeleteDBInstance の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- DB クラスターの開始方法

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
dbInstanceIdentifier)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DeleteDBInstance」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DeleteDBInstanceRequest request;
            request.SetDBInstanceIdentifier(dbInstanceIdentifier);
            request.SetSkipFinalSnapshot(true);
            request.SetDeleteAutomatedBackups(true);

            Aws::RDS::Model::DeleteDBInstanceOutcome outcome =
                    client.DeleteDBInstance(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB instance deletion has started."
                          << std::endl;
                instanceDeleting = true;
                std::cout
                        << "Waiting for DB instance to delete before deleting the
parameter group."
                        << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBInstance. "
                          << outcome.GetError().GetMessage()
```

```
                        << std::endl;
                result = false;
            }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「DeleteDBInstance」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteInstance deletes a DB instance.
func (clusters *DbClusters) DeleteInstance(instanceName string) error {
 _, err := clusters.AuroraClient.DeleteDBInstance(context.TODO(),
 &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      true,
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「DeleteDBInstance」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   public static void deleteDatabaseInstance(RdsClient rdsClient, String
dbInstanceIdentifier) {
       try {
           DeleteDbInstanceRequest deleteDbInstanceRequest =
DeleteDbInstanceRequest.builder()
                   .dbInstanceIdentifier(dbInstanceIdentifier)
                   .deleteAutomatedBackups(true)
                   .skipFinalSnapshot(true)
                   .build();

           DeleteDbInstanceResponse response =
rdsClient.deleteDBInstance(deleteDbInstanceRequest);
           System.out.println("The status of the database is " +
response.dbInstance().dbInstanceStatus());

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「DeleteDBInstance」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun deleteDBInstance(dbInstanceIdentifierVal: String) {
    val deleteDbInstanceRequest = DeleteDbInstanceRequest {
        dbInstanceIdentifier = dbInstanceIdentifierVal
        deleteAutomatedBackups = true
        skipFinalSnapshot = true
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.deleteDbInstance(deleteDbInstanceRequest)
        print("The status of the database is
 ${response.dbInstance?.dbInstanceStatus}")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「DeleteDBInstance」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""
```

```
    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_db_instance(self, instance_id):
        """
        Deletes a DB instance.

        :param instance_id: The ID of the DB instance to delete.
        :return: Data about the deleted DB instance.
        """
        try:
            response = self.rds_client.delete_db_instance(
                DBInstanceIdentifier=instance_id,
                SkipFinalSnapshot=True,
                DeleteAutomatedBackups=True,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't delete DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「DeleteDBInstance」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
```

```
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
```

```
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
```

```
        }

        // Delete the DB cluster parameter group.
 rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
 DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn delete_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DeleteDbInstanceOutput, SdkError<DeleteDBInstanceError>> {
        self.inner
            .delete_db_instance()
            .db_instance_identifier(instance_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }

#[tokio::test]
```

```
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
```

```
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
```

```
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
```

```
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.get(0), Some(ScenarioError {message, context: _}) if
message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
message == "Failed to check cluster state during deletion");
    });
```

```
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「DeleteDBInstance」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **DescribeDBClusterParameterGroups** を使用する

以下のコード例は、DescribeDBClusterParameterGroups の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- DB クラスターの開始方法

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</
param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
 DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeDBClusterParameterGroups」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::DescribeDBClusterParameterGroupsRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);

        Aws::RDS::Model::DescribeDBClusterParameterGroupsOutcome outcome =
```

```
                client.DescribeDBClusterParameterGroups(request);

        if (outcome.IsSuccess()) {
            std::cout << "DB cluster parameter group named '" <<
                      CLUSTER_PARAMETER_GROUP_NAME << "' already exists." <<
 std::endl;
            dbParameterGroupFamily =
 outcome.GetResult().GetDBClusterParameterGroups()
[0].GetDBParameterGroupFamily();
        }

        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameterGroups. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「DescribeDBClusterParameterGroups」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetParameterGroup gets a DB cluster parameter group by name.
func (clusters *DbClusters) GetParameterGroup(parameterGroupName string) (
```

```
 *types.DBClusterParameterGroup, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterParameterGroups(
  context.TODO(), &rds.DescribeDBClusterParameterGroupsInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
 } else {
  return &output.DBClusterParameterGroups[0], err
 }
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「DescribeDBClusterParameterGroups」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   public static void describeDbClusterParameterGroups(RdsClient rdsClient,
String dbClusterGroupName) {
       try {
           DescribeDbClusterParameterGroupsRequest groupsRequest =
DescribeDbClusterParameterGroupsRequest.builder()
                   .dbClusterParameterGroupName(dbClusterGroupName)
                   .maxRecords(20)
```

```
                    .build();

            List<DBClusterParameterGroup> groups =
rdsClient.describeDBClusterParameterGroups(groupsRequest)
                    .dbClusterParameterGroups();
            for (DBClusterParameterGroup group : groups) {
                System.out.println("The group name is " +
group.dbClusterParameterGroupName());
                System.out.println("The group ARN is " +
group.dbClusterParameterGroupArn());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「DescribeDBClusterParameterGroups」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun describeDbClusterParameterGroups(dbClusterGroupName: String?) {
    val groupsRequest = DescribeDbClusterParameterGroupsRequest {
        dbClusterParameterGroupName = dbClusterGroupName
        maxRecords = 20
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbClusterParameterGroups(groupsRequest)
        response.dbClusterParameterGroups?.forEach { group ->
```

```
            println("The group name is ${group.dbClusterParameterGroupName}")
            println("The group ARN is ${group.dbClusterParameterGroupArn}")
        }
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「DescribeDBClusterParameterGroups」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_parameter_group(self, parameter_group_name):
        """
```

```
        Gets a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to retrieve.
        :return: The requested parameter group.
        """
        try:
            response = self.rds_client.describe_db_cluster_parameter_groups(
                DBClusterParameterGroupName=parameter_group_name
            )
            parameter_group = response["DBClusterParameterGroups"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBParameterGroupNotFound":
                logger.info("Parameter group %s does not exist.",
parameter_group_name)
            else:
                logger.error(
                    "Couldn't get parameter group %s. Here's why: %s: %s",
                    parameter_group_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return parameter_group
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「DescribeDBClusterParameterGroups」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **DescribeDBClusterParameters** を使用する

以下のコード例は、DescribeDBClusterParameters の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- DB クラスターの開始方法

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
        var paramList = new List<Parameter>();

        DescribeDBClusterParametersResponse response;
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await
_amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
        }
        while (response.Marker is not null);

        return paramList;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeDBClusterParameters」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets DB parameters using the 'DescribeDBClusterParameters' api.
/*!
 \sa getDBClusterParameters()
 \param parameterGroupName: The name of the cluster parameter group.
 \param namePrefix: Prefix string to filter results by parameter name.
 \param source: A source such as 'user', ignored if empty.
 \param parametersResult: Vector of 'Parameter' objects returned by the routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBClusterParameters(const Aws::String
 &parameterGroupName,
                                            const Aws::String &namePrefix,
                                            const Aws::String &source,

 Aws::Vector<Aws::RDS::Model::Parameter> &parametersResult,
                                            const Aws::RDS::RDSClient &client) {
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeDBClusterParametersRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
```

```
        if (!marker.empty()) {
            request.SetMarker(marker);
        }
        if (!source.empty()) {
            request.SetSource(source);
        }

        Aws::RDS::Model::DescribeDBClusterParametersOutcome outcome =
                client.DescribeDBClusterParameters(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::Parameter> &parameters =
                    outcome.GetResult().GetParameters();
            for (const Aws::RDS::Model::Parameter &parameter: parameters) {
                if (!namePrefix.empty()) {
                    if (parameter.GetParameterName().find(namePrefix) == 0) {
                        parametersResult.push_back(parameter);
                    }
                }
                else {
                    parametersResult.push_back(parameter);
                }
            }

            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameters. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    return true;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「DescribeDBClusterParameters」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetParameters gets the parameters that are contained in a DB cluster parameter
 group.
func (clusters *DbClusters) GetParameters(parameterGroupName string, source
 string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBClusterParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator :=
 rds.NewDescribeDBClusterParametersPaginator(clusters.AuroraClient,
  &rds.DescribeDBClusterParametersInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Source:                      aws.String(source),
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(context.TODO())
  if err != nil {
   log.Printf("Couldn't get paramaeters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
```

```
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「DescribeDBClusterParameters」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   public static void describeDbClusterParameters(RdsClient rdsClient, String
dbCLusterGroupName, int flag) {
       try {
           DescribeDbClusterParametersRequest dbParameterGroupsRequest;
           if (flag == 0) {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .build();
           } else {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .source("user")
                       .build();
           }

           DescribeDbClusterParametersResponse response = rdsClient
                   .describeDBClusterParameters(dbParameterGroupsRequest);
           List<Parameter> dbParameters = response.parameters();
           String paraName;
           for (Parameter para : dbParameters) {
               // Only print out information about either auto_increment_offset
or
```

```
                // auto_increment_increment.
                paraName = para.parameterName();
                if ((paraName.compareTo("auto_increment_offset") == 0)
                        || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                    System.out.println("*** The parameter name is  " + paraName);
                    System.out.println("*** The parameter value is  " +
para.parameterValue());
                    System.out.println("*** The parameter data type is " +
para.dataType());
                    System.out.println("*** The parameter description is " +
para.description());
                    System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
                }
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「DescribeDBClusterParameters」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun describeDbClusterParameters(dbCLusterGroupName: String?, flag: Int) {
    val dbParameterGroupsRequest: DescribeDbClusterParametersRequest
    dbParameterGroupsRequest = if (flag == 0) {
        DescribeDbClusterParametersRequest {
```

```
            dbClusterParameterGroupName = dbCLusterGroupName
        }
    } else {
        DescribeDbClusterParametersRequest {
            dbClusterParameterGroupName = dbCLusterGroupName
            source = "user"
        }
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeDbClusterParameters(dbParameterGroupsRequest)
        response.parameters?.forEach { para ->
            // Only print out information about either auto_increment_offset or
 auto_increment_increment.
            val paraName = para.parameterName
            if (paraName != null) {
                if (paraName.compareTo("auto_increment_offset") == 0 ||
 paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    println("*** The parameter value is  ${para.parameterValue}")
                    println("*** The parameter data type is ${para.dataType}")
                    println("*** The parameter description is
 ${para.description}")
                    println("*** The parameter allowed values  is
 ${para.allowedValues}")
                }
            }
        }
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「DescribeDBClusterParameters」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_parameters(self, parameter_group_name, name_prefix="", source=None):
        """
        Gets the parameters that are contained in a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to query.
        :param name_prefix: When specified, the retrieved list of parameters is
 filtered
                            to contain only parameters that start with this
 prefix.
        :param source: When specified, only parameters from this source are
 retrieved.
                       For example, a source of 'user' retrieves only parameters
 that
```

```
                        were set by a user.
        :return: The list of requested parameters.
        """
        try:
            kwargs = {"DBClusterParameterGroupName": parameter_group_name}
            if source is not None:
                kwargs["Source"] = source
            parameters = []
            paginator =
self.rds_client.get_paginator("describe_db_cluster_parameters")
            for page in paginator.paginate(**kwargs):
                parameters += [
                    p
                    for p in page["Parameters"]
                    if p["ParameterName"].startswith(name_prefix)
                ]
        except ClientError as err:
            logger.error(
                "Couldn't get parameters for %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return parameters
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「DescribeDBClusterParameters」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   // Get the parameter group. rds.DescribeDbClusterParameterGroups
   // Get parameters in the group. This is a long list so you will have to
paginate. Find the auto_increment_offset and auto_increment_increment parameters
(by ParameterName). rds.DescribeDbClusterParameters
   // Parse the ParameterName, Description, and AllowedValues values and display
them.
   pub async fn cluster_parameters(&self) ->
Result<Vec<AuroraScenarioParameter>, ScenarioError> {
       let parameters_output = self
           .rds
           .describe_db_cluster_parameters(DB_CLUSTER_PARAMETER_GROUP_NAME)
           .await;

       if let Err(err) = parameters_output {
           return Err(ScenarioError::new(
               format!("Failed to retrieve parameters for
{DB_CLUSTER_PARAMETER_GROUP_NAME}"),
               &err,
           ));
       }

       let parameters = parameters_output
           .unwrap()
           .into_iter()
           .flat_map(|p| p.parameters.unwrap_or_default().into_iter())
           .filter(|p|
FILTER_PARAMETER_NAMES.contains(p.parameter_name().unwrap_or_default()))
           .map(AuroraScenarioParameter::from)
           .collect::<Vec<_>>();

       Ok(parameters)
   }

   pub async fn describe_db_cluster_parameters(
       &self,
       name: &str,
   ) -> Result<Vec<DescribeDbClusterParametersOutput>,
SdkError<DescribeDBClusterParametersError>>
   {
       self.inner
           .describe_db_cluster_parameters()
           .db_cluster_parameter_group_name(name)
           .into_paginator()
```

```
            .send()
            .try_collect()
            .await
    }

#[tokio::test]
async fn test_scenario_cluster_parameters() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Ok(vec![DescribeDbClusterParametersOutput::builder()
                .parameters(Parameter::builder().parameter_name("a").build())
                .parameters(Parameter::builder().parameter_name("b").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("c").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("d").build())
                .build()])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());

    let params = scenario.cluster_parameters().await.expect("cluster params");
    let names: Vec<String> = params.into_iter().map(|p| p.name).collect();
    assert_eq!(
        names,
        vec!["auto_increment_offset", "auto_increment_increment"]
    );
}

#[tokio::test]
async fn test_scenario_cluster_parameters_error() {
```

```
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClusterParametersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_cluster_parameters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let params = scenario.cluster_parameters().await;
    assert_matches!(params, Err(ScenarioError { message, context: _ }) if message
 == "Failed to retrieve parameters for RustSDKCodeExamplesDBParameterGroup");
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「DescribeDBClusterParameters」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **DescribeDBClusterSnapshots** を使用する

以下のコード例は、DescribeDBClusterSnapshots の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- DB クラスターの開始方法

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Return a list of DB snapshots for a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>List of DB snapshots.</returns>
    public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
    {
        var results = new List<DBClusterSnapshot>();

        DescribeDBClusterSnapshotsResponse response;
        DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
            results.AddRange(response.DBClusterSnapshots);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeDBClusterSnapshots」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DescribeDBClusterSnapshotsRequest request;
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::DescribeDBClusterSnapshotsOutcome outcome =
                    client.DescribeDBClusterSnapshots(request);

            if (outcome.IsSuccess()) {
                snapshot = outcome.GetResult().GetDBClusterSnapshots()[0];
            }
            else {
                std::cerr << "Error with Aurora::DescribeDBClusterSnapshots. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「DescribeDBClusterSnapshots」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetClusterSnapshot gets a DB cluster snapshot.
func (clusters *DbClusters) GetClusterSnapshot(snapshotName string)
 (*types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterSnapshots(context.TODO(),
  &rds.DescribeDBClusterSnapshotsInput{
   DBClusterSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return &output.DBClusterSnapshots[0], nil
 }
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「DescribeDBClusterSnapshots」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   public static void waitForSnapshotReady(RdsClient rdsClient, String
dbSnapshotIdentifier,
           String dbInstanceClusterIdentifier) {
       try {
           boolean snapshotReady = false;
           String snapshotReadyStr;
           System.out.println("Waiting for the snapshot to become available.");

           DescribeDbClusterSnapshotsRequest snapshotsRequest =
DescribeDbClusterSnapshotsRequest.builder()
                   .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                   .dbClusterIdentifier(dbInstanceClusterIdentifier)
                   .build();

           while (!snapshotReady) {
               DescribeDbClusterSnapshotsResponse response =
rdsClient.describeDBClusterSnapshots(snapshotsRequest);
               List<DBClusterSnapshot> snapshotList =
response.dbClusterSnapshots();
               for (DBClusterSnapshot snapshot : snapshotList) {
                   snapshotReadyStr = snapshot.status();
                   if (snapshotReadyStr.contains("available")) {
                       snapshotReady = true;
                   } else {
                       System.out.println(".");
                       Thread.sleep(sleepTime * 5000);
                   }
               }
           }

           System.out.println("The Snapshot is available!");
```

```
        } catch (RdsException | InterruptedException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「DescribeDBClusterSnapshots」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun waitSnapshotReady(dbSnapshotIdentifier: String?,
 dbInstanceClusterIdentifier: String?) {
    var snapshotReady = false
    var snapshotReadyStr: String
    println("Waiting for the snapshot to become available.")

    val snapshotsRequest = DescribeDbClusterSnapshotsRequest {
        dbClusterSnapshotIdentifier = dbSnapshotIdentifier
        dbClusterIdentifier = dbInstanceClusterIdentifier
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        while (!snapshotReady) {
            val response = rdsClient.describeDbClusterSnapshots(snapshotsRequest)
            val snapshotList = response.dbClusterSnapshots
            if (snapshotList != null) {
                for (snapshot in snapshotList) {
                    snapshotReadyStr = snapshot.status.toString()
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true
                    } else {
```

```
                        println(".")
                        delay(slTime * 5000)
                    }
                }
            }
        }
    }
    println("The Snapshot is available!")
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「DescribeDBClusterSnapshots」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)
```

```
    def get_cluster_snapshot(self, snapshot_id):
        """
        Gets a DB cluster snapshot.

        :param snapshot_id: The ID of the snapshot to retrieve.
        :return: The retrieved snapshot.
        """
        try:
            response = self.rds_client.describe_db_cluster_snapshots(
                DBClusterSnapshotIdentifier=snapshot_id
            )
            snapshot = response["DBClusterSnapshots"][0]
        except ClientError as err:
            logger.error(
                "Couldn't get DB cluster snapshot %s. Here's why: %s: %s",
                snapshot_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「DescribeDBClusterSnapshots」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **DescribeDBClusters** を使用する

以下のコード例は、DescribeDBClusters の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- DB クラスターの開始方法

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Returns a list of DB clusters.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
cluster.</param>
    /// <returns>List of DB clusters.</returns>
    public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
dbClusterIdentifier = null)
    {
        var results = new List<DBCluster>();

        DescribeDBClustersResponse response;
        DescribeDBClustersRequest request = new DescribeDBClustersRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClustersAsync(request);
            results.AddRange(response.DBClusters);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeDBClusters」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

//! Routine which gets a DB cluster description.
/*!
 \sa describeDBCluster()
 \param dbClusterIdentifier: A DB cluster identifier.
 \param clusterResult: The 'DBCluster' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBCluster(const Aws::String &dbClusterIdentifier,
                                       Aws::RDS::Model::DBCluster &clusterResult,
                                       const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBClustersRequest request;
    request.SetDBClusterIdentifier(dbClusterIdentifier);

    Aws::RDS::Model::DescribeDBClustersOutcome outcome =
            client.DescribeDBClusters(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        clusterResult = outcome.GetResult().GetDBClusters()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_CLUSTER_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::GDescribeDBClusters. "
                  << outcome.GetError().GetMessage()
```

```
                    << std::endl;
    }
        // This example does not log an error if the DB cluster does not exist.
        // Instead, clusterResult is set to empty.
    else {
        clusterResult = Aws::RDS::Model::DBCluster();
    }

    return result;

}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「DescribeDBClusters」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetDbCluster gets data about an Aurora DB cluster.
func (clusters *DbClusters) GetDbCluster(clusterName string) (*types.DBCluster,
 error) {
 output, err := clusters.AuroraClient.DescribeDBClusters(context.TODO(),
  &rds.DescribeDBClustersInput{
   DBClusterIdentifier: aws.String(clusterName),
  })
 if err != nil {
```

```
  var notFoundError *types.DBClusterNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB cluster %v does not exist.\n", clusterName)
   err = nil
  } else {
   log.Printf("Couldn't get DB cluster %v: %v\n", clusterName, err)
  }
  return nil, err
 } else {
  return &output.DBClusters[0], err
 }
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「DescribeDBClusters」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   public static void describeDbClusterParameters(RdsClient rdsClient, String
dbCLusterGroupName, int flag) {
       try {
           DescribeDbClusterParametersRequest dbParameterGroupsRequest;
           if (flag == 0) {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .build();
           } else {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
```

```
                        .source("user")
                        .build();
            }

            DescribeDbClusterParametersResponse response = rdsClient
                    .describeDBClusterParameters(dbParameterGroupsRequest);
            List<Parameter> dbParameters = response.parameters();
            String paraName;
            for (Parameter para : dbParameters) {
                // Only print out information about either auto_increment_offset
or
                // auto_increment_increment.
                paraName = para.parameterName();
                if ((paraName.compareTo("auto_increment_offset") == 0)
                        || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                    System.out.println("*** The parameter name is  " + paraName);
                    System.out.println("*** The parameter value is  " +
para.parameterValue());
                    System.out.println("*** The parameter data type is " +
para.dataType());
                    System.out.println("*** The parameter description is " +
para.description());
                    System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
                }
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「DescribeDBClusters」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun describeDbClusterParameters(dbCLusterGroupName: String?, flag: Int) {
    val dbParameterGroupsRequest: DescribeDbClusterParametersRequest
    dbParameterGroupsRequest = if (flag == 0) {
        DescribeDbClusterParametersRequest {
            dbClusterParameterGroupName = dbCLusterGroupName
        }
    } else {
        DescribeDbClusterParametersRequest {
            dbClusterParameterGroupName = dbCLusterGroupName
            source = "user"
        }
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeDbClusterParameters(dbParameterGroupsRequest)
        response.parameters?.forEach { para ->
            // Only print out information about either auto_increment_offset or
 auto_increment_increment.
            val paraName = para.parameterName
            if (paraName != null) {
                if (paraName.compareTo("auto_increment_offset") == 0 ||
 paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    println("*** The parameter value is  ${para.parameterValue}")
                    println("*** The parameter data type is ${para.dataType}")
                    println("*** The parameter description is
 ${para.description}")
                    println("*** The parameter allowed values  is
 ${para.allowedValues}")
                }
            }
```

```
        }
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「DescribeDBClusters」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_db_cluster(self, cluster_name):
        """
        Gets data about an Aurora DB cluster.
```

```
        :param cluster_name: The name of the DB cluster to retrieve.
        :return: The retrieved DB cluster.
        """
        try:
            response = self.rds_client.describe_db_clusters(
                DBClusterIdentifier=cluster_name
            )
            cluster = response["DBClusters"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBClusterNotFoundFault":
                logger.info("Cluster %s does not exist.", cluster_name)
            else:
                logger.error(
                    "Couldn't verify the existence of DB cluster %s. Here's why:
%s: %s",
                    cluster_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return cluster
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「DescribeDBClusters」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
```

```
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_ENGINE,
               self.engine_version.as_deref().expect("engine version"),
               self.username.as_deref().expect("username"),
               self.password
                   .replace(SecretString::new("".to_string()))
                   .expect("password"),
           )
           .await;
       if let Err(err) = create_db_cluster {
           return Err(ScenarioError::new(
               "Failed to create DB Cluster with cluster group",
               &err,
           ));
       }

       self.db_cluster_identifier = create_db_cluster
           .unwrap()
           .db_cluster
           .and_then(|c| c.db_cluster_identifier);

       if self.db_cluster_identifier.is_none() {
```

```
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;
```

```
            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }
```

```
            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn describe_db_clusters(
        &self,
        id: &str,
    ) -> Result<DescribeDbClustersOutput, SdkError<DescribeDBClustersError>> {
        self.inner
            .describe_db_clusters()
            .db_cluster_identifier(id)
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
```

```
.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
```

```
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
```

```
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));
```

```
    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
```

```
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
```

```
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()
```

```
 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「DescribeDBClusters」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **DescribeDBEngineVersions** を使用する

以下のコード例は、DescribeDBEngineVersions の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- DB クラスターの開始方法

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family
name.</param>
    /// <returns>A list of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>>
DescribeDBEngineVersionsForEngineAsync(string engine,
        string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeDBEngineVersions」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets available DB engine versions for an engine name and
//! an optional parameter group family.
/*!
 \sa getDBEngineVersions()
 \param engineName: A DB engine name.
 \param parameterGroupFamily: A parameter group family name, ignored if empty.
 \param engineVersionsResult: Vector of 'DBEngineVersion' objects returned by the
 routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBEngineVersions(const Aws::String &engineName,
                                         const Aws::String &parameterGroupFamily,

 Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersionsResult,
                                         const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBEngineVersionsRequest request;
    request.SetEngine(engineName);
    if (!parameterGroupFamily.empty()) {
        request.SetDBParameterGroupFamily(parameterGroupFamily);
    }

    engineVersionsResult.clear();
    Aws::String marker; // The marker is used for pagination.
    do {
```

```
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeDBEngineVersionsOutcome outcome =
                client.DescribeDBEngineVersions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersions =
                    outcome.GetResult().GetDBEngineVersions();

            engineVersionsResult.insert(engineVersionsResult.end(),
                                        engineVersions.begin(),
 engineVersions.end());
            marker  = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBEngineVersionsRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    } while (!marker.empty());

    return true;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「DescribeDBEngineVersions」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
func (clusters *DbClusters) GetEngineVersions(engine string, parameterGroupFamily
 string) (
 []types.DBEngineVersion, error) {
 output, err := clusters.AuroraClient.DescribeDBEngineVersions(context.TODO(),
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
 }
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「DescribeDBEngineVersions」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    public static void describeDBEngines(RdsClient rdsClient) {
```

```
        try {
            DescribeDbEngineVersionsRequest engineVersionsRequest =
DescribeDbEngineVersionsRequest.builder()
                    .engine("aurora-mysql")
                    .defaultOnly(true)
                    .maxRecords(20)
                    .build();

            DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(engineVersionsRequest);
            List<DBEngineVersion> engines = response.dbEngineVersions();

            // Get all DBEngineVersion objects.
            for (DBEngineVersion engineOb : engines) {
                System.out.println("The name of the DB parameter group family for
the database engine is "
                        + engineOb.dbParameterGroupFamily());
                System.out.println("The name of the database engine " +
engineOb.engine());
                System.out.println("The version number of the database engine " +
engineOb.engineVersion());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「DescribeDBEngineVersions」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// Get a list of allowed engine versions.
suspend fun getAllowedClusterEngines(dbParameterGroupFamilyVal: String?) {
    val versionsRequest = DescribeDbEngineVersionsRequest {
        dbParameterGroupFamily = dbParameterGroupFamilyVal
        engine = "aurora-mysql"
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(versionsRequest)
        response.dbEngineVersions?.forEach { dbEngine ->
            println("The engine version is ${dbEngine.engineVersion}")
            println("The engine description is ${dbEngine.dbEngineDescription}")
        }
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「DescribeDBEngineVersions」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
```

```
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_engine_versions(self, engine, parameter_group_family=None):
        """
        Gets database engine versions that are available for the specified engine
        and parameter group family.

        :param engine: The database engine to look up.
        :param parameter_group_family: When specified, restricts the returned
list of
                                       engine versions to those that are
compatible with
                                       this parameter group family.
        :return: The list of database engine versions.
        """
        try:
            kwargs = {"Engine": engine}
            if parameter_group_family is not None:
                kwargs["DBParameterGroupFamily"] = parameter_group_family
            response = self.rds_client.describe_db_engine_versions(**kwargs)
            versions = response["DBEngineVersions"]
        except ClientError as err:
            logger.error(
                "Couldn't get engine versions for %s. Here's why: %s: %s",
                engine,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return versions
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「DescribeDBEngineVersions」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   // Get available engine families for Aurora MySql.
rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
   pub async fn get_engines(&self) -> Result<HashMap<String, Vec<String>>,
ScenarioError> {
       let describe_db_engine_versions =
self.rds.describe_db_engine_versions(DB_ENGINE).await;
       trace!(versions=?describe_db_engine_versions, "full list of versions");

       if let Err(err) = describe_db_engine_versions {
           return Err(ScenarioError::new(
               "Failed to retrieve DB Engine Versions",
               &err,
           ));
       };

       let version_count = describe_db_engine_versions
           .as_ref()
           .map(|o| o.db_engine_versions().len())
           .unwrap_or_default();
       info!(version_count, "got list of versions");

       // Create a map of engine families to their available versions.
       let mut versions = HashMap::<String, Vec<String>>::new();
       describe_db_engine_versions
           .unwrap()
           .db_engine_versions()
           .iter()
           .filter_map(
               |v| match (&v.db_parameter_group_family, &v.engine_version) {
                   (Some(family), Some(version)) => Some((family.clone(),
version.clone())),
```

```
                    _ => None,
                },
            )
            .for_each(|(family, version)|
 versions.entry(family).or_default().push(version));

        Ok(versions)
    }

    pub async fn describe_db_engine_versions(
        &self,
        engine: &str,
    ) -> Result<DescribeDbEngineVersionsOutput,
 SdkError<DescribeDBEngineVersionsError>> {
        self.inner
            .describe_db_engine_versions()
            .engine(engine)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_get_engines() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Ok(DescribeDbEngineVersionsOutput::builder()
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1a")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1b")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
```

```
                        .db_parameter_group_family("f2")
                        .engine_version("f2a")
                        .build(),
                )
                .db_engine_versions(DbEngineVersion::builder().build())
                .build())
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;

    assert_eq!(
        versions_map,
        Ok(HashMap::from([
            ("f1".into(), vec!["f1a".into(), "f1b".into()]),
            ("f2".into(), vec!["f2a".into()])
        ]))
    );
}

#[tokio::test]
async fn test_scenario_get_engines_failed() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBEngineVersionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_engine_versions error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;
    assert_matches!(
        versions_map,
```

```
        Err(ScenarioError { message, context: _ }) if message == "Failed to
 retrieve DB Engine Versions"
    );
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「DescribeDBEngineVersions」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **DescribeDBInstances** を使用する

以下のコード例は、DescribeDBInstances の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- DB クラスターの開始方法

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
    /// <returns>List of DB instances.</returns>
    public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
dbInstanceIdentifier = null)
```

```
    {
        var results = new List<DBInstance>();
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });
        // Get the entire list using the paginator.
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「DescribeDBInstances」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets a DB instance description.
/*!
 \sa describeDBCluster()
 \param dbInstanceIdentifier: A DB instance identifier.
 \param instanceResult: The 'DBInstance' object containing the description.
```

```
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBInstance(const Aws::String &dbInstanceIdentifier,
                                        Aws::RDS::Model::DBInstance
 &instanceResult,
                                        const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBInstancesRequest request;
    request.SetDBInstanceIdentifier(dbInstanceIdentifier);

    Aws::RDS::Model::DescribeDBInstancesOutcome outcome =
            client.DescribeDBInstances(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        instanceResult = outcome.GetResult().GetDBInstances()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_INSTANCE_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::DescribeDBInstances. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB instance does not exist.
        // Instead, instanceResult is set to empty.
    else {
        instanceResult = Aws::RDS::Model::DBInstance();
    }

    return result;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「DescribeDBInstances」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetInstance gets data about a DB instance.
func (clusters *DbClusters) GetInstance(instanceName string) (
 *types.DBInstance, error) {
 output, err := clusters.AuroraClient.DescribeDBInstances(context.TODO(),
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「DescribeDBInstances」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   // Waits until the database instance is available.
   public static void waitForInstanceReady(RdsClient rdsClient, String
dbClusterIdentifier) {
       boolean instanceReady = false;
       String instanceReadyStr;
       System.out.println("Waiting for instance to become available.");
       try {
           DescribeDbClustersRequest instanceRequest =
DescribeDbClustersRequest.builder()
                   .dbClusterIdentifier(dbClusterIdentifier)
                   .build();

           while (!instanceReady) {
               DescribeDbClustersResponse response =
rdsClient.describeDBClusters(instanceRequest);
               List<DBCluster> clusterList = response.dbClusters();
               for (DBCluster cluster : clusterList) {
                   instanceReadyStr = cluster.status();
                   if (instanceReadyStr.contains("available")) {
                       instanceReady = true;
                   } else {
                       System.out.print(".");
                       Thread.sleep(sleepTime * 1000);
                   }
               }
           }
           System.out.println("Database cluster is available!");

       } catch (RdsException | InterruptedException e) {
           System.err.println(e.getMessage());
           System.exit(1);
       }
```

```
    }
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「DescribeDBInstances」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
suspend fun waitDBAuroraInstanceReady(dbInstanceIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")
    val instanceRequest = DescribeDbInstancesRequest {
        dbInstanceIdentifier = dbInstanceIdentifierVal
    }

    var endpoint = ""
    RdsClient { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
            val response = rdsClient.describeDbInstances(instanceRequest)
            response.dbInstances?.forEach { instance ->
                instanceReadyStr = instance.dbInstanceStatus.toString()
                if (instanceReadyStr.contains("available")) {
                    endpoint = instance.endpoint?.address.toString()
                    instanceReady = true
                } else {
                    print(".")
                    delay(sleepTime * 1000)
                }
            }
        }
    }
```

```
    println("Database instance is available! The connection endpoint is
 $endpoint")
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「DescribeDBInstances」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_db_instance(self, instance_id):
        """
        Gets data about a DB instance.
```

```
        :param instance_id: The ID of the DB instance to retrieve.
        :return: The retrieved DB instance.
        """
        try:
            response = self.rds_client.describe_db_instances(
                DBInstanceIdentifier=instance_id
            )
            db_inst = response["DBInstances"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBInstanceNotFound":
                logger.info("Instance %s does not exist.", instance_id)
            else:
                logger.error(
                    "Couldn't get DB instance %s. Here's why: %s: %s",
                    instance_id,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return db_inst
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「DescribeDBInstances」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
```

```
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
```

```
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
```

```
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
```

```
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn describe_db_instances(
        &self,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner.describe_db_instances().send().await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));
```

```
    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });
```

```
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
```

```
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
```

```
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.get(0), Some(ScenarioError {message, context: _}) if
 message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「DescribeDBInstances」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **DescribeOrderableDBInstanceOptions** を使用する

以下のコード例は、DescribeOrderableDBInstanceOptions の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- DB クラスターの開始方法

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    /// <summary>
    /// Get a list of orderable DB instance options for a specific
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
    public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string
engineVersion)
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
        // Get the entire list using the paginator.
```

```
        await foreach (var instanceOptions in
 paginateInstanceOptions.OrderableDBInstanceOptions)
        {
            results.Add(instanceOptions);
        }
        return results;
    }
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の「DescribeOrderableDBInstanceOptions」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets available DB instance classes, displays the list
//! to the user, and returns the user selection.
/*!
 \sa chooseDBInstanceClass()
 \param engineName: The DB engine name.
 \param engineVersion: The DB engine version.
 \param dbInstanceClass: String for DB instance class chosen by the user.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::chooseDBInstanceClass(const Aws::String &engine,
                                           const Aws::String &engineVersion,
```

```
                                            Aws::String &dbInstanceClass,
                                            const Aws::RDS::RDSClient &client) {
   std::vector<Aws::String> instanceClasses;
   Aws::String marker; // The marker is used for pagination.
   do {
       Aws::RDS::Model::DescribeOrderableDBInstanceOptionsRequest request;
       request.SetEngine(engine);
       request.SetEngineVersion(engineVersion);
       if (!marker.empty()) {
           request.SetMarker(marker);
       }

       Aws::RDS::Model::DescribeOrderableDBInstanceOptionsOutcome outcome =
               client.DescribeOrderableDBInstanceOptions(request);

       if (outcome.IsSuccess()) {
           const Aws::Vector<Aws::RDS::Model::OrderableDBInstanceOption>
&options =
                   outcome.GetResult().GetOrderableDBInstanceOptions();
           for (const Aws::RDS::Model::OrderableDBInstanceOption &option:
options) {
               const Aws::String &instanceClass = option.GetDBInstanceClass();
               if (std::find(instanceClasses.begin(), instanceClasses.end(),
                             instanceClass) == instanceClasses.end()) {
                   instanceClasses.push_back(instanceClass);
               }
           }
           marker = outcome.GetResult().GetMarker();
       }
       else {
           std::cerr << "Error with Aurora::DescribeOrderableDBInstanceOptions.
"
                     << outcome.GetError().GetMessage()
                     << std::endl;
           return false;
       }
   } while (!marker.empty());

   std::cout << "The available DB instance classes for your database engine
are:"
             << std::endl;
   for (int i = 0; i < instanceClasses.size(); ++i) {
       std::cout << "   " << i + 1 << ": " << instanceClasses[i] << std::endl;
   }
```

```
    int choice = askQuestionForIntRange(
            "Which DB instance class do you want to use? ",
            1, static_cast<int>(instanceClasses.size()));
    dbInstanceClass = instanceClasses[choice - 1];
    return true;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「DescribeOrderableDBInstanceOptions」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (clusters *DbClusters) GetOrderableInstances(engine string, engineVersion
 string) (
 []types.OrderableDBInstanceOption, error) {

 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instances []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(clusters.AuroraClient,
```

```
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(context.TODO())
  if err != nil {
   log.Printf("Couldn't get orderable DB instances: %v\n", err)
   break
  } else {
   instances = append(instances, output.OrderableDBInstanceOptions...)
  }
 }
 return instances, err
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の「DescribeOrderableDBInstanceOptions」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
    public static void describeDBEngines(RdsClient rdsClient) {
        try {
            DescribeDbEngineVersionsRequest engineVersionsRequest =
 DescribeDbEngineVersionsRequest.builder()
                    .engine("aurora-mysql")
                    .defaultOnly(true)
                    .maxRecords(20)
                    .build();
```

```
            DescribeDbEngineVersionsResponse response =
 rdsClient.describeDBEngineVersions(engineVersionsRequest);
            List<DBEngineVersion> engines = response.dbEngineVersions();

            // Get all DBEngineVersion objects.
            for (DBEngineVersion engineOb : engines) {
                System.out.println("The name of the DB parameter group family for
 the database engine is "
                        + engineOb.dbParameterGroupFamily());
                System.out.println("The name of the database engine " +
 engineOb.engine());
                System.out.println("The version number of the database engine " +
 engineOb.engineVersion());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の「DescribeOrderableDBInstanceOptions」を参照してください。

PowerShell

Tools for PowerShell

例 1: この例は、AWS リージョン 内で特定の DB インスタンスクラスをサポートする DB エンジンのバージョンを一覧表示します。

```
$params = @{
  Engine = 'aurora-postgresql'
  DBInstanceClass = 'db.r5.large'
  Region = 'us-east-1'
}
Get-RDSOrderableDBInstanceOption @params
```

例 2: この例は、AWS リージョン 内で特定の DB エンジンのバージョンをサポートする DB インスタンスクラスを一覧表示します。

```
$params = @{
  Engine = 'aurora-postgresql'
  EngineVersion = '13.6'
  Region = 'us-east-1'
}
Get-RDSOrderableDBInstanceOption @params
```

- API の詳細については、「AWS Tools for PowerShell Cmdlet リファレンス」の「DescribeOrderableDBInstanceOptions」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_orderable_instances(self, db_engine, db_engine_version):
        """
```

```
        Gets DB instance options that can be used to create DB instances that are
        compatible with a set of specifications.

        :param db_engine: The database engine that must be supported by the DB
instance.
        :param db_engine_version: The engine version that must be supported by
the DB instance.
        :return: The list of DB instance options that can be used to create a
compatible DB instance.
        """
        try:
            inst_opts = []
            paginator = self.rds_client.get_paginator(
                "describe_orderable_db_instance_options"
            )
            for page in paginator.paginate(
                Engine=db_engine, EngineVersion=db_engine_version
            ):
                inst_opts += page["OrderableDBInstanceOptions"]
        except ClientError as err:
            logger.error(
                "Couldn't get orderable DB instances. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return inst_opts
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「DescribeOrderableDBInstanceOptions」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   pub async fn get_instance_classes(&self) -> Result<Vec<String>,
ScenarioError> {
       let describe_orderable_db_instance_options_items = self
           .rds
           .describe_orderable_db_instance_options(
               DB_ENGINE,
               self.engine_version
                   .as_ref()
                   .expect("engine version for db instance options")
                   .as_str(),
           )
           .await;

       describe_orderable_db_instance_options_items
           .map(|options| {
               options
                   .iter()
                   .map(|o|
o.db_instance_class().unwrap_or_default().to_string())
                   .collect::<Vec<String>>()
           })
           .map_err(|err| ScenarioError::new("Could not get available instance
classes", &err))
   }

   pub async fn describe_orderable_db_instance_options(
       &self,
       engine: &str,
       engine_version: &str,
   ) -> Result<Vec<OrderableDbInstanceOption>,
SdkError<DescribeOrderableDBInstanceOptionsError>>
   {
```

```
        self.inner
            .describe_orderable_db_instance_options()
            .engine(engine)
            .engine_version(engine_version)
            .into_paginator()
            .items()
            .send()
            .try_collect()
            .await
    }

#[tokio::test]
async fn test_scenario_get_instance_classes() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Ok(vec![
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t2")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t3")
                    .build(),
            ])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario
        .set_engine("aurora-mysql", "aurora-mysql8.0")
```

```
        .await
        .expect("set engine");

    let instance_classes = scenario.get_instance_classes().await;

    assert_eq!(
        instance_classes,
        Ok(vec!["t1".into(), "t2".into(), "t3".into()])
    );
}

#[tokio::test]
async fn test_scenario_get_instance_classes_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Err(SdkError::service_error(

 DescribeOrderableDBInstanceOptionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_orderable_db_instance_options_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_family = Some("aurora-mysql".into());
    scenario.engine_version = Some("aurora-mysql8.0".into());

    let instance_classes = scenario.get_instance_classes().await;

    assert_matches!(
        instance_classes,
        Err(ScenarioError {message, context: _}) if message == "Could not get
 available instance classes"
    );
}
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の「DescribeOrderableDBInstanceOptions」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK または CLI で **ModifyDBClusterParameterGroup** を使用する

以下のコード例は、ModifyDBClusterParameterGroup の使用方法を示しています。

アクション例は、より大きなプログラムからのコードの抜粋であり、コンテキスト内で実行する必要があります。次のコード例で、このアクションのコンテキストを確認できます。

- DB クラスターの開始方法

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

```
    /// <summary>
    /// Modify the specified integer parameters with new values from user input.
    /// </summary>
    /// <param name="groupName">The group name for the parameters.</param>
    /// <param name="parameters">The list of integer parameters to modify.</
param>
    /// <param name="newValue">Optional int value to set for parameters.</param>
    /// <returns>The name of the group that was modified.</returns>
    public async Task<string> ModifyIntegerParametersInGroupAsync(string
 groupName, List<Parameter> parameters, int newValue = 0)
    {
        foreach (var p in parameters)
```

```
        {
            if (p.IsModifiable && p.DataType == "integer")
            {
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the
allowed values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    int.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
        }

        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
            DBClusterParameterGroupName = groupName,
        };

        var result = await
_amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }
```

- API の詳細については、AWS SDK for .NET API リファレンスの「ModifyDBClusterParameterGroup」を参照してください。

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::ModifyDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetParameters(updateParameters);

        Aws::RDS::Model::ModifyDBClusterParameterGroupOutcome outcome =
                client.ModifyDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
 modified."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::ModifyDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の「ModifyDBClusterParameterGroup」を参照してください。

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
type DbClusters struct {
 AuroraClient *rds.Client
```

```
}



// UpdateParameters updates parameters in a named DB cluster parameter group.
func (clusters *DbClusters) UpdateParameters(parameterGroupName string, params
 []types.Parameter) error {
 _, err := clusters.AuroraClient.ModifyDBClusterParameterGroup(context.TODO(),
  &rds.ModifyDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Parameters:                  params,
  })
 if err != nil {
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

- API の詳細については、AWS SDK for Go API リファレンスの「ModifyDBClusterParameterGroup」を参照してください。

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   public static void describeDbClusterParameterGroups(RdsClient rdsClient,
String dbClusterGroupName) {
       try {
           DescribeDbClusterParameterGroupsRequest groupsRequest =
DescribeDbClusterParameterGroupsRequest.builder()
                   .dbClusterParameterGroupName(dbClusterGroupName)
```

```
                    .maxRecords(20)
                    .build();

            List<DBClusterParameterGroup> groups =
 rdsClient.describeDBClusterParameterGroups(groupsRequest)
                    .dbClusterParameterGroups();
            for (DBClusterParameterGroup group : groups) {
                System.out.println("The group name is " +
 group.dbClusterParameterGroupName());
                System.out.println("The group ARN is " +
 group.dbClusterParameterGroupArn());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- API の詳細については、AWS SDK for Java 2.x API リファレンスの「ModifyDBClusterParameterGroup」を参照してください。

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
// Modify the auto_increment_offset parameter.
suspend fun modifyDBClusterParas(dClusterGroupName: String?) {
    val parameter1 = Parameter {
        parameterName = "auto_increment_offset"
        applyMethod = ApplyMethod.fromValue("immediate")
        parameterValue = "5"
    }
```

```
    val paraList = ArrayList<Parameter>()
    paraList.add(parameter1)
    val groupRequest = ModifyDbClusterParameterGroupRequest {
        dbClusterParameterGroupName = dClusterGroupName
        parameters = paraList
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.modifyDbClusterParameterGroup(groupRequest)
        println("The parameter group ${response.dbClusterParameterGroupName} was
 successfully modified")
    }
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「ModifyDBClusterParameterGroup」を参照してください。

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
```

```
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def update_parameters(self, parameter_group_name, update_parameters):
        """
        Updates parameters in a custom DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to update.
        :param update_parameters: The parameters to update in the group.
        :return: Data about the modified parameter group.
        """
        try:
            response = self.rds_client.modify_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                Parameters=update_parameters,
            )
        except ClientError as err:
            logger.error(
                "Couldn't update parameters in %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- API の詳細については、AWS SDK for Python (Boto3) API リファレンスの「ModifyDBClusterParameterGroup」を参照してください。

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
   // Modify both the auto_increment_offset and auto_increment_increment
parameters in one call in the custom parameter group. Set their ParameterValue
fields to a new allowable value. rds.ModifyDbClusterParameterGroup.
   pub async fn update_auto_increment(
       &self,
       offset: u8,
       increment: u8,
   ) -> Result<(), ScenarioError> {
       let modify_db_cluster_parameter_group = self
           .rds
           .modify_db_cluster_parameter_group(
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               vec![
                   Parameter::builder()
                       .parameter_name("auto_increment_offset")
                       .parameter_value(format!("{offset}"))
                       .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                       .build(),
                   Parameter::builder()
                       .parameter_name("auto_increment_increment")
                       .parameter_value(format!("{increment}"))
                       .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                       .build(),
               ],
           )
           .await;

       if let Err(error) = modify_db_cluster_parameter_group {
           return Err(ScenarioError::new(
               "Failed to modify cluster parameter group",
               &error,
           ));
```

```
        }

        Ok(())
    }

    pub async fn modify_db_cluster_parameter_group(
        &self,
        name: &str,
        parameters: Vec<Parameter>,
    ) -> Result<ModifyDbClusterParameterGroupOutput,
 SdkError<ModifyDBClusterParameterGroupError>>
    {
        self.inner
            .modify_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .set_parameters(Some(parameters))
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_update_auto_increment() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .withf(|name, params| {
            assert_eq!(name, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(
                params,
                &vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value("10")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value("20")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ]
            );
            true
```

```
        })
        .return_once(|_, _|
 Ok(ModifyDbClusterParameterGroupOutput::builder().build()));

    let scenario = AuroraScenario::new(mock_rds);

    scenario
        .update_auto_increment(10, 20)
        .await
        .expect("update auto increment");
}

#[tokio::test]
async fn test_scenario_update_auto_increment_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .return_once(|_, _| {
            Err(SdkError::service_error(

 ModifyDBClusterParameterGroupError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "modify_db_cluster_parameter_group_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let update = scenario.update_auto_increment(10, 20).await;
    assert_matches!(update, Err(ScenarioError { message, context: _}) if message
 == "Failed to modify cluster parameter group");
}
```

- API の詳細については、AWS SDK for Kotlin API リファレンスの「ModifyDBClusterParameterGroup」を参照してください。

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用した Aurora のシナリオ

以下のコード例は、Aurora で AWS SDK を使用した一般的なシナリオを実装する方法を示しています。これらのシナリオは、Aurora 内で複数の関数を呼び出すことによって特定のタスクを実行する方法を示しています。それぞれのシナリオには、GitHub へのリンクがあり、コードを設定および実行する方法についての説明が記載されています。

例

- AWS SDK を使用して Aurora DB クラスターの使用を開始する

AWS SDK を使用して Aurora DB クラスターの使用を開始する

次のコード例は、以下を実行する方法を示しています。

- カスタム Aurora DB クラスターパラメータグループを作成し、パラメータ値を設定します。
- パラメータグループを使用する DB クラスターを作成する
- データベースを含む DB インスタンスを作成します。
- DB クラスターのスナップショットを作成して、リソースをクリーンアップします。

.NET

AWS SDK for .NET

> **Note**
>
> GitHub には、その他のリソースもあります。AWS コード例リポジトリ で全く同じ例を見つけて、設定と実行の方法を確認してください。

コマンドプロンプトからインタラクティブのシナリオを実行します。

```
using Amazon.RDS;
```

```
using Amazon.RDS.Model;
using AuroraActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace AuroraScenario;

/// <summary>
/// Scenario for Amazon Aurora examples.
/// </summary>
public class AuroraScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    This .NET example performs the following tasks:
    1.  Return a list of the available DB engine families for Aurora MySql using
 the DescribeDBEngineVersionsAsync method.
    2.  Select an engine family and create a custom DB cluster parameter group
 using the CreateDBClusterParameterGroupAsync method.
    3.  Get the parameter group using the DescribeDBClusterParameterGroupsAsync
 method.
    4.  Get some parameters in the group using the
 DescribeDBClusterParametersAsync method.
    5.  Parse and display some parameters in the group.
    6.  Modify the auto_increment_offset and auto_increment_increment parameters
        using the ModifyDBClusterParameterGroupAsync method.
    7.  Get and display the updated parameters using the
 DescribeDBClusterParametersAsync method with a source of "user".
    8.  Get a list of allowed engine versions using the
 DescribeDBEngineVersionsAsync method.
    9.  Create an Aurora DB cluster that contains a MySql database and uses the
 parameter group.
        using the CreateDBClusterAsync method.
    10. Wait for the DB cluster to be ready using the DescribeDBClustersAsync
 method.
    11. Display and select from a list of instance classes available for the
 selected engine and version
        using the paginated DescribeOrderableDBInstanceOptions method.
```

```
   12. Create a database instance in the cluster using the CreateDBInstanceAsync
method.
   13. Wait for the DB instance to be ready using the DescribeDBInstances
method.
   14. Display the connection endpoint string for the new DB cluster.
   15. Create a snapshot of the DB cluster using the
CreateDBClusterSnapshotAsync method.
   16. Wait for DB snapshot to be ready using the
DescribeDBClusterSnapshotsAsync method.
   17. Delete the DB instance using the DeleteDBInstanceAsync method.
   18. Delete the DB cluster using the DeleteDBClusterAsync method.
   19. Wait for DB cluster to be deleted using the DescribeDBClustersAsync
methods.
   20. Delete the cluster parameter group using the
DeleteDBClusterParameterGroupAsync.
   */

   private static readonly string sepBar = new('-', 80);
   private static AuroraWrapper auroraWrapper = null!;
   private static ILogger logger = null!;
   private static readonly string engine = "aurora-mysql";
   static async Task Main(string[] args)
   {
       // Set up dependency injection for the Amazon Relational Database Service
(Amazon RDS).
       using var host = Host.CreateDefaultBuilder(args)
           .ConfigureLogging(logging =>
               logging.AddFilter("System", LogLevel.Debug)
                   .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                   .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
           .ConfigureServices((_, services) =>
               services.AddAWSService<IAmazonRDS>()
                   .AddTransient<AuroraWrapper>()
           )
           .Build();

       logger = LoggerFactory.Create(builder =>
       {
           builder.AddConsole();
       }).CreateLogger<AuroraScenario>();

       auroraWrapper = host.Services.GetRequiredService<AuroraWrapper>();
```

```
        Console.WriteLine(sepBar);
        Console.WriteLine(
            "Welcome to the Amazon Aurora: get started with DB clusters
example.");
        Console.WriteLine(sepBar);

        DBClusterParameterGroup parameterGroup = null!;
        DBCluster? newCluster = null;
        DBInstance? newInstance = null;

        try
        {
            var parameterGroupFamily = await ChooseParameterGroupFamilyAsync();

            parameterGroup = await
CreateDBParameterGroupAsync(parameterGroupFamily);

            var parameters = await
DescribeParametersInGroupAsync(parameterGroup.DBClusterParameterGroupName,
                new List<string> { "auto_increment_offset",
"auto_increment_increment" });

            await
ModifyParametersAsync(parameterGroup.DBClusterParameterGroupName, parameters);

            await
DescribeUserSourceParameters(parameterGroup.DBClusterParameterGroupName);

            var engineVersionChoice = await
ChooseDBEngineVersionAsync(parameterGroupFamily);

            var newClusterIdentifier = "Example-Cluster-" + DateTime.Now.Ticks;

            newCluster = await CreateNewCluster
            (
                parameterGroup,
                engine,
                engineVersionChoice.EngineVersion,
                newClusterIdentifier
            );

            var instanceClassChoice = await ChooseDBInstanceClass(engine,
engineVersionChoice.EngineVersion);
```

```
            var newInstanceIdentifier = "Example-Instance-" + DateTime.Now.Ticks;

            newInstance = await CreateNewInstance(
                newClusterIdentifier,
                engine,
                engineVersionChoice.EngineVersion,
                instanceClassChoice.DBInstanceClass,
                newInstanceIdentifier
            );

            DisplayConnectionString(newCluster!);
            await CreateSnapshot(newCluster!);
            await CleanupResources(newInstance, newCluster, parameterGroup);

            Console.WriteLine("Scenario complete.");
            Console.WriteLine(sepBar);
        }
        catch (Exception ex)

        {
            await CleanupResources(newInstance, newCluster, parameterGroup);
            logger.LogError(ex, "There was a problem executing the scenario.");
        }
    }

    /// <summary>
    /// Choose the Aurora DB parameter group family from a list of available
options.
    /// </summary>
    /// <returns>The selected parameter group family.</returns>
    public static async Task<string> ChooseParameterGroupFamilyAsync()
    {
        Console.WriteLine(sepBar);
        // 1. Get a list of available engines.
        var engines = await
auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine);

        Console.WriteLine($"1. The following is a list of available DB parameter
group families for engine {engine}:");

        var parameterGroupFamilies =
            engines.GroupBy(e => e.DBParameterGroupFamily).ToList();
        for (var i = 1; i <= parameterGroupFamilies.Count; i++)
```

```
        {
            var parameterGroupFamily = parameterGroupFamilies[i - 1];
            // List the available parameter group families.
            Console.WriteLine(
                $"\t{i}. Family: {parameterGroupFamily.Key}");
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > parameterGroupFamilies.Count)
        {
            Console.WriteLine("2. Select an available DB parameter group family
by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }
        var parameterGroupFamilyChoice = parameterGroupFamilies[choiceNumber -
1];
        Console.WriteLine(sepBar);
        return parameterGroupFamilyChoice.Key;
    }

    /// <summary>
    /// Create and get information on a DB parameter group.
    /// </summary>
    /// <param name="dbParameterGroupFamily">The DBParameterGroupFamily for the
new DB parameter group.</param>
    /// <returns>The new DBParameterGroup.</returns>
    public static async Task<DBClusterParameterGroup>
CreateDBParameterGroupAsync(string dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"2. Create new DB parameter group with family
{dbParameterGroupFamily}:");

        var parameterGroup = await
auroraWrapper.CreateCustomClusterParameterGroupAsync(
            dbParameterGroupFamily,
            "ExampleParameterGroup-" + DateTime.Now.Ticks,
            "New example parameter group");

        var groupInfo =
            await
auroraWrapper.DescribeCustomDBClusterParameterGroupAsync(parameterGroup.DBClusterParamet
```

```
        Console.WriteLine(
            $"3. New DB parameter group created: \n\t{groupInfo?.Description}, \n
\tARN {groupInfo?.DBClusterParameterGroupName}");
        Console.WriteLine(sepBar);
        return parameterGroup;
    }

    /// <summary>
    /// Get and describe parameters from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">The name of the DBParameterGroup.</
param>
    /// <param name="parameterNames">Optional specific names of parameters to
 describe.</param>
    /// <returns>The list of requested parameters.</returns>
    public static async Task<List<Parameter>>
 DescribeParametersInGroupAsync(string parameterGroupName, List<string>?
 parameterNames = null)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("4. Get some parameters from the group.");
        Console.WriteLine(sepBar);

        var parameters =
            await
 auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName);

        var matchingParameters =
            parameters.Where(p => parameterNames == null ||
 parameterNames.Contains(p.ParameterName)).ToList();

        Console.WriteLine("5. Parameter information:");
        matchingParameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);

        return matchingParameters;
    }
```

```
    /// <summary>
    /// Modify a parameter from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
    /// <param name="parameters">The parameters to modify.</param>
    /// <returns>Async task.</returns>
    public static async Task ModifyParametersAsync(string parameterGroupName,
 List<Parameter> parameters)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("6. Modify some parameters in the group.");

        await
 auroraWrapper.ModifyIntegerParametersInGroupAsync(parameterGroupName,
 parameters);

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Describe the user source parameters in the group.
    /// </summary>
    /// <param name="parameterGroupName">The name of the DBParameterGroup.</
param>
    /// <returns>Async task.</returns>
    public static async Task DescribeUserSourceParameters(string
 parameterGroupName)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("7. Describe updated user source parameters in the
 group.");

        var parameters =
            await
 auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName,
 "user");

        parameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));
```

```
        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Choose a DB engine version.
    /// </summary>
    /// <param name="dbParameterGroupFamily">DB parameter group family for engine
choice.</param>
    /// <returns>The selected engine version.</returns>
    public static async Task<DBEngineVersion> ChooseDBEngineVersionAsync(string
dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed engines.
        var allowedEngines =
            await auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine,
dbParameterGroupFamily);

        Console.WriteLine($"Available DB engine versions for parameter group
family {dbParameterGroupFamily}:");
        int i = 1;
        foreach (var version in allowedEngines)
        {
            Console.WriteLine(
                $"\t{i}. {version.DBEngineVersionDescription}.");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedEngines.Count)
        {
            Console.WriteLine("8. Select an available DB engine version by
entering a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var engineChoice = allowedEngines[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return engineChoice;
    }

    /// <summary>
    /// Create a new RDS DB cluster.
```

```
    /// </summary>
    /// <param name="parameterGroup">Parameter group to use for the DB cluster.</
param>
    /// <param name="engineName">Engine to use for the DB cluster.</param>
    /// <param name="engineVersion">Engine version to use for the DB cluster.</
param>
    /// <param name="clusterIdentifier">Cluster identifier to use for the DB
 cluster.</param>
    /// <returns>The new DB cluster.</returns>
    public static async Task<DBCluster?> CreateNewCluster(DBClusterParameterGroup
 parameterGroup,
        string engineName, string engineVersion, string clusterIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"9. Create a new DB cluster with identifier
 {clusterIdentifier}.");

        DBCluster newCluster;
        var clusters = await auroraWrapper.DescribeDBClustersPagedAsync();
        var isClusterCreated = clusters.Any(i => i.DBClusterIdentifier ==
 clusterIdentifier);

        if (isClusterCreated)
        {
            Console.WriteLine("Cluster already created.");
            newCluster = clusters.First(i => i.DBClusterIdentifier ==
 clusterIdentifier);
        }
        else
        {
            Console.WriteLine("Enter an admin username:");
            var username = Console.ReadLine();

            Console.WriteLine("Enter an admin password:");
            var password = Console.ReadLine();

            newCluster = await auroraWrapper.CreateDBClusterWithAdminAsync(
                "ExampleDatabase",
                clusterIdentifier,
                parameterGroup.DBClusterParameterGroupName,
                engineName,
                engineVersion,
                username!,
                password!
```

```
            );

            Console.WriteLine("10. Waiting for DB cluster to be ready...");
            while (newCluster.Status != "available")
            {
                Console.Write(".");
                Thread.Sleep(5000);
                clusters = await
 auroraWrapper.DescribeDBClustersPagedAsync(clusterIdentifier);
                newCluster = clusters.First();
            }
        }

        Console.WriteLine(sepBar);
        return newCluster;
    }

    /// <summary>
    /// Choose a DB instance class for a particular engine and engine version.
    /// </summary>
    /// <param name="engine">DB engine for DB instance choice.</param>
    /// <param name="engineVersion">DB engine version for DB instance choice.</
param>
    /// <returns>The selected orderable DB instance option.</returns>
    public static async Task<OrderableDBInstanceOption>
 ChooseDBInstanceClass(string engine, string engineVersion)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed DB instance classes.
        var allowedInstances =
            await
 auroraWrapper.DescribeOrderableDBInstanceOptionsPagedAsync(engine,
 engineVersion);

        Console.WriteLine($"Available DB instance classes for engine {engine} and
 version {engineVersion}:");
        int i = 1;


        foreach (var instance in allowedInstances)
        {
            Console.WriteLine(
                $"\t{i}. Instance class: {instance.DBInstanceClass} (storage type
 {instance.StorageType})");
```

```
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedInstances.Count)
        {
            Console.WriteLine("11. Select an available DB instance class by
 entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var instanceChoice = allowedInstances[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return instanceChoice;
    }

    /// <summary>
    /// Create a new DB instance.
    /// </summary>
    /// <param name="engineName">Engine to use for the DB instance.</param>
    /// <param name="engineVersion">Engine version to use for the DB instance.</
param>
    /// <param name="instanceClass">Instance class to use for the DB instance.</
param>
    /// <param name="instanceIdentifier">Instance identifier to use for the DB
 instance.</param>
    /// <returns>The new DB instance.</returns>
    public static async Task<DBInstance?> CreateNewInstance(
        string clusterIdentifier,
        string engineName,
        string engineVersion,
        string instanceClass,
        string instanceIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"12. Create a new DB instance with identifier
 {instanceIdentifier}.");
        bool isInstanceReady = false;
        DBInstance newInstance;
        var instances = await auroraWrapper.DescribeDBInstancesPagedAsync();
        isInstanceReady = instances.FirstOrDefault(i =>
            i.DBInstanceIdentifier == instanceIdentifier)?.DBInstanceStatus ==
 "available";
```

```
        if (isInstanceReady)
        {
            Console.WriteLine("Instance already created.");
            newInstance = instances.First(i => i.DBInstanceIdentifier ==
 instanceIdentifier);
        }
        else
        {

            newInstance = await auroraWrapper.CreateDBInstanceInClusterAsync(
                clusterIdentifier,
                instanceIdentifier,
                engineName,
                engineVersion,
                instanceClass
            );

            Console.WriteLine("13. Waiting for DB instance to be ready...");
            while (!isInstanceReady)
            {
                Console.Write(".");
                Thread.Sleep(5000);
                instances = await
 auroraWrapper.DescribeDBInstancesPagedAsync(instanceIdentifier);
                isInstanceReady = instances.FirstOrDefault()?.DBInstanceStatus ==
 "available";
                newInstance = instances.First();
            }
        }

        Console.WriteLine(sepBar);
        return newInstance;
    }

    /// <summary>
    /// Display a connection string for an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">The DB cluster to use to get a connection string.</
param>
    public static void DisplayConnectionString(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Display the connection string.
```

```
        Console.WriteLine("14. New DB cluster connection string: ");
        Console.WriteLine(
            $"\n{engine} -h {cluster.Endpoint} -P {cluster.Port} "
            + $"-u {cluster.MasterUsername} -p [YOUR PASSWORD]\n");

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Create a snapshot from an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">DB cluster to use when creating a snapshot.</param>
    /// <returns>The snapshot object.</returns>
    public static async Task<DBClusterSnapshot> CreateSnapshot(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Create a snapshot.
        Console.WriteLine($"15. Creating snapshot from DB cluster
{cluster.DBClusterIdentifier}.");
        var snapshot = await
auroraWrapper.CreateClusterSnapshotByIdentifierAsync(
            cluster.DBClusterIdentifier,
            "ExampleSnapshot-" + DateTime.Now.Ticks);

        // Wait for the snapshot to be available.
        bool isSnapshotReady = false;

        Console.WriteLine($"16. Waiting for snapshot to be ready...");
        while (!isSnapshotReady)
        {
            Console.Write(".");
            Thread.Sleep(5000);
            var snapshots =
                await
auroraWrapper.DescribeDBClusterSnapshotsByIdentifierAsync(cluster.DBClusterIdentifier);
            isSnapshotReady = snapshots.FirstOrDefault()?.Status == "available";
            snapshot = snapshots.First();
        }

        Console.WriteLine(
            $"Snapshot {snapshot.DBClusterSnapshotIdentifier} status is
{snapshot.Status}.");
        Console.WriteLine(sepBar);
        return snapshot;
```

```
    }

    /// <summary>
    /// Clean up resources from the scenario.
    /// </summary>
    /// <param name="newInstance">The instance to clean up.</param>
    /// <param name="newCluster">The cluster to clean up.</param>
    /// <param name="parameterGroup">The parameter group to clean up.</param>
    /// <returns>Async Task.</returns>
    private static async Task CleanupResources(
        DBInstance? newInstance,
        DBCluster? newCluster,
        DBClusterParameterGroup? parameterGroup)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        if (newInstance is not null && GetYesNoResponse($"\tClean up instance
{newInstance.DBInstanceIdentifier}? (y/n)"))
        {
            // Delete the DB instance.
            Console.WriteLine($"17. Deleting the DB instance
{newInstance.DBInstanceIdentifier}.");
            await
auroraWrapper.DeleteDBInstanceByIdentifierAsync(newInstance.DBInstanceIdentifier);
        }

        if (newCluster is not null && GetYesNoResponse($"\tClean up cluster
{newCluster.DBClusterIdentifier}? (y/n)"))
        {
            // Delete the DB cluster.
            Console.WriteLine($"18. Deleting the DB cluster
{newCluster.DBClusterIdentifier}.");
            await
auroraWrapper.DeleteDBClusterByIdentifierAsync(newCluster.DBClusterIdentifier);

            // Wait for the DB cluster to delete.
            Console.WriteLine($"19. Waiting for the DB cluster to delete...");
            bool isClusterDeleted = false;

            while (!isClusterDeleted)
            {
                Console.Write(".");
                Thread.Sleep(5000);
```

```
                var cluster = await auroraWrapper.DescribeDBClustersPagedAsync();
                isClusterDeleted = cluster.All(i => i.DBClusterIdentifier !=
 newCluster.DBClusterIdentifier);
            }

            Console.WriteLine("DB cluster deleted.");
        }

        if (parameterGroup is not null && GetYesNoResponse($"\tClean up parameter
 group? (y/n)"))
        {
            Console.WriteLine($"20. Deleting the DB parameter group
 {parameterGroup.DBClusterParameterGroupName}.");
            await
 auroraWrapper.DeleteClusterParameterGroupByNameAsync(parameterGroup.DBClusterParameterGr
            Console.WriteLine("Parameter group deleted.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</
param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
                       ynResponse.Equals("y",
                           StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
```

Aurora アクションを管理するためにシナリオによって呼び出されるラッパーメソッド。

```
using Amazon.RDS;
using Amazon.RDS.Model;
```

```
namespace AuroraActions;

/// <summary>
/// Wrapper for the Amazon Aurora cluster client operations.
/// </summary>
public class AuroraWrapper
{
    private readonly IAmazonRDS _amazonRDS;
    public AuroraWrapper(IAmazonRDS amazonRDS)
    {
        _amazonRDS = amazonRDS;
    }

    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family
 name.</param>
    /// <returns>A list of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>>
 DescribeDBEngineVersionsForEngineAsync(string engine,
        string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
    }

    /// <summary>
    /// Create a custom cluster parameter group.
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
    /// <param name="description">A description for the new parameter group.</
param>
    /// <returns>The new parameter group object.</returns>
```

```
    public async Task<DBClusterParameterGroup>
CreateCustomClusterParameterGroupAsync(
        string parameterGroupFamily,
        string groupName,
        string description)
    {
        var request = new CreateDBClusterParameterGroupRequest
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
            Description = description,
        };

        var response = await
_amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }

    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
        var paramList = new List<Parameter>();

        DescribeDBClusterParametersResponse response;
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await
_amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
```

```
        }
        while (response.Marker is not null);

        return paramList;
    }

    /// <summary>
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</
param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
 DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }

    /// <summary>
    /// Modify the specified integer parameters with new values from user input.
    /// </summary>
    /// <param name="groupName">The group name for the parameters.</param>
    /// <param name="parameters">The list of integer parameters to modify.</
param>
    /// <param name="newValue">Optional int value to set for parameters.</param>
    /// <returns>The name of the group that was modified.</returns>
    public async Task<string> ModifyIntegerParametersInGroupAsync(string
 groupName, List<Parameter> parameters, int newValue = 0)
    {
        foreach (var p in parameters)
        {
            if (p.IsModifiable && p.DataType == "integer")
            {
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the
 allowed values {p.AllowedValues} ");
```

```
                    var choice = Console.ReadLine();
                    int.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
        }

        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
            DBClusterParameterGroupName = groupName,
        };

        var result = await
_amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }


    /// <summary>
    /// Get a list of orderable DB instance options for a specific
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
    public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string
engineVersion)
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
        // Get the entire list using the paginator.
        await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
        {
```

```
            results.Add(instanceOptions);
        }
        return results;
    }

    /// <summary>
    /// Delete a particular parameter group by name.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterParameterGroupByNameAsync(string
groupName)
    {
        var request = new DeleteDBClusterParameterGroupRequest
        {
            DBClusterParameterGroupName = groupName,
        };

        var response = await
_amazonRDS.DeleteDBClusterParameterGroupAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create a new cluster and database.
    /// </summary>
    /// <param name="dbName">The name of the new database.</param>
    /// <param name="clusterIdentifier">The identifier of the cluster.</param>
    /// <param name="parameterGroupName">The name of the parameter group.</param>
    /// <param name="dbEngine">The engine to use for the new cluster.</param>
    /// <param name="dbEngineVersion">The version of the engine to use.</param>
    /// <param name="adminName">The admin username.</param>
    /// <param name="adminPassword">The primary admin password.</param>
    /// <returns>The cluster object.</returns>
    public async Task<DBCluster> CreateDBClusterWithAdminAsync(
        string dbName,
        string clusterIdentifier,
        string parameterGroupName,
        string dbEngine,
        string dbEngineVersion,
        string adminName,
        string adminPassword)
    {
        var request = new CreateDBClusterRequest
```

```
        {
            DatabaseName = dbName,
            DBClusterIdentifier = clusterIdentifier,
            DBClusterParameterGroupName = parameterGroupName,
            Engine = dbEngine,
            EngineVersion = dbEngineVersion,
            MasterUsername = adminName,
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }

    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
    /// <returns>List of DB instances.</returns>
    public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
dbInstanceIdentifier = null)
    {
        var results = new List<DBInstance>();
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });
        // Get the entire list using the paginator.
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }

    /// <summary>
    /// Returns a list of DB clusters.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
cluster.</param>
    /// <returns>List of DB clusters.</returns>
```

```
    public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
dbClusterIdentifier = null)
    {
        var results = new List<DBCluster>();

        DescribeDBClustersResponse response;
        DescribeDBClustersRequest request = new DescribeDBClustersRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClustersAsync(request);
            results.AddRange(response.DBClusters);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
    /// with a particular set of properties. Use the action
DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
        // When creating the instance within a cluster, do not specify the name
or size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
```

```
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }

    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });

        return response.DBClusterSnapshot;
    }

    /// <summary>
    /// Return a list of DB snapshots for a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>List of DB snapshots.</returns>
    public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
    {
        var results = new List<DBClusterSnapshot>();

        DescribeDBClusterSnapshotsResponse response;
        DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
```

```
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
            results.AddRange(response.DBClusterSnapshots);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Delete a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>DB cluster object.</returns>
    public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
    {
        var response = await _amazonRDS.DeleteDBClusterAsync(
            new DeleteDBClusterRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                SkipFinalSnapshot = true
            });

        return response.DBCluster;
    }

    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
dbInstanceIdentifier)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
```

```
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
}
```

- API の詳細については、「AWS SDK for .NET API リファレンス」の以下のトピックを参照してください。
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeOrderableDBInstanceOptions
  - ModifyDBClusterParameterGroup

C++

SDK for C++

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Routine which creates an Amazon Aurora DB cluster and demonstrates several
 operations
//! on that cluster.
/*!
 \sa gettingStartedWithDBClusters()
 \param clientConfiguration: AWS client configuration.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::gettingStartedWithDBClusters(
        const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::RDS::RDSClient client(clientConfig);

    printAsterisksLine();
    std::cout << "Welcome to the Amazon Relational Database Service (Amazon
 Aurora)"
              << std::endl;
    std::cout << "get started with DB clusters demo." << std::endl;
    printAsterisksLine();

    std::cout << "Checking for an existing DB cluster parameter group named '" <<
              CLUSTER_PARAMETER_GROUP_NAME << "'." << std::endl;
    Aws::String dbParameterGroupFamily("Undefined");
    bool parameterGroupFound = true;
    {
        // 1. Check if the DB cluster parameter group already exists.
        Aws::RDS::Model::DescribeDBClusterParameterGroupsRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);

        Aws::RDS::Model::DescribeDBClusterParameterGroupsOutcome outcome =
                client.DescribeDBClusterParameterGroups(request);

        if (outcome.IsSuccess()) {
            std::cout << "DB cluster parameter group named '" <<
                      CLUSTER_PARAMETER_GROUP_NAME << "' already exists." <<
 std::endl;
            dbParameterGroupFamily =
 outcome.GetResult().GetDBClusterParameterGroups()
[0].GetDBParameterGroupFamily();
        }
```

```
        else if (outcome.GetError().GetErrorType() ==
                 Aws::RDS::RDSErrors::D_B_PARAMETER_GROUP_NOT_FOUND_FAULT) {
            std::cout << "DB cluster parameter group named '" <<
                      CLUSTER_PARAMETER_GROUP_NAME << "' does not exist." <<
std::endl;
            parameterGroupFound = false;
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameterGroups. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    if (!parameterGroupFound) {
        Aws::Vector<Aws::RDS::Model::DBEngineVersion> engineVersions;

        // 2. Get available parameter group families for the specified engine.
        if (!getDBEngineVersions(DB_ENGINE, NO_PARAMETER_GROUP_FAMILY,
                                 engineVersions, client)) {
            return false;
        }

        std::cout << "Getting available parameter group families for " <<
DB_ENGINE
                  << "."
                  << std::endl;
        std::vector<Aws::String> families;
        for (const Aws::RDS::Model::DBEngineVersion &version: engineVersions) {
            Aws::String family = version.GetDBParameterGroupFamily();
            if (std::find(families.begin(), families.end(), family) ==
                families.end()) {
                families.push_back(family);
                std::cout << "  " << families.size() << ": " << family <<
std::endl;
            }
        }

        int choice = askQuestionForIntRange("Which family do you want to use? ",
1,
                                            static_cast<int>(families.size()));
        dbParameterGroupFamily = families[choice - 1];
    }
```

```
    if (!parameterGroupFound) {
        // 3.  Create a DB cluster parameter group.
        Aws::RDS::Model::CreateDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetDBParameterGroupFamily(dbParameterGroupFamily);
        request.SetDescription("Example cluster parameter group.");

        Aws::RDS::Model::CreateDBClusterParameterGroupOutcome outcome =
                client.CreateDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
created."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    printAsterisksLine();
    std::cout << "Let's set some parameter values in your cluster parameter
group."
              << std::endl;

    Aws::Vector<Aws::RDS::Model::Parameter> autoIncrementParameters;
    // 4.  Get the parameters in the DB cluster parameter group.
    if (!getDBClusterParameters(CLUSTER_PARAMETER_GROUP_NAME,
AUTO_INCREMENT_PREFIX,
                                NO_SOURCE,
                                autoIncrementParameters,
                                client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
        return false;
    }

    Aws::Vector<Aws::RDS::Model::Parameter> updateParameters;

    for (Aws::RDS::Model::Parameter &autoIncParameter: autoIncrementParameters) {
        if (autoIncParameter.GetIsModifiable() &&
            (autoIncParameter.GetDataType() == "integer")) {
```

```
            std::cout << "The " << autoIncParameter.GetParameterName()
                      << " is described as: " <<
                      autoIncParameter.GetDescription() << "." << std::endl;
            if (autoIncParameter.ParameterValueHasBeenSet()) {
                std::cout << "The current value is "
                          << autoIncParameter.GetParameterValue()
                          << "." << std::endl;
            }
            std::vector<int> splitValues = splitToInts(
                    autoIncParameter.GetAllowedValues(), '-');
            if (splitValues.size() == 2) {
                int newValue = askQuestionForIntRange(
                        Aws::String("Enter a new value between ") +
                        autoIncParameter.GetAllowedValues() + ": ",
                        splitValues[0], splitValues[1]);
                autoIncParameter.SetParameterValue(std::to_string(newValue));
                updateParameters.push_back(autoIncParameter);

            }
            else {
                std::cerr << "Error parsing " <<
autoIncParameter.GetAllowedValues()
                          << std::endl;
            }
        }
    }

    {
        // 5.  Modify the auto increment parameters in the DB cluster parameter
group.
        Aws::RDS::Model::ModifyDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetParameters(updateParameters);

        Aws::RDS::Model::ModifyDBClusterParameterGroupOutcome outcome =
                client.ModifyDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
modified."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::ModifyDBClusterParameterGroup. "
```

```
                    << outcome.GetError().GetMessage()
                    << std::endl;
        }
    }

    std::cout
            << "You can get a list of parameters you've set by specifying a
source of 'user'."
            << std::endl;

    Aws::Vector<Aws::RDS::Model::Parameter> userParameters;
    // 6.  Display the modified parameters in the DB cluster parameter group.
    if (!getDBClusterParameters(CLUSTER_PARAMETER_GROUP_NAME, NO_NAME_PREFIX,
"user",
                                userParameters,
                                client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
        return false;
    }

    for (const auto &userParameter: userParameters) {
        std::cout << "  " << userParameter.GetParameterName() << ", " <<
                  userParameter.GetDescription() << ", parameter value - "
                  << userParameter.GetParameterValue() << std::endl;
    }

    printAsterisksLine();
    std::cout << "Checking for an existing DB Cluster." << std::endl;

    Aws::RDS::Model::DBCluster dbCluster;
    // 7.  Check if the DB cluster already exists.
    if (!describeDBCluster(DB_CLUSTER_IDENTIFIER, dbCluster, client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
        return false;
    }

    Aws::String engineVersionName;
    Aws::String engineName;
    if (dbCluster.DBClusterIdentifierHasBeenSet()) {
        std::cout << "The DB cluster already exists." << std::endl;
        engineVersionName = dbCluster.GetEngineVersion();
        engineName = dbCluster.GetEngine();

    }
```

```
   else {
       std::cout << "Let's create a DB cluster." << std::endl;
       const Aws::String administratorName = askQuestion(
               "Enter an administrator username for the database: ");
       const Aws::String administratorPassword = askQuestion(
               "Enter a password for the administrator (at least 8 characters):
");
       Aws::Vector<Aws::RDS::Model::DBEngineVersion> engineVersions;

       // 8.  Get a list of engine versions for the parameter group family.
       if (!getDBEngineVersions(DB_ENGINE, dbParameterGroupFamily,
engineVersions,
                                client)) {
           cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
           return false;
       }

       std::cout << "The available engines for your parameter group family are:"
                 << std::endl;

       int index = 1;
       for (const Aws::RDS::Model::DBEngineVersion &engineVersion:
engineVersions) {
           std::cout << "  " << index << ": " <<
engineVersion.GetEngineVersion()
                     << std::endl;
           ++index;
       }
       int choice = askQuestionForIntRange("Which engine do you want to use? ",
1,

static_cast<int>(engineVersions.size()));
       const Aws::RDS::Model::DBEngineVersion engineVersion =
engineVersions[choice -
                                                                              1];

       engineName = engineVersion.GetEngine();
       engineVersionName = engineVersion.GetEngineVersion();
       std::cout << "Creating a DB cluster named '" << DB_CLUSTER_IDENTIFIER
                 << "' and database '" << DB_NAME << "'.\n"
                 << "The DB cluster is configured to use your custom cluster
parameter group '"
                 << CLUSTER_PARAMETER_GROUP_NAME << "', and \n"
```

```
                  << "selected engine version " <<
engineVersion.GetEngineVersion()
                  << ".\nThis typically takes several minutes." << std::endl;

       Aws::RDS::Model::CreateDBClusterRequest request;
       request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
       request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
       request.SetEngine(engineName);
       request.SetEngineVersion(engineVersionName);
       request.SetMasterUsername(administratorName);
       request.SetMasterUserPassword(administratorPassword);

       Aws::RDS::Model::CreateDBClusterOutcome outcome =
               client.CreateDBCluster(request);

       if (outcome.IsSuccess()) {
           std::cout << "The DB cluster creation has started."
                     << std::endl;
       }
       else {
           std::cerr << "Error with Aurora::CreateDBCluster. "
                     << outcome.GetError().GetMessage()
                     << std::endl;
           cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
           return false;
       }
   }

   std::cout << "Waiting for the DB cluster to become available." << std::endl;

   int counter = 0;
   // 11. Wait for the DB cluster to become available.
   do {
       std::this_thread::sleep_for(std::chrono::seconds(1));
       ++counter;
       if (counter > 900) {
           std::cerr << "Wait for cluster to become available timed out ofter "
                     << counter
                     << " seconds." << std::endl;
           cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                            DB_CLUSTER_IDENTIFIER, "", client);
           return false;
       }
```

```
        dbCluster = Aws::RDS::Model::DBCluster();
        if (!describeDBCluster(DB_CLUSTER_IDENTIFIER, dbCluster, client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, "", client);
            return false;
        }

        if ((counter % 20) == 0) {
            std::cout << "Current DB cluster status is '"
                      << dbCluster.GetStatus()
                      << "' after " << counter << " seconds." << std::endl;
        }
    } while (dbCluster.GetStatus() != "available");

    if (dbCluster.GetStatus() == "available") {
        std::cout << "The DB cluster has been created." << std::endl;
    }

    printAsterisksLine();
    Aws::RDS::Model::DBInstance dbInstance;
    // 11.  Check if the DB instance already exists.
    if (!describeDBInstance(DB_INSTANCE_IDENTIFIER, dbInstance, client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER, "",
                         client);
        return false;
    }

    if (dbInstance.DbInstancePortHasBeenSet()) {
        std::cout << "The DB instance already exists." << std::endl;
    }
    else {
        std::cout << "Let's create a DB instance." << std::endl;

        Aws::String dbInstanceClass;
        // 12.  Get a list of instance classes.
        if (!chooseDBInstanceClass(engineName,
                                   engineVersionName,
                                   dbInstanceClass,
                                   client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER,
""",
                             client);
            return false;
        }
```

```
        std::cout << "Creating a DB instance named '" << DB_INSTANCE_IDENTIFIER
                  << "' with selected DB instance class '" << dbInstanceClass
                  << "'.\nThis typically takes several minutes." << std::endl;

        // 13. Create a DB instance.
        Aws::RDS::Model::CreateDBInstanceRequest request;
        request.SetDBInstanceIdentifier(DB_INSTANCE_IDENTIFIER);
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetEngine(engineName);
        request.SetDBInstanceClass(dbInstanceClass);

        Aws::RDS::Model::CreateDBInstanceOutcome outcome =
                client.CreateDBInstance(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB instance creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with RDS::CreateDBInstance. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER,
"",
                             client);
            return false;
        }
    }

    std::cout << "Waiting for the DB instance to become available." << std::endl;

    counter = 0;
    // 14. Wait for the DB instance to become available.
    do {
        std::this_thread::sleep_for(std::chrono::seconds(1));
        ++counter;
        if (counter > 900) {
            std::cerr << "Wait for instance to become available timed out ofter "
                      << counter
                      << " seconds." << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
```

```
            return false;
        }

        dbInstance = Aws::RDS::Model::DBInstance();
        if (!describeDBInstance(DB_INSTANCE_IDENTIFIER, dbInstance, client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
            return false;
        }

        if ((counter % 20) == 0) {
            std::cout << "Current DB instance status is '"
                      << dbInstance.GetDBInstanceStatus()
                      << "' after " << counter << " seconds." << std::endl;
        }
    } while (dbInstance.GetDBInstanceStatus() != "available");

    if (dbInstance.GetDBInstanceStatus() == "available") {
        std::cout << "The DB instance has been created." << std::endl;
    }

    // 15. Display the connection string that can be used to connect a 'mysql'
shell to the database.
    displayConnection(dbCluster);

    printAsterisksLine();

    if (askYesNoQuestion(
            "Do you want to create a snapshot of your DB cluster (y/n)? ")) {
        Aws::String snapshotID(DB_CLUSTER_IDENTIFIER + "-" +
                               Aws::String(Aws::Utils::UUID::RandomUUID()));
        {
            std::cout << "Creating a snapshot named " << snapshotID << "." <<
std::endl;
            std::cout << "This typically takes a few minutes." << std::endl;

            // 16. Create a snapshot of the DB cluster. (CreateDBClusterSnapshot)
            Aws::RDS::Model::CreateDBClusterSnapshotRequest request;
            request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::CreateDBClusterSnapshotOutcome outcome =
                    client.CreateDBClusterSnapshot(request);
```

```
            if (outcome.IsSuccess()) {
                std::cout << "Snapshot creation has started."
                          << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::CreateDBClusterSnapshot. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }
        }

        std::cout << "Waiting for the snapshot to become available." <<
std::endl;

        Aws::RDS::Model::DBClusterSnapshot snapshot;
        counter = 0;
        do {
            std::this_thread::sleep_for(std::chrono::seconds(1));
            ++counter;
            if (counter > 600) {
                std::cerr << "Wait for snapshot to be available timed out ofter "
                          << counter
                          << " seconds." << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }

            // 17. Wait for the snapshot to become available.
            Aws::RDS::Model::DescribeDBClusterSnapshotsRequest request;
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::DescribeDBClusterSnapshotsOutcome outcome =
                    client.DescribeDBClusterSnapshots(request);

            if (outcome.IsSuccess()) {
                snapshot = outcome.GetResult().GetDBClusterSnapshots()[0];
            }
```

```
            else {
                std::cerr << "Error with Aurora::DescribeDBClusterSnapshots. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
 client);
                return false;
            }

            if ((counter % 20) == 0) {
                std::cout << "Current snapshot status is '"
                          << snapshot.GetStatus()
                          << "' after " << counter << " seconds." << std::endl;
            }
        } while (snapshot.GetStatus() != "available");

        if (snapshot.GetStatus() != "available") {
            std::cout << "A snapshot has been created." << std::endl;
        }
    }

    printAsterisksLine();

    bool result = true;
    if (askYesNoQuestion(
            "Do you want to delete the DB cluster, DB instance, and parameter
 group (y/n)? ")) {
        result = cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                  DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
                                  client);
    }

    return result;
}

//! Routine which gets a DB cluster description.
/*!
 \sa describeDBCluster()
 \param dbClusterIdentifier: A DB cluster identifier.
 \param clusterResult: The 'DBCluster' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
```

```
bool AwsDoc::Aurora::describeDBCluster(const Aws::String &dbClusterIdentifier,
                                       Aws::RDS::Model::DBCluster &clusterResult,
                                       const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBClustersRequest request;
    request.SetDBClusterIdentifier(dbClusterIdentifier);

    Aws::RDS::Model::DescribeDBClustersOutcome outcome =
            client.DescribeDBClusters(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        clusterResult = outcome.GetResult().GetDBClusters()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_CLUSTER_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::GDescribeDBClusters. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB cluster does not exist.
        // Instead, clusterResult is set to empty.
    else {
        clusterResult = Aws::RDS::Model::DBCluster();
    }

    return result;

}


//! Routine which gets DB parameters using the 'DescribeDBClusterParameters' api.
/*!
 \sa getDBCLusterParameters()
 \param parameterGroupName: The name of the cluster parameter group.
 \param namePrefix: Prefix string to filter results by parameter name.
 \param source: A source such as 'user', ignored if empty.
 \param parametersResult: Vector of 'Parameter' objects returned by the routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBCLusterParameters(const Aws::String
 &parameterGroupName,
                                            const Aws::String &namePrefix,
```

```
                                            const Aws::String &source,

 Aws::Vector<Aws::RDS::Model::Parameter> &parametersResult,
                                            const Aws::RDS::RDSClient &client) {
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeDBClusterParametersRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }
        if (!source.empty()) {
            request.SetSource(source);
        }

        Aws::RDS::Model::DescribeDBClusterParametersOutcome outcome =
                client.DescribeDBClusterParameters(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::Parameter> &parameters =
                    outcome.GetResult().GetParameters();
            for (const Aws::RDS::Model::Parameter &parameter: parameters) {
                if (!namePrefix.empty()) {
                    if (parameter.GetParameterName().find(namePrefix) == 0) {
                        parametersResult.push_back(parameter);
                    }
                }
                else {
                    parametersResult.push_back(parameter);
                }
            }

            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameters. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    return true;
}
```

```
//! Routine which gets available DB engine versions for an engine name and
//! an optional parameter group family.
/*!
 \sa getDBEngineVersions()
 \param engineName: A DB engine name.
 \param parameterGroupFamily: A parameter group family name, ignored if empty.
 \param engineVersionsResult: Vector of 'DBEngineVersion' objects returned by the
 routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBEngineVersions(const Aws::String &engineName,
                                         const Aws::String &parameterGroupFamily,

 Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersionsResult,
                                         const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBEngineVersionsRequest request;
    request.SetEngine(engineName);
    if (!parameterGroupFamily.empty()) {
        request.SetDBParameterGroupFamily(parameterGroupFamily);
    }

    engineVersionsResult.clear();
    Aws::String marker; // The marker is used for pagination.
    do {
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeDBEngineVersionsOutcome outcome =
                client.DescribeDBEngineVersions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersions =
                    outcome.GetResult().GetDBEngineVersions();

            engineVersionsResult.insert(engineVersionsResult.end(),
                                        engineVersions.begin(),
 engineVersions.end());
            marker  = outcome.GetResult().GetMarker();
        }
        else {
```

```
            std::cerr << "Error with Aurora::DescribeDBEngineVersionsRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    } while (!marker.empty());

    return true;
}


//! Routine which gets a DB instance description.
/*!
 \sa describeDBCluster()
 \param dbInstanceIdentifier: A DB instance identifier.
 \param instanceResult: The 'DBInstance' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBInstance(const Aws::String &dbInstanceIdentifier,
                                        Aws::RDS::Model::DBInstance
 &instanceResult,
                                        const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBInstancesRequest request;
    request.SetDBInstanceIdentifier(dbInstanceIdentifier);

    Aws::RDS::Model::DescribeDBInstancesOutcome outcome =
            client.DescribeDBInstances(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        instanceResult = outcome.GetResult().GetDBInstances()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_INSTANCE_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::DescribeDBInstances. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB instance does not exist.
        // Instead, instanceResult is set to empty.
    else {
        instanceResult = Aws::RDS::Model::DBInstance();
    }
```

```
    return result;
}


//! Routine which gets available DB instance classes, displays the list
//! to the user, and returns the user selection.
/*!
 \sa chooseDBInstanceClass()
 \param engineName: The DB engine name.
 \param engineVersion: The DB engine version.
 \param dbInstanceClass: String for DB instance class chosen by the user.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::chooseDBInstanceClass(const Aws::String &engine,
                                           const Aws::String &engineVersion,
                                           Aws::String &dbInstanceClass,
                                           const Aws::RDS::RDSClient &client) {
    std::vector<Aws::String> instanceClasses;
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsRequest request;
        request.SetEngine(engine);
        request.SetEngineVersion(engineVersion);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsOutcome outcome =
                client.DescribeOrderableDBInstanceOptions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::OrderableDBInstanceOption>
 &options =
                    outcome.GetResult().GetOrderableDBInstanceOptions();
            for (const Aws::RDS::Model::OrderableDBInstanceOption &option:
 options) {
                const Aws::String &instanceClass = option.GetDBInstanceClass();
                if (std::find(instanceClasses.begin(), instanceClasses.end(),
                              instanceClass) == instanceClasses.end()) {
                    instanceClasses.push_back(instanceClass);
                }
            }
```

```
            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeOrderableDBInstanceOptions.
 "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    std::cout << "The available DB instance classes for your database engine
 are:"
              << std::endl;
    for (int i = 0; i < instanceClasses.size(); ++i) {
        std::cout << "   " << i + 1 << ": " << instanceClasses[i] << std::endl;
    }

    int choice = askQuestionForIntRange(
            "Which DB instance class do you want to use? ",
            1, static_cast<int>(instanceClasses.size()));
    dbInstanceClass = instanceClasses[choice - 1];
    return true;
}

//! Routine which deletes resources created by the scenario.
/*!
\sa cleanUpResources()
\param parameterGroupName: A parameter group name, this may be empty.
\param dbInstanceIdentifier: A DB instance identifier, this may be empty.
\param client: 'RDSClient' instance.
\return bool: Successful completion.
*/
bool AwsDoc::Aurora::cleanUpResources(const Aws::String &parameterGroupName,
                                      const Aws::String &dbClusterIdentifier,
                                      const Aws::String &dbInstanceIdentifier,
                                      const Aws::RDS::RDSClient &client) {
    bool result = true;
    bool instanceDeleting = false;
    bool clusterDeleting = false;
    if (!dbInstanceIdentifier.empty()) {
        {
            // 18. Delete the DB instance.
            Aws::RDS::Model::DeleteDBInstanceRequest request;
```

```
            request.SetDBInstanceIdentifier(dbInstanceIdentifier);
            request.SetSkipFinalSnapshot(true);
            request.SetDeleteAutomatedBackups(true);

            Aws::RDS::Model::DeleteDBInstanceOutcome outcome =
                    client.DeleteDBInstance(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB instance deletion has started."
                          << std::endl;
                instanceDeleting = true;
                std::cout
                        << "Waiting for DB instance to delete before deleting the
parameter group."
                        << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBInstance. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
        }
    }

    if (!dbClusterIdentifier.empty()) {
        {
            // 19. Delete the DB cluster.
            Aws::RDS::Model::DeleteDBClusterRequest request;
            request.SetDBClusterIdentifier(dbClusterIdentifier);
            request.SetSkipFinalSnapshot(true);

            Aws::RDS::Model::DeleteDBClusterOutcome outcome =
                    client.DeleteDBCluster(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB cluster deletion has started."
                          << std::endl;
                clusterDeleting = true;
                std::cout
                        << "Waiting for DB cluster to delete before deleting the
parameter group."
                        << std::endl;
                std::cout << "This may take a while." << std::endl;
```

```
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBCluster. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
        }
    }
    int counter = 0;

    while (clusterDeleting || instanceDeleting) {
        // 20. Wait for the DB cluster and instance to be deleted.
        std::this_thread::sleep_for(std::chrono::seconds(1));
        ++counter;
        if (counter > 800) {
            std::cerr << "Wait for instance to delete timed out ofter " <<
counter
                      << " seconds." << std::endl;
            return false;
        }

        Aws::RDS::Model::DBInstance dbInstance = Aws::RDS::Model::DBInstance();
        if (instanceDeleting) {
            if (!describeDBInstance(dbInstanceIdentifier, dbInstance, client)) {
                return false;
            }
            instanceDeleting = dbInstance.DBInstanceIdentifierHasBeenSet();
        }

        Aws::RDS::Model::DBCluster dbCluster = Aws::RDS::Model::DBCluster();
        if (clusterDeleting) {
            if (!describeDBCluster(dbClusterIdentifier, dbCluster, client)) {
                return false;
            }

            clusterDeleting = dbCluster.DBClusterIdentifierHasBeenSet();
        }

        if ((counter % 20) == 0) {
            if (instanceDeleting) {
                std::cout << "Current DB instance status is '"
                          << dbInstance.GetDBInstanceStatus() << "." <<
std::endl;
```

```
            }

            if (clusterDeleting) {
                std::cout << "Current DB cluster status is '"
                          << dbCluster.GetStatus() << "." << std::endl;
            }
        }
    }

    if (!parameterGroupName.empty()) {
        // 21. Delete the DB cluster parameter group.
        Aws::RDS::Model::DeleteDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(parameterGroupName);

        Aws::RDS::Model::DeleteDBClusterParameterGroupOutcome outcome =
                client.DeleteDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB parameter group was successfully deleted."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::DeleteDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
    }

    return result;
}
```

- API の詳細については、「AWS SDK for C++ API リファレンス」の以下のトピックを参照してください。
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup

- DeleteDBInstance
- DescribeDBClusterParameterGroups
- DescribeDBClusterParameters
- DescribeDBClusterSnapshots
- DescribeDBClusters
- DescribeDBEngineVersions
- DescribeDBInstances
- DescribeOrderableDBInstanceOptions
- ModifyDBClusterParameterGroup

Go

SDK for Go V2

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

コマンドプロンプトからインタラクティブのシナリオを実行します。

```
// GetStartedClusters is an interactive example that shows you how to use the AWS
 SDK for Go
// with Amazon Aurora to do the following:
//
// 1. Create a custom DB cluster parameter group and set parameter values.
// 2. Create an Aurora DB cluster that is configured to use the parameter group.
// 3. Create a DB instance in the DB cluster that contains a database.
// 4. Take a snapshot of the DB cluster.
// 5. Delete the DB instance, DB cluster, and parameter group.
type GetStartedClusters struct {
 sdkConfig  aws.Config
 dbClusters actions.DbClusters
 questioner demotools.IQuestioner
 helper     IScenarioHelper
 isTestRun  bool
```

```
}

// NewGetStartedClusters constructs a GetStartedClusters instance from a
 configuration.
// It uses the specified config to get an Amazon Relational Database Service
 (Amazon RDS)
// client and create wrappers for the actions used in the scenario.
func NewGetStartedClusters(sdkConfig aws.Config, questioner
 demotools.IQuestioner,
 helper IScenarioHelper) GetStartedClusters {
 auroraClient := rds.NewFromConfig(sdkConfig)
 return GetStartedClusters{
  sdkConfig:  sdkConfig,
  dbClusters: actions.DbClusters{AuroraClient: auroraClient},
  questioner: questioner,
  helper:     helper,
 }
}

// Run runs the interactive scenario.
func (scenario GetStartedClusters) Run(dbEngine string, parameterGroupName
 string,
 clusterName string, dbName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon Aurora DB Cluster demo.")
 log.Println(strings.Repeat("-", 88))

 parameterGroup := scenario.CreateParameterGroup(dbEngine, parameterGroupName)
 scenario.SetUserParameters(parameterGroupName)
 cluster := scenario.CreateCluster(clusterName, dbEngine, dbName, parameterGroup)
 scenario.helper.Pause(5)
 dbInstance := scenario.CreateInstance(cluster)
 scenario.DisplayConnection(cluster)
 scenario.CreateSnapshot(clusterName)
 scenario.Cleanup(dbInstance, cluster, parameterGroup)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
```

```
 log.Println(strings.Repeat("-", 88))
}

// CreateParameterGroup shows how to get available engine versions for a
 specified
// database engine and create a DB cluster parameter group that is compatible
 with a
// selected engine family.
func (scenario GetStartedClusters) CreateParameterGroup(dbEngine string,
 parameterGroupName string) *types.DBClusterParameterGroup {

 log.Printf("Checking for an existing DB cluster parameter group named %v.\n",
  parameterGroupName)
 parameterGroup, err := scenario.dbClusters.GetParameterGroup(parameterGroupName)
 if err != nil {
  panic(err)
 }
 if parameterGroup == nil {
  log.Printf("Getting available database engine versions for %v.\n", dbEngine)
  engineVersions, err := scenario.dbClusters.GetEngineVersions(dbEngine, "")
  if err != nil {
   panic(err)
  }

  familySet := map[string]struct{}{}
  for _, family := range engineVersions {
   familySet[*family.DBParameterGroupFamily] = struct{}{}
  }
  var families []string
  for family := range familySet {
   families = append(families, family)
  }
  sort.Strings(families)
  familyIndex := scenario.questioner.AskChoice("Which family do you want to use?
\n", families)
  log.Println("Creating a DB cluster parameter group.")
  _, err = scenario.dbClusters.CreateParameterGroup(
   parameterGroupName, families[familyIndex], "Example parameter group.")
  if err != nil {
   panic(err)
  }
  parameterGroup, err = scenario.dbClusters.GetParameterGroup(parameterGroupName)
  if err != nil {
   panic(err)
```

```
  }
 }
 log.Printf("Parameter group %v:\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tName: %v\n", *parameterGroup.DBClusterParameterGroupName)
 log.Printf("\tARN: %v\n", *parameterGroup.DBClusterParameterGroupArn)
 log.Printf("\tFamily: %v\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tDescription: %v\n", *parameterGroup.Description)
 log.Println(strings.Repeat("-", 88))
 return parameterGroup

}

// SetUserParameters shows how to get the parameters contained in a custom
 parameter
// group and update some of the parameter values in the group.
func (scenario GetStartedClusters) SetUserParameters(parameterGroupName string) {
 log.Println("Let's set some parameter values in your parameter group.")
 dbParameters, err := scenario.dbClusters.GetParameters(parameterGroupName, "")
 if err != nil {
  panic(err)
 }
 var updateParams []types.Parameter
 for _, dbParam := range dbParameters {
  if strings.HasPrefix(*dbParam.ParameterName, "auto_increment") &&
   dbParam.IsModifiable && *dbParam.DataType == "integer" {
   log.Printf("The %v parameter is described as:\n\t%v",
    *dbParam.ParameterName, *dbParam.Description)
   rangeSplit := strings.Split(*dbParam.AllowedValues, "-")
   lower, _ := strconv.Atoi(rangeSplit[0])
   upper, _ := strconv.Atoi(rangeSplit[1])
   newValue := scenario.questioner.AskInt(
    fmt.Sprintf("Enter a value between %v and %v:", lower, upper),
    demotools.InIntRange{Lower: lower, Upper: upper})
   dbParam.ParameterValue = aws.String(strconv.Itoa(newValue))
   updateParams = append(updateParams, dbParam)
  }
 }
 err = scenario.dbClusters.UpdateParameters(parameterGroupName, updateParams)
 if err != nil {
  panic(err)
 }
 log.Println("You can get a list of parameters you've set by specifying a source
 of 'user'.")
```

```
 userParameters, err := scenario.dbClusters.GetParameters(parameterGroupName,
 "user")
 if err != nil {
  panic(err)
 }
 log.Println("Here are the parameters you've set:")
 for _, param := range userParameters {
  log.Printf("\t%v: %v\n", *param.ParameterName, *param.ParameterValue)
 }
 log.Println(strings.Repeat("-", 88))
}

// CreateCluster shows how to create an Aurora DB cluster that contains a
 database
// of a specified type. The database is also configured to use a custom DB
 cluster
// parameter group.
func (scenario GetStartedClusters) CreateCluster(clusterName string, dbEngine
 string,
 dbName string, parameterGroup *types.DBClusterParameterGroup) *types.DBCluster {

 log.Println("Checking for an existing DB cluster.")
 cluster, err := scenario.dbClusters.GetDbCluster(clusterName)
 if err != nil {
  panic(err)
 }
 if cluster == nil {
  adminUsername := scenario.questioner.Ask(
   "Enter an administrator user name for the database: ", demotools.NotEmpty{})
  adminPassword := scenario.questioner.Ask(
   "Enter a password for the administrator (at least 8 characters): ",
 demotools.NotEmpty{})
  engineVersions, err := scenario.dbClusters.GetEngineVersions(dbEngine,
 *parameterGroup.DBParameterGroupFamily)
  if err != nil {
   panic(err)
  }
  var engineChoices []string
  for _, engine := range engineVersions {
   engineChoices = append(engineChoices, *engine.EngineVersion)
  }
  log.Println("The available engines for your parameter group are:")
  engineIndex := scenario.questioner.AskChoice("Which engine do you want to use?
\n", engineChoices)
```

```
  log.Printf("Creating DB cluster %v and database %v.\n", clusterName, dbName)
  log.Printf("The DB cluster is configured to use\nyour custom parameter group %v
\n",
   *parameterGroup.DBClusterParameterGroupName)
  log.Printf("and selected engine %v.\n", engineChoices[engineIndex])
  log.Println("This typically takes several minutes.")
  cluster, err = scenario.dbClusters.CreateDbCluster(
   clusterName, *parameterGroup.DBClusterParameterGroupName, dbName, dbEngine,
   engineChoices[engineIndex], adminUsername, adminPassword)
  if err != nil {
   panic(err)
  }
  for *cluster.Status != "available" {
   scenario.helper.Pause(30)
   cluster, err = scenario.dbClusters.GetDbCluster(clusterName)
   if err != nil {
    panic(err)
   }
   log.Println("Cluster created and available.")
  }
 }
 log.Println("Cluster data:")
 log.Printf("\tDBClusterIdentifier: %v\n", *cluster.DBClusterIdentifier)
 log.Printf("\tARN: %v\n", *cluster.DBClusterArn)
 log.Printf("\tStatus: %v\n", *cluster.Status)
 log.Printf("\tEngine: %v\n", *cluster.Engine)
 log.Printf("\tEngine version: %v\n", *cluster.EngineVersion)
 log.Printf("\tDBClusterParameterGroup: %v\n", *cluster.DBClusterParameterGroup)
 log.Printf("\tEngineMode: %v\n", *cluster.EngineMode)
 log.Println(strings.Repeat("-", 88))
 return cluster
}

// CreateInstance shows how to create a DB instance in an existing Aurora DB
 cluster.
// A new DB cluster contains no DB instances, so you must add one. The first DB
 instance
// that is added to a DB cluster defaults to a read-write DB instance.
func (scenario GetStartedClusters) CreateInstance(cluster *types.DBCluster)
 *types.DBInstance {
 log.Println("Checking for an existing database instance.")
 dbInstance, err := scenario.dbClusters.GetInstance(*cluster.DBClusterIdentifier)
 if err != nil {
  panic(err)
```

```
 }
 if dbInstance == nil {
  log.Println("Let's create a database instance in your DB cluster.")
  log.Println("First, choose a DB instance type:")
  instOpts, err := scenario.dbClusters.GetOrderableInstances(
   *cluster.Engine, *cluster.EngineVersion)
  if err != nil {
   panic(err)
  }
  var instChoices []string
  for _, opt := range instOpts {
   instChoices = append(instChoices, *opt.DBInstanceClass)
  }
  instIndex := scenario.questioner.AskChoice(
   "Which DB instance class do you want to use?\n", instChoices)
  log.Println("Creating a database instance. This typically takes several
 minutes.")
  dbInstance, err = scenario.dbClusters.CreateInstanceInCluster(
   *cluster.DBClusterIdentifier, *cluster.DBClusterIdentifier, *cluster.Engine,
   instChoices[instIndex])
  if err != nil {
   panic(err)
  }
  for *dbInstance.DBInstanceStatus != "available" {
   scenario.helper.Pause(30)
   dbInstance, err =
 scenario.dbClusters.GetInstance(*cluster.DBClusterIdentifier)
   if err != nil {
    panic(err)
   }
  }
 }
 log.Println("Instance data:")
 log.Printf("\tDBInstanceIdentifier: %v\n", *dbInstance.DBInstanceIdentifier)
 log.Printf("\tARN: %v\n", *dbInstance.DBInstanceArn)
 log.Printf("\tStatus: %v\n", *dbInstance.DBInstanceStatus)
 log.Printf("\tEngine: %v\n", *dbInstance.Engine)
 log.Printf("\tEngine version: %v\n", *dbInstance.EngineVersion)
 log.Println(strings.Repeat("-", 88))
 return dbInstance
}

// DisplayConnection displays connection information about an Aurora DB cluster
 and tips
```

```
// on how to connect to it.
func (scenario GetStartedClusters) DisplayConnection(cluster *types.DBCluster) {
 log.Println(
  "You can now connect to your database using your favorite MySql client.\n" +
   "One way to connect is by using the 'mysql' shell on an Amazon EC2 instance\n"
 +
   "that is running in the same VPC as your database cluster. Pass the endpoint,
\n" +
   "port, and administrator user name to 'mysql' and enter your password\n" +
   "when prompted:")
 log.Printf("\n\tmysql -h %v -P %v -u %v -p\n",
  *cluster.Endpoint, *cluster.Port, *cluster.MasterUsername)
 log.Println("For more information, see the User Guide for Aurora:\n" +
  "\thttps://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
CHAP_GettingStartedAurora.CreatingConnecting.Aurora.html#CHAP_GettingStartedAurora.Aurora
 log.Println(strings.Repeat("-", 88))
}

// CreateSnapshot shows how to create a DB cluster snapshot and wait until it's
 available.
func (scenario GetStartedClusters) CreateSnapshot(clusterName string) {
 if scenario.questioner.AskBool(
  "Do you want to create a snapshot of your DB cluster (y/n)? ", "y") {
  snapshotId := fmt.Sprintf("%v-%v", clusterName, scenario.helper.UniqueId())
  log.Printf("Creating a snapshot named %v. This typically takes a few minutes.
\n", snapshotId)
  snapshot, err := scenario.dbClusters.CreateClusterSnapshot(clusterName,
 snapshotId)
  if err != nil {
   panic(err)
  }
  for *snapshot.Status != "available" {
   scenario.helper.Pause(30)
   snapshot, err = scenario.dbClusters.GetClusterSnapshot(snapshotId)
   if err != nil {
    panic(err)
   }
  }
  log.Println("Snapshot data:")
  log.Printf("\tDBClusterSnapshotIdentifier: %v\n",
 *snapshot.DBClusterSnapshotIdentifier)
  log.Printf("\tARN: %v\n", *snapshot.DBClusterSnapshotArn)
  log.Printf("\tStatus: %v\n", *snapshot.Status)
  log.Printf("\tEngine: %v\n", *snapshot.Engine)
```

```
  log.Printf("\tEngine version: %v\n", *snapshot.EngineVersion)
  log.Printf("\tDBClusterIdentifier: %v\n", *snapshot.DBClusterIdentifier)
  log.Printf("\tSnapshotCreateTime: %v\n", *snapshot.SnapshotCreateTime)
  log.Println(strings.Repeat("-", 88))
 }
}

// Cleanup shows how to clean up a DB instance, DB cluster, and DB cluster
 parameter group.
// Before the DB cluster parameter group can be deleted, all associated DB
 instances and
// DB clusters must first be deleted.
func (scenario GetStartedClusters) Cleanup(dbInstance *types.DBInstance, cluster
 *types.DBCluster,
 parameterGroup *types.DBClusterParameterGroup) {

 if scenario.questioner.AskBool(
  "\nDo you want to delete the database instance, DB cluster, and parameter group
 (y/n)? ", "y") {
  log.Printf("Deleting database instance %v.\n",
 *dbInstance.DBInstanceIdentifier)
  err := scenario.dbClusters.DeleteInstance(*dbInstance.DBInstanceIdentifier)
  if err != nil {
   panic(err)
  }
  log.Printf("Deleting database cluster %v.\n", *cluster.DBClusterIdentifier)
  err = scenario.dbClusters.DeleteDbCluster(*cluster.DBClusterIdentifier)
  if err != nil {
   panic(err)
  }
  log.Println(
   "Waiting for the DB instance and DB cluster to delete. This typically takes
 several minutes.")
  for dbInstance != nil || cluster != nil {
   scenario.helper.Pause(30)
   if dbInstance != nil {
    dbInstance, err =
 scenario.dbClusters.GetInstance(*dbInstance.DBInstanceIdentifier)
    if err != nil {
     panic(err)
    }
   }
   if cluster != nil {
    cluster, err = scenario.dbClusters.GetDbCluster(*cluster.DBClusterIdentifier)
```

```
    if err != nil {
     panic(err)
    }
   }
  }
  log.Printf("Deleting parameter group %v.",
 *parameterGroup.DBClusterParameterGroupName)
  err =
 scenario.dbClusters.DeleteParameterGroup(*parameterGroup.DBClusterParameterGroupName)
  if err != nil {
   panic(err)
  }
 }
}
```

Aurora アクションを管理するためにシナリオによって呼び出される関数を定義します。

```
type DbClusters struct {
 AuroraClient *rds.Client
}


// GetParameterGroup gets a DB cluster parameter group by name.
func (clusters *DbClusters) GetParameterGroup(parameterGroupName string) (
 *types.DBClusterParameterGroup, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterParameterGroups(
  context.TODO(), &rds.DescribeDBClusterParameterGroupsInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
 } else {
  return &output.DBClusterParameterGroups[0], err
```

```
 }
}


// CreateParameterGroup creates a DB cluster parameter group that is based on the
 specified
// parameter group family.
func (clusters *DbClusters) CreateParameterGroup(
 parameterGroupName string, parameterGroupFamily string, description string) (
 *types.DBClusterParameterGroup, error) {

 output, err :=
 clusters.AuroraClient.CreateDBClusterParameterGroup(context.TODO(),
  &rds.CreateDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   DBParameterGroupFamily:      aws.String(parameterGroupFamily),
   Description:                 aws.String(description),
  })
 if err != nil {
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBClusterParameterGroup, err
 }
}



// DeleteParameterGroup deletes the named DB cluster parameter group.
func (clusters *DbClusters) DeleteParameterGroup(parameterGroupName string) error
 {
 _, err := clusters.AuroraClient.DeleteDBClusterParameterGroup(context.TODO(),
  &rds.DeleteDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

```
// GetParameters gets the parameters that are contained in a DB cluster parameter
 group.
func (clusters *DbClusters) GetParameters(parameterGroupName string, source
 string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBClusterParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator :=
 rds.NewDescribeDBClusterParametersPaginator(clusters.AuroraClient,
  &rds.DescribeDBClusterParametersInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Source:                      aws.String(source),
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(context.TODO())
  if err != nil {
   log.Printf("Couldn't get paramaeters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
}



// UpdateParameters updates parameters in a named DB cluster parameter group.
func (clusters *DbClusters) UpdateParameters(parameterGroupName string, params
 []types.Parameter) error {
 _, err := clusters.AuroraClient.ModifyDBClusterParameterGroup(context.TODO(),
  &rds.ModifyDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Parameters:                  params,
  })
 if err != nil {
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
```

```
}



// GetDbCluster gets data about an Aurora DB cluster.
func (clusters *DbClusters) GetDbCluster(clusterName string) (*types.DBCluster,
 error) {
 output, err := clusters.AuroraClient.DescribeDBClusters(context.TODO(),
  &rds.DescribeDBClustersInput{
   DBClusterIdentifier: aws.String(clusterName),
  })
 if err != nil {
  var notFoundError *types.DBClusterNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB cluster %v does not exist.\n", clusterName)
   err = nil
  } else {
   log.Printf("Couldn't get DB cluster %v: %v\n", clusterName, err)
  }
  return nil, err
 } else {
  return &output.DBClusters[0], err
 }
}



// CreateDbCluster creates a DB cluster that is configured to use the specified
 parameter group.
// The newly created DB cluster contains a database that uses the specified
 engine and
// engine version.
func (clusters *DbClusters) CreateDbCluster(clusterName string,
 parameterGroupName string,
 dbName string, dbEngine string, dbEngineVersion string, adminName string,
 adminPassword string) (
 *types.DBCluster, error) {

 output, err := clusters.AuroraClient.CreateDBCluster(context.TODO(),
 &rds.CreateDBClusterInput{
  DBClusterIdentifier:         aws.String(clusterName),
  Engine:                      aws.String(dbEngine),
  DBClusterParameterGroupName: aws.String(parameterGroupName),
  DatabaseName:                aws.String(dbName),
```

```
  EngineVersion:               aws.String(dbEngineVersion),
  MasterUserPassword:          aws.String(adminPassword),
  MasterUsername:              aws.String(adminName),
 })
 if err != nil {
  log.Printf("Couldn't create DB cluster %v: %v\n", clusterName, err)
  return nil, err
 } else {
  return output.DBCluster, err
 }
}



// DeleteDbCluster deletes a DB cluster without keeping a final snapshot.
func (clusters *DbClusters) DeleteDbCluster(clusterName string) error {
 _, err := clusters.AuroraClient.DeleteDBCluster(context.TODO(),
 &rds.DeleteDBClusterInput{
  DBClusterIdentifier: aws.String(clusterName),
  SkipFinalSnapshot:   true,
 })
 if err != nil {
  log.Printf("Couldn't delete DB cluster %v: %v\n", clusterName, err)
  return err
 } else {
  return nil
 }
}



// CreateClusterSnapshot creates a snapshot of a DB cluster.
func (clusters *DbClusters) CreateClusterSnapshot(clusterName string,
 snapshotName string) (
 *types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.CreateDBClusterSnapshot(context.TODO(),
 &rds.CreateDBClusterSnapshotInput{
  DBClusterIdentifier:         aws.String(clusterName),
  DBClusterSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
```

```
  return output.DBClusterSnapshot, nil
 }
}



// GetClusterSnapshot gets a DB cluster snapshot.
func (clusters *DbClusters) GetClusterSnapshot(snapshotName string)
 (*types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterSnapshots(context.TODO(),
  &rds.DescribeDBClusterSnapshotsInput{
   DBClusterSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return &output.DBClusterSnapshots[0], nil
 }
}



// CreateInstanceInCluster creates a database instance in an existing DB cluster.
 The first database that is
// created defaults to a read-write DB instance.
func (clusters *DbClusters) CreateInstanceInCluster(clusterName string,
 instanceName string,
 dbEngine string, dbInstanceClass string) (*types.DBInstance, error) {
 output, err := clusters.AuroraClient.CreateDBInstance(context.TODO(),
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBClusterIdentifier:  aws.String(clusterName),
  Engine:               aws.String(dbEngine),
  DBInstanceClass:      aws.String(dbInstanceClass),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
  return output.DBInstance, nil
 }
}
```

```
// GetInstance gets data about a DB instance.
func (clusters *DbClusters) GetInstance(instanceName string) (
 *types.DBInstance, error) {
 output, err := clusters.AuroraClient.DescribeDBInstances(context.TODO(),
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}



// DeleteInstance deletes a DB instance.
func (clusters *DbClusters) DeleteInstance(instanceName string) error {
 _, err := clusters.AuroraClient.DeleteDBInstance(context.TODO(),
 &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      true,
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}
```

```
// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
func (clusters *DbClusters) GetEngineVersions(engine string, parameterGroupFamily
 string) (
 []types.DBEngineVersion, error) {
 output, err := clusters.AuroraClient.DescribeDBEngineVersions(context.TODO(),
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
 }
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (clusters *DbClusters) GetOrderableInstances(engine string, engineVersion
 string) (
 []types.OrderableDBInstanceOption, error) {

 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instances []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(clusters.AuroraClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(context.TODO())
  if err != nil {
   log.Printf("Couldn't get orderable DB instances: %v\n", err)
   break
  } else {
   instances = append(instances, output.OrderableDBInstanceOptions...)
```

```
  }
 }
 return instances, err
}
```

- API の詳細については、「AWS SDK for Go API リファレンス」の以下のトピックを参照してください。
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeOrderableDBInstanceOptions
  - ModifyDBClusterParameterGroup

Java

SDK for Java 2.x

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/**
 * Before running this Java (v2) code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * This example requires an AWS Secrets Manager secret that contains the
 * database credentials. If you do not create a
 * secret, this example will not work. For details, see:
 *
 * https://docs.aws.amazon.com/secretsmanager/latest/userguide/integrating_how-
services-use-secrets_RS.html
 *
 * This Java example performs the following tasks:
 *
 * 1. Gets available engine families for Amazon Aurora MySQL-Compatible Edition
 * by calling the DescribeDbEngineVersions(Engine='aurora-mysql') method.
 * 2. Selects an engine family and creates a custom DB cluster parameter group
 * by invoking the describeDBClusterParameters method.
 * 3. Gets the parameter groups by invoking the describeDBClusterParameterGroups
 * method.
 * 4. Gets parameters in the group by invoking the describeDBClusterParameters
 * method.
 * 5. Modifies the auto_increment_offset parameter by invoking the
 * modifyDbClusterParameterGroupRequest method.
 * 6. Gets and displays the updated parameters.
 * 7. Gets a list of allowed engine versions by invoking the
 * describeDbEngineVersions method.
 * 8. Creates an Aurora DB cluster database cluster that contains a MySQL
 * database.
 * 9. Waits for DB instance to be ready.
 * 10. Gets a list of instance classes available for the selected engine.
 * 11. Creates a database instance in the cluster.
 * 12. Waits for DB instance to be ready.
 * 13. Creates a snapshot.
 * 14. Waits for DB snapshot to be ready.
 * 15. Deletes the DB cluster.
 * 16. Deletes the DB cluster group.
 */
public class AuroraScenario {
```

```
   public static long sleepTime = 20;
   public static final String DASHES = new String(new char[80]).replace("\0",
"-");

   public static void main(String[] args) throws InterruptedException {
       final String usage = "\n" +
               "Usage:\n" +
               "    <dbClusterGroupName> <dbParameterGroupFamily>
<dbInstanceClusterIdentifier> <dbInstanceIdentifier> <dbName>
<dbSnapshotIdentifier><secretName>"
               +
               "Where:\n" +
               "    dbClusterGroupName - The name of the DB cluster parameter
group. \n" +
               "    dbParameterGroupFamily - The DB cluster parameter group
family name (for example, aurora-mysql5.7). \n"
               +
               "    dbInstanceClusterIdentifier - The instance cluster
identifier value.\n" +
               "    dbInstanceIdentifier - The database instance identifier.\n"
+
               "    dbName - The database name.\n" +
               "    dbSnapshotIdentifier - The snapshot identifier.\n" +
               "    secretName - The name of the AWS Secrets Manager secret that
contains the database credentials\"\n";
       ;

       if (args.length != 7) {
           System.out.println(usage);
           System.exit(1);
       }

       String dbClusterGroupName = args[0];
       String dbParameterGroupFamily = args[1];
       String dbInstanceClusterIdentifier = args[2];
       String dbInstanceIdentifier = args[3];
       String dbName = args[4];
       String dbSnapshotIdentifier = args[5];
       String secretName = args[6];

       // Retrieve the database credentials using AWS Secrets Manager.
       Gson gson = new Gson();
       User user = gson.fromJson(String.valueOf(getSecretValues(secretName)),
User.class);
```

```
        String username = user.getUsername();
        String userPassword = user.getPassword();

        Region region = Region.US_WEST_2;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        System.out.println(DASHES);
        System.out.println("Welcome to the Amazon Aurora example scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("1. Return a list of the available DB engines");
        describeDBEngines(rdsClient);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Create a custom parameter group");
        createDBClusterParameterGroup(rdsClient, dbClusterGroupName,
dbParameterGroupFamily);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Get the parameter group");
        describeDbClusterParameterGroups(rdsClient, dbClusterGroupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Get the parameters in the group");
        describeDbClusterParameters(rdsClient, dbClusterGroupName, 0);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Modify the auto_increment_offset parameter");
        modifyDBClusterParas(rdsClient, dbClusterGroupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Display the updated parameter value");
        describeDbClusterParameters(rdsClient, dbClusterGroupName, -1);
        System.out.println(DASHES);

        System.out.println(DASHES);
```

```
        System.out.println("7. Get a list of allowed engine versions");
        getAllowedEngines(rdsClient, dbParameterGroupFamily);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Create an Aurora DB cluster database");
        String arnClusterVal = createDBCluster(rdsClient, dbClusterGroupName,
dbName, dbInstanceClusterIdentifier,
                username, userPassword);
        System.out.println("The ARN of the cluster is " + arnClusterVal);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. Wait for DB instance to be ready");
        waitForInstanceReady(rdsClient, dbInstanceClusterIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("10. Get a list of instance classes available for the
selected engine");
        String instanceClass = getListInstanceClasses(rdsClient);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("11. Create a database instance in the cluster.");
        String clusterDBARN = createDBInstanceCluster(rdsClient,
dbInstanceIdentifier, dbInstanceClusterIdentifier,
                instanceClass);
        System.out.println("The ARN of the database is " + clusterDBARN);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("12. Wait for DB instance to be ready");
        waitDBInstanceReady(rdsClient, dbInstanceIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("13. Create a snapshot");
        createDBClusterSnapshot(rdsClient, dbInstanceClusterIdentifier,
dbSnapshotIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("14. Wait for DB snapshot to be ready");
```

```
        waitForSnapshotReady(rdsClient, dbSnapshotIdentifier,
dbInstanceClusterIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("14. Delete the DB instance");
        deleteDatabaseInstance(rdsClient, dbInstanceIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("15. Delete the DB cluster");
        deleteCluster(rdsClient, dbInstanceClusterIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("16. Delete the DB cluster group");
        deleteDBClusterGroup(rdsClient, dbClusterGroupName, clusterDBARN);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The Scenario has successfully completed.");
        System.out.println(DASHES);
        rdsClient.close();
    }

    private static SecretsManagerClient getSecretClient() {
        Region region = Region.US_WEST_2;
        return SecretsManagerClient.builder()
                .region(region)

.credentialsProvider(EnvironmentVariableCredentialsProvider.create())
                .build();
    }

    private static String getSecretValues(String secretName) {
        SecretsManagerClient secretClient = getSecretClient();
        GetSecretValueRequest valueRequest = GetSecretValueRequest.builder()
                .secretId(secretName)
                .build();

        GetSecretValueResponse valueResponse =
secretClient.getSecretValue(valueRequest);
        return valueResponse.secretString();
    }
```

```
   public static void deleteDBClusterGroup(RdsClient rdsClient, String
dbClusterGroupName, String clusterDBARN)
           throws InterruptedException {
       try {
           boolean isDataDel = false;
           boolean didFind;
           String instanceARN;

           // Make sure that the database has been deleted.
           while (!isDataDel) {
               DescribeDbInstancesResponse response =
rdsClient.describeDBInstances();
               List<DBInstance> instanceList = response.dbInstances();
               int listSize = instanceList.size();
               didFind = false;
               int index = 1;
               for (DBInstance instance : instanceList) {
                   instanceARN = instance.dbInstanceArn();
                   if (instanceARN.compareTo(clusterDBARN) == 0) {
                       System.out.println(clusterDBARN + " still exists");
                       didFind = true;
                   }
                   if ((index == listSize) && (!didFind)) {
                       // Went through the entire list and did not find the
database ARN.
                       isDataDel = true;
                   }
                   Thread.sleep(sleepTime * 1000);
                   index++;
               }
           }

           DeleteDbClusterParameterGroupRequest clusterParameterGroupRequest =
DeleteDbClusterParameterGroupRequest
                   .builder()
                   .dbClusterParameterGroupName(dbClusterGroupName)
                   .build();


rdsClient.deleteDBClusterParameterGroup(clusterParameterGroupRequest);
           System.out.println(dbClusterGroupName + " was deleted.");

       } catch (RdsException e) {
```

```
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void deleteCluster(RdsClient rdsClient, String
dbInstanceClusterIdentifier) {
        try {
            DeleteDbClusterRequest deleteDbClusterRequest =
DeleteDbClusterRequest.builder()
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .skipFinalSnapshot(true)
                    .build();

            rdsClient.deleteDBCluster(deleteDbClusterRequest);
            System.out.println(dbInstanceClusterIdentifier + " was deleted!");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void deleteDatabaseInstance(RdsClient rdsClient, String
dbInstanceIdentifier) {
        try {
            DeleteDbInstanceRequest deleteDbInstanceRequest =
DeleteDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .deleteAutomatedBackups(true)
                    .skipFinalSnapshot(true)
                    .build();

            DeleteDbInstanceResponse response =
rdsClient.deleteDBInstance(deleteDbInstanceRequest);
            System.out.println("The status of the database is " +
response.dbInstance().dbInstanceStatus());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

```
   public static void waitForSnapshotReady(RdsClient rdsClient, String
dbSnapshotIdentifier,
           String dbInstanceClusterIdentifier) {
       try {
           boolean snapshotReady = false;
           String snapshotReadyStr;
           System.out.println("Waiting for the snapshot to become available.");

           DescribeDbClusterSnapshotsRequest snapshotsRequest =
DescribeDbClusterSnapshotsRequest.builder()
                   .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                   .dbClusterIdentifier(dbInstanceClusterIdentifier)
                   .build();

           while (!snapshotReady) {
               DescribeDbClusterSnapshotsResponse response =
rdsClient.describeDBClusterSnapshots(snapshotsRequest);
               List<DBClusterSnapshot> snapshotList =
response.dbClusterSnapshots();
               for (DBClusterSnapshot snapshot : snapshotList) {
                   snapshotReadyStr = snapshot.status();
                   if (snapshotReadyStr.contains("available")) {
                       snapshotReady = true;
                   } else {
                       System.out.println(".");
                       Thread.sleep(sleepTime * 5000);
                   }
               }
           }

           System.out.println("The Snapshot is available!");

       } catch (RdsException | InterruptedException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }

   public static void createDBClusterSnapshot(RdsClient rdsClient, String
dbInstanceClusterIdentifier,
           String dbSnapshotIdentifier) {
       try {
           CreateDbClusterSnapshotRequest snapshotRequest =
CreateDbClusterSnapshotRequest.builder()
```

```
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                    .build();

            CreateDbClusterSnapshotResponse response =
rdsClient.createDBClusterSnapshot(snapshotRequest);
            System.out.println("The Snapshot ARN is " +
response.dbClusterSnapshot().dbClusterSnapshotArn());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void waitDBInstanceReady(RdsClient rdsClient, String
dbInstanceIdentifier) {
        boolean instanceReady = false;
        String instanceReadyStr;
        System.out.println("Waiting for instance to become available.");
        try {
            DescribeDbInstancesRequest instanceRequest =
DescribeDbInstancesRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .build();

            String endpoint = "";
            while (!instanceReady) {
                DescribeDbInstancesResponse response =
rdsClient.describeDBInstances(instanceRequest);
                List<DBInstance> instanceList = response.dbInstances();
                for (DBInstance instance : instanceList) {
                    instanceReadyStr = instance.dbInstanceStatus();
                    if (instanceReadyStr.contains("available")) {
                        endpoint = instance.endpoint().address();
                        instanceReady = true;
                    } else {
                        System.out.print(".");
                        Thread.sleep(sleepTime * 1000);
                    }
                }
            }
            System.out.println("Database instance is available! The connection
endpoint is " + endpoint);
```

```
        } catch (RdsException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static String createDBInstanceCluster(RdsClient rdsClient,
            String dbInstanceIdentifier,
            String dbInstanceClusterIdentifier,
            String instanceClass) {
        try {
            CreateDbInstanceRequest instanceRequest =
CreateDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .engine("aurora-mysql")
                    .dbInstanceClass(instanceClass)
                    .build();

            CreateDbInstanceResponse response =
rdsClient.createDBInstance(instanceRequest);
            System.out.print("The status is " +
response.dbInstance().dbInstanceStatus());
            return response.dbInstance().dbInstanceArn();

        } catch (RdsException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }

    public static String getListInstanceClasses(RdsClient rdsClient) {
        try {
            DescribeOrderableDbInstanceOptionsRequest optionsRequest =
DescribeOrderableDbInstanceOptionsRequest
                    .builder()
                    .engine("aurora-mysql")
                    .maxRecords(20)
                    .build();

            DescribeOrderableDbInstanceOptionsResponse response = rdsClient
                    .describeOrderableDBInstanceOptions(optionsRequest);
```

```
            List<OrderableDBInstanceOption> instanceOptions =
response.orderableDBInstanceOptions();
            String instanceClass = "";
            for (OrderableDBInstanceOption instanceOption : instanceOptions) {
                instanceClass = instanceOption.dbInstanceClass();
                System.out.println("The instance class is " +
instanceOption.dbInstanceClass());
                System.out.println("The engine version is " +
instanceOption.engineVersion());
            }
            return instanceClass;

        } catch (RdsException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }

    // Waits until the database instance is available.
    public static void waitForInstanceReady(RdsClient rdsClient, String
dbClusterIdentifier) {
        boolean instanceReady = false;
        String instanceReadyStr;
        System.out.println("Waiting for instance to become available.");
        try {
            DescribeDbClustersRequest instanceRequest =
DescribeDbClustersRequest.builder()
                    .dbClusterIdentifier(dbClusterIdentifier)
                    .build();

            while (!instanceReady) {
                DescribeDbClustersResponse response =
rdsClient.describeDBClusters(instanceRequest);
                List<DBCluster> clusterList = response.dbClusters();
                for (DBCluster cluster : clusterList) {
                    instanceReadyStr = cluster.status();
                    if (instanceReadyStr.contains("available")) {
                        instanceReady = true;
                    } else {
                        System.out.print(".");
                        Thread.sleep(sleepTime * 1000);
                    }
                }
```

```
            }
            System.out.println("Database cluster is available!");

        } catch (RdsException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static String createDBCluster(RdsClient rdsClient, String
dbParameterGroupFamily, String dbName,
            String dbClusterIdentifier, String userName, String password) {
        try {
            CreateDbClusterRequest clusterRequest =
CreateDbClusterRequest.builder()
                    .databaseName(dbName)
                    .dbClusterIdentifier(dbClusterIdentifier)
                    .dbClusterParameterGroupName(dbParameterGroupFamily)
                    .engine("aurora-mysql")
                    .masterUsername(userName)
                    .masterUserPassword(password)
                    .build();

            CreateDbClusterResponse response =
rdsClient.createDBCluster(clusterRequest);
            return response.dbCluster().dbClusterArn();

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
        return "";
    }

    // Get a list of allowed engine versions.
    public static void getAllowedEngines(RdsClient rdsClient, String
dbParameterGroupFamily) {
        try {
            DescribeDbEngineVersionsRequest versionsRequest =
DescribeDbEngineVersionsRequest.builder()
                    .dbParameterGroupFamily(dbParameterGroupFamily)
                    .engine("aurora-mysql")
                    .build();
```

```
            DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(versionsRequest);
            List<DBEngineVersion> dbEngines = response.dbEngineVersions();
            for (DBEngineVersion dbEngine : dbEngines) {
                System.out.println("The engine version is " +
dbEngine.engineVersion());
                System.out.println("The engine description is " +
dbEngine.dbEngineDescription());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    // Modify the auto_increment_offset parameter.
    public static void modifyDBClusterParas(RdsClient rdsClient, String
dClusterGroupName) {
        try {
            Parameter parameter1 = Parameter.builder()
                    .parameterName("auto_increment_offset")
                    .applyMethod("immediate")
                    .parameterValue("5")
                    .build();

            List<Parameter> paraList = new ArrayList<>();
            paraList.add(parameter1);
            ModifyDbClusterParameterGroupRequest groupRequest =
ModifyDbClusterParameterGroupRequest.builder()
                    .dbClusterParameterGroupName(dClusterGroupName)
                    .parameters(paraList)
                    .build();

            ModifyDbClusterParameterGroupResponse response =
rdsClient.modifyDBClusterParameterGroup(groupRequest);
            System.out.println(
                    "The parameter group " +
response.dbClusterParameterGroupName() + " was successfully modified");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
```

```
   }

   public static void describeDbClusterParameters(RdsClient rdsClient, String
dbCLusterGroupName, int flag) {
       try {
           DescribeDbClusterParametersRequest dbParameterGroupsRequest;
           if (flag == 0) {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .build();
           } else {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .source("user")
                       .build();
           }

           DescribeDbClusterParametersResponse response = rdsClient
                   .describeDBClusterParameters(dbParameterGroupsRequest);
           List<Parameter> dbParameters = response.parameters();
           String paraName;
           for (Parameter para : dbParameters) {
               // Only print out information about either auto_increment_offset
or
               // auto_increment_increment.
               paraName = para.parameterName();
               if ((paraName.compareTo("auto_increment_offset") == 0)
                       || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                   System.out.println("*** The parameter name is  " + paraName);
                   System.out.println("*** The parameter value is  " +
para.parameterValue());
                   System.out.println("*** The parameter data type is " +
para.dataType());
                   System.out.println("*** The parameter description is " +
para.description());
                   System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
               }
           }

       } catch (RdsException e) {
```

```
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void describeDbClusterParameterGroups(RdsClient rdsClient,
String dbClusterGroupName) {
        try {
            DescribeDbClusterParameterGroupsRequest groupsRequest =
DescribeDbClusterParameterGroupsRequest.builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .maxRecords(20)
                    .build();

            List<DBClusterParameterGroup> groups =
rdsClient.describeDBClusterParameterGroups(groupsRequest)
                    .dbClusterParameterGroups();
            for (DBClusterParameterGroup group : groups) {
                System.out.println("The group name is " +
group.dbClusterParameterGroupName());
                System.out.println("The group ARN is " +
group.dbClusterParameterGroupArn());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void createDBClusterParameterGroup(RdsClient rdsClient, String
dbClusterGroupName,
            String dbParameterGroupFamily) {
        try {
            CreateDbClusterParameterGroupRequest groupRequest =
CreateDbClusterParameterGroupRequest.builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .dbParameterGroupFamily(dbParameterGroupFamily)
                    .description("Created by using the AWS SDK for Java")
                    .build();

            CreateDbClusterParameterGroupResponse response =
rdsClient.createDBClusterParameterGroup(groupRequest);
```

```
            System.out.println("The group name is " +
 response.dbClusterParameterGroup().dbClusterParameterGroupName());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void describeDBEngines(RdsClient rdsClient) {
        try {
            DescribeDbEngineVersionsRequest engineVersionsRequest =
 DescribeDbEngineVersionsRequest.builder()
                    .engine("aurora-mysql")
                    .defaultOnly(true)
                    .maxRecords(20)
                    .build();

            DescribeDbEngineVersionsResponse response =
 rdsClient.describeDBEngineVersions(engineVersionsRequest);
            List<DBEngineVersion> engines = response.dbEngineVersions();

            // Get all DBEngineVersion objects.
            for (DBEngineVersion engineOb : engines) {
                System.out.println("The name of the DB parameter group family for
 the database engine is "
                        + engineOb.dbParameterGroupFamily());
                System.out.println("The name of the database engine " +
 engineOb.engine());
                System.out.println("The version number of the database engine " +
 engineOb.engineVersion());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
}
```

- API の詳細については、「AWS SDK for Java 2.x API リファレンス」の以下のトピックを参照してください。

- CreateDBCluster
- CreateDBClusterParameterGroup
- CreateDBClusterSnapshot
- CreateDBInstance
- DeleteDBCluster
- DeleteDBClusterParameterGroup
- DeleteDBInstance
- DescribeDBClusterParameterGroups
- DescribeDBClusterParameters
- DescribeDBClusterSnapshots
- DescribeDBClusters
- DescribeDBEngineVersions
- DescribeDBInstances
- DescribeOrderableDBInstanceOptions
- ModifyDBClusterParameterGroup

Kotlin

SDK for Kotlin

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

```
/**
Before running this Kotlin code example, set up your development environment,
 including your credentials.

For more information, see the following documentation topic:

https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html
```

```
This example requires an AWS Secrets Manager secret that contains the database
 credentials. If you do not create a
secret, this example will not work. For more details, see:

https://docs.aws.amazon.com/secretsmanager/latest/userguide/integrating_how-
services-use-secrets_RS.html

This Kotlin example performs the following tasks:

1. Returns a list of the available DB engines.
2. Creates a custom DB parameter group.
3. Gets the parameter groups.
4. Gets the parameters in the group.
5. Modifies the auto_increment_increment parameter.
6. Displays the updated parameter value.
7. Gets a list of allowed engine versions.
8. Creates an Aurora DB cluster database.
9. Waits for DB instance to be ready.
10. Gets a list of instance classes available for the selected engine.
11. Creates a database instance in the cluster.
12. Waits for the database instance in the cluster to be ready.
13. Creates a snapshot.
14. Waits for DB snapshot to be ready.
15. Deletes the DB instance.
16. Deletes the DB cluster.
17. Deletes the DB cluster group.
 */

var slTime: Long = 20
suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
            <dbClusterGroupName> <dbParameterGroupFamily>
 <dbInstanceClusterIdentifier> <dbName> <dbSnapshotIdentifier> <secretName>
        Where:
            dbClusterGroupName - The database group name.
            dbParameterGroupFamily - The database parameter group name.
            dbInstanceClusterIdentifier - The database instance identifier.
            dbName -  The database name.
            dbSnapshotIdentifier - The snapshot identifier.
            secretName - The name of the AWS Secrets Manager secret that contains
 the database credentials.
    """
```

```
    if (args.size != 7) {
        println(usage)
        exitProcess(1)
    }

    val dbClusterGroupName = args[0]
    val dbParameterGroupFamily = args[1]
    val dbInstanceClusterIdentifier = args[2]
    val dbInstanceIdentifier = args[3]
    val dbName = args[4]
    val dbSnapshotIdentifier = args[5]
    val secretName = args[6]

    val gson = Gson()
    val user = gson.fromJson(getSecretValues(secretName).toString(),
User::class.java)
    val username = user.username
    val userPassword = user.password

    println("1. Return a list of the available DB engines")
    describeAuroraDBEngines()

    println("2. Create a custom parameter group")
    createDBClusterParameterGroup(dbClusterGroupName, dbParameterGroupFamily)

    println("3. Get the parameter group")
    describeDbClusterParameterGroups(dbClusterGroupName)

    println("4. Get the parameters in the group")
    describeDbClusterParameters(dbClusterGroupName, 0)

    println("5. Modify the auto_increment_offset parameter")
    modifyDBClusterParas(dbClusterGroupName)

    println("6. Display the updated parameter value")
    describeDbClusterParameters(dbClusterGroupName, -1)

    println("7. Get a list of allowed engine versions")
    getAllowedClusterEngines(dbParameterGroupFamily)

    println("8. Create an Aurora DB cluster database")
    val arnClusterVal = createDBCluster(dbClusterGroupName, dbName,
dbInstanceClusterIdentifier, username, userPassword)
    println("The ARN of the cluster is $arnClusterVal")
```

```
    println("9. Wait for DB instance to be ready")
    waitForClusterInstanceReady(dbInstanceClusterIdentifier)

    println("10. Get a list of instance classes available for the selected
 engine")
    val instanceClass = getListInstanceClasses()

    println("11. Create a database instance in the cluster.")
    val clusterDBARN = createDBInstanceCluster(dbInstanceIdentifier,
 dbInstanceClusterIdentifier, instanceClass)
    println("The ARN of the database is $clusterDBARN")

    println("12. Wait for DB instance to be ready")
    waitDBAuroraInstanceReady(dbInstanceIdentifier)

    println("13. Create a snapshot")
    createDBClusterSnapshot(dbInstanceClusterIdentifier, dbSnapshotIdentifier)

    println("14. Wait for DB snapshot to be ready")
    waitSnapshotReady(dbSnapshotIdentifier, dbInstanceClusterIdentifier)

    println("15. Delete the DB instance")
    deleteDBInstance(dbInstanceIdentifier)

    println("16. Delete the DB cluster")
    deleteCluster(dbInstanceClusterIdentifier)

    println("17. Delete the DB cluster group")
    if (clusterDBARN != null) {
        deleteDBClusterGroup(dbClusterGroupName, clusterDBARN)
    }
    println("The Scenario has successfully completed.")
}

@Throws(InterruptedException::class)
suspend fun deleteDBClusterGroup(dbClusterGroupName: String, clusterDBARN:
 String) {
    var isDataDel = false
    var didFind: Boolean
    var instanceARN: String

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        // Make sure that the database has been deleted.
```

```
        while (!isDataDel) {
            val response = rdsClient.describeDbInstances()
            val instanceList = response.dbInstances
            val listSize = instanceList?.size
            isDataDel = false
            didFind = false
            var index = 1
            if (instanceList != null) {
                for (instance in instanceList) {
                    instanceARN = instance.dbInstanceArn.toString()
                    if (instanceARN.compareTo(clusterDBARN) == 0) {
                        println("$clusterDBARN still exists")
                        didFind = true
                    }
                    if (index == listSize && !didFind) {
                        // Went through the entire list and did not find the
 database ARN.
                        isDataDel = true
                    }
                    delay(slTime * 1000)
                    index++
                }
            }
        }
        val clusterParameterGroupRequest = DeleteDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dbClusterGroupName
        }

        rdsClient.deleteDbClusterParameterGroup(clusterParameterGroupRequest)
        println("$dbClusterGroupName was deleted.")
    }
}

suspend fun deleteCluster(dbInstanceClusterIdentifier: String) {
    val deleteDbClusterRequest = DeleteDbClusterRequest {
        dbClusterIdentifier = dbInstanceClusterIdentifier
        skipFinalSnapshot = true
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        rdsClient.deleteDbCluster(deleteDbClusterRequest)
        println("$dbInstanceClusterIdentifier was deleted!")
    }
}
```

```
suspend fun deleteDBInstance(dbInstanceIdentifierVal: String) {
    val deleteDbInstanceRequest = DeleteDbInstanceRequest {
        dbInstanceIdentifier = dbInstanceIdentifierVal
        deleteAutomatedBackups = true
        skipFinalSnapshot = true
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.deleteDbInstance(deleteDbInstanceRequest)
        print("The status of the database is
 ${response.dbInstance?.dbInstanceStatus}")
    }
}

suspend fun waitSnapshotReady(dbSnapshotIdentifier: String?,
 dbInstanceClusterIdentifier: String?) {
    var snapshotReady = false
    var snapshotReadyStr: String
    println("Waiting for the snapshot to become available.")

    val snapshotsRequest = DescribeDbClusterSnapshotsRequest {
        dbClusterSnapshotIdentifier = dbSnapshotIdentifier
        dbClusterIdentifier = dbInstanceClusterIdentifier
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        while (!snapshotReady) {
            val response = rdsClient.describeDbClusterSnapshots(snapshotsRequest)
            val snapshotList = response.dbClusterSnapshots
            if (snapshotList != null) {
                for (snapshot in snapshotList) {
                    snapshotReadyStr = snapshot.status.toString()
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true
                    } else {
                        println(".")
                        delay(slTime * 5000)
                    }
                }
            }
        }
    }
    println("The Snapshot is available!")
```

```
}

suspend fun createDBClusterSnapshot(dbInstanceClusterIdentifier: String?,
 dbSnapshotIdentifier: String?) {
    val snapshotRequest = CreateDbClusterSnapshotRequest {
        dbClusterIdentifier = dbInstanceClusterIdentifier
        dbClusterSnapshotIdentifier = dbSnapshotIdentifier
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterSnapshot(snapshotRequest)
        println("The Snapshot ARN is
 ${response.dbClusterSnapshot?.dbClusterSnapshotArn}")
    }
}

suspend fun waitDBAuroraInstanceReady(dbInstanceIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")
    val instanceRequest = DescribeDbInstancesRequest {
        dbInstanceIdentifier = dbInstanceIdentifierVal
    }

    var endpoint = ""
    RdsClient { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
            val response = rdsClient.describeDbInstances(instanceRequest)
            response.dbInstances?.forEach { instance ->
                instanceReadyStr = instance.dbInstanceStatus.toString()
                if (instanceReadyStr.contains("available")) {
                    endpoint = instance.endpoint?.address.toString()
                    instanceReady = true
                } else {
                    print(".")
                    delay(sleepTime * 1000)
                }
            }
        }
    }
    println("Database instance is available! The connection endpoint is
 $endpoint")
}
```

```
suspend fun createDBInstanceCluster(dbInstanceIdentifierVal: String?,
 dbInstanceClusterIdentifierVal: String?, instanceClassVal: String?): String? {
    val instanceRequest = CreateDbInstanceRequest {
        dbInstanceIdentifier = dbInstanceIdentifierVal
        dbClusterIdentifier = dbInstanceClusterIdentifierVal
        engine = "aurora-mysql"
        dbInstanceClass = instanceClassVal
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbInstance(instanceRequest)
        print("The status is ${response.dbInstance?.dbInstanceStatus}")
        return response.dbInstance?.dbInstanceArn
    }
}

suspend fun getListInstanceClasses(): String {
    val optionsRequest = DescribeOrderableDbInstanceOptionsRequest {
        engine = "aurora-mysql"
        maxRecords = 20
    }
    var instanceClass = ""
    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeOrderableDbInstanceOptions(optionsRequest)
        response.orderableDbInstanceOptions?.forEach { instanceOption ->
            instanceClass = instanceOption.dbInstanceClass.toString()
            println("The instance class is ${instanceOption.dbInstanceClass}")
            println("The engine version is ${instanceOption.engineVersion}")
        }
    }
    return instanceClass
}

// Waits until the database instance is available.
suspend fun waitForClusterInstanceReady(dbClusterIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")

    val instanceRequest = DescribeDbClustersRequest {
        dbClusterIdentifier = dbClusterIdentifierVal
    }
```

```
    RdsClient { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
            val response = rdsClient.describeDbClusters(instanceRequest)
            response.dbClusters?.forEach { cluster ->
                instanceReadyStr = cluster.status.toString()
                if (instanceReadyStr.contains("available")) {
                    instanceReady = true
                } else {
                    print(".")
                    delay(sleepTime * 1000)
                }
            }
        }
    }
    println("Database cluster is available!")
}

suspend fun createDBCluster(dbParameterGroupFamilyVal: String?, dbName: String?,
 dbClusterIdentifierVal: String?, userName: String?, password: String?): String?
 {
    val clusterRequest = CreateDbClusterRequest {
        databaseName = dbName
        dbClusterIdentifier = dbClusterIdentifierVal
        dbClusterParameterGroupName = dbParameterGroupFamilyVal
        engine = "aurora-mysql"
        masterUsername = userName
        masterUserPassword = password
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbCluster(clusterRequest)
        return response.dbCluster?.dbClusterArn
    }
}

// Get a list of allowed engine versions.
suspend fun getAllowedClusterEngines(dbParameterGroupFamilyVal: String?) {
    val versionsRequest = DescribeDbEngineVersionsRequest {
        dbParameterGroupFamily = dbParameterGroupFamilyVal
        engine = "aurora-mysql"
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(versionsRequest)
```

```
        response.dbEngineVersions?.forEach { dbEngine ->
            println("The engine version is ${dbEngine.engineVersion}")
            println("The engine description is ${dbEngine.dbEngineDescription}")
        }
    }
}

// Modify the auto_increment_offset parameter.
suspend fun modifyDBClusterParas(dClusterGroupName: String?) {
    val parameter1 = Parameter {
        parameterName = "auto_increment_offset"
        applyMethod = ApplyMethod.fromValue("immediate")
        parameterValue = "5"
    }

    val paraList = ArrayList<Parameter>()
    paraList.add(parameter1)
    val groupRequest = ModifyDbClusterParameterGroupRequest {
        dbClusterParameterGroupName = dClusterGroupName
        parameters = paraList
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.modifyDbClusterParameterGroup(groupRequest)
        println("The parameter group ${response.dbClusterParameterGroupName} was
 successfully modified")
    }
}

suspend fun describeDbClusterParameters(dbCLusterGroupName: String?, flag: Int) {
    val dbParameterGroupsRequest: DescribeDbClusterParametersRequest
    dbParameterGroupsRequest = if (flag == 0) {
        DescribeDbClusterParametersRequest {
            dbClusterParameterGroupName = dbCLusterGroupName
        }
    } else {
        DescribeDbClusterParametersRequest {
            dbClusterParameterGroupName = dbCLusterGroupName
            source = "user"
        }
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
```

```
        val response =
 rdsClient.describeDbClusterParameters(dbParameterGroupsRequest)
        response.parameters?.forEach { para ->
            // Only print out information about either auto_increment_offset or
 auto_increment_increment.
            val paraName = para.parameterName
            if (paraName != null) {
                if (paraName.compareTo("auto_increment_offset") == 0 ||
 paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    println("*** The parameter value is  ${para.parameterValue}")
                    println("*** The parameter data type is ${para.dataType}")
                    println("*** The parameter description is
 ${para.description}")
                    println("*** The parameter allowed values  is
 ${para.allowedValues}")
                }
            }
        }
    }
}

suspend fun describeDbClusterParameterGroups(dbClusterGroupName: String?) {
    val groupsRequest = DescribeDbClusterParameterGroupsRequest {
        dbClusterParameterGroupName = dbClusterGroupName
        maxRecords = 20
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbClusterParameterGroups(groupsRequest)
        response.dbClusterParameterGroups?.forEach { group ->
            println("The group name is ${group.dbClusterParameterGroupName}")
            println("The group ARN is ${group.dbClusterParameterGroupArn}")
        }
    }
}

suspend fun createDBClusterParameterGroup(dbClusterGroupNameVal: String?,
 dbParameterGroupFamilyVal: String?) {
    val groupRequest = CreateDbClusterParameterGroupRequest {
        dbClusterParameterGroupName = dbClusterGroupNameVal
        dbParameterGroupFamily = dbParameterGroupFamilyVal
        description = "Created by using the AWS SDK for Kotlin"
    }
```

```
    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterParameterGroup(groupRequest)
        println("The group name is
 ${response.dbClusterParameterGroup?.dbClusterParameterGroupName}")
    }
}

suspend fun describeAuroraDBEngines() {
    val engineVersionsRequest = DescribeDbEngineVersionsRequest {
        engine = "aurora-mysql"
        defaultOnly = true
        maxRecords = 20
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(engineVersionsRequest)
        response.dbEngineVersions?.forEach { engineOb ->
            println("The name of the DB parameter group family for the database
 engine is ${engineOb.dbParameterGroupFamily}")
            println("The name of the database engine ${engineOb.engine}")
            println("The version number of the database engine
 ${engineOb.engineVersion}")
        }
    }
}
```

- API の詳細については、「AWS SDK for Kotlin API リファレンス」の以下のトピックを参照してください。
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups

- DescribeDBClusterParameters
- DescribeDBClusterSnapshots
- DescribeDBClusters
- DescribeDBEngineVersions
- DescribeDBInstances
- DescribeOrderableDBInstanceOptions
- ModifyDBClusterParameterGroup

Python

SDK for Python (Boto3)

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

コマンドプロンプトからインタラクティブのシナリオを実行します。

```
class AuroraClusterScenario:
    """Runs a scenario that shows how to get started using Aurora DB clusters."""

    def __init__(self, aurora_wrapper):
        """
        :param aurora_wrapper: An object that wraps Aurora DB cluster actions.
        """
        self.aurora_wrapper = aurora_wrapper

    def create_parameter_group(self, db_engine, parameter_group_name):
        """
        Shows how to get available engine versions for a specified database
 engine and
        create a DB cluster parameter group that is compatible with a selected
 engine family.

        :param db_engine: The database engine to use as a basis.
        :param parameter_group_name: The name given to the newly created
 parameter group.
```

```
        :return: The newly created parameter group.
        """
        print(
            f"Checking for an existing DB cluster parameter group named
{parameter_group_name}."
        )
        parameter_group =
self.aurora_wrapper.get_parameter_group(parameter_group_name)
        if parameter_group is None:
            print(f"Getting available database engine versions for {db_engine}.")
            engine_versions = self.aurora_wrapper.get_engine_versions(db_engine)
            families = list({ver["DBParameterGroupFamily"] for ver in
engine_versions})
            family_index = q.choose("Which family do you want to use? ",
families)
            print(f"Creating a DB cluster parameter group.")
            self.aurora_wrapper.create_parameter_group(
                parameter_group_name, families[family_index], "Example parameter
group."
            )
            parameter_group = self.aurora_wrapper.get_parameter_group(
                parameter_group_name
            )
        print(f"Parameter group
{parameter_group['DBClusterParameterGroupName']}:")
        pp(parameter_group)
        print("-" * 88)
        return parameter_group

    def set_user_parameters(self, parameter_group_name):
        """
        Shows how to get the parameters contained in a custom parameter group and
        update some of the parameter values in the group.

        :param parameter_group_name: The name of the parameter group to query and
modify.
        """
        print("Let's set some parameter values in your parameter group.")
        auto_inc_parameters = self.aurora_wrapper.get_parameters(
            parameter_group_name, name_prefix="auto_increment"
        )
        update_params = []
        for auto_inc in auto_inc_parameters:
            if auto_inc["IsModifiable"] and auto_inc["DataType"] == "integer":
```

```
                print(f"The {auto_inc['ParameterName']} parameter is described
as:")
                print(f"\t{auto_inc['Description']}")
                param_range = auto_inc["AllowedValues"].split("-")
                auto_inc["ParameterValue"] = str(
                    q.ask(
                        f"Enter a value between {param_range[0]} and
{param_range[1]}: ",
                        q.is_int,
                        q.in_range(int(param_range[0]), int(param_range[1])),
                    )
                )
                update_params.append(auto_inc)
        self.aurora_wrapper.update_parameters(parameter_group_name,
update_params)
        print(
            "You can get a list of parameters you've set by specifying a source
of 'user'."
        )
        user_parameters = self.aurora_wrapper.get_parameters(
            parameter_group_name, source="user"
        )
        pp(user_parameters)
        print("-" * 88)

    def create_cluster(self, cluster_name, db_engine, db_name, parameter_group):
        """
        Shows how to create an Aurora DB cluster that contains a database of a
specified
        type. The database is also configured to use a custom DB cluster
parameter group.

        :param cluster_name: The name given to the newly created DB cluster.
        :param db_engine: The engine of the created database.
        :param db_name: The name given to the created database.
        :param parameter_group: The parameter group that is associated with the
DB cluster.
        :return: The newly created DB cluster.
        """
        print("Checking for an existing DB cluster.")
        cluster = self.aurora_wrapper.get_db_cluster(cluster_name)
        if cluster is None:
            admin_username = q.ask(
```

```
                "Enter an administrator user name for the database: ",
q.non_empty
            )
            admin_password = q.ask(
                "Enter a password for the administrator (at least 8 characters):
",
                q.non_empty,
            )
            engine_versions = self.aurora_wrapper.get_engine_versions(
                db_engine, parameter_group["DBParameterGroupFamily"]
            )
            engine_choices = [ver["EngineVersionDescription"] for ver in
engine_versions]
            print("The available engines for your parameter group are:")
            engine_index = q.choose("Which engine do you want to use? ",
engine_choices)
            print(
                f"Creating DB cluster {cluster_name} and database {db_name}.\n"
                f"The DB cluster is configured to use\n"
                f"your custom parameter group
{parameter_group['DBClusterParameterGroupName']}\n"
                f"and selected engine {engine_choices[engine_index]}.\n"
                f"This typically takes several minutes."
            )
            cluster = self.aurora_wrapper.create_db_cluster(
                cluster_name,
                parameter_group["DBClusterParameterGroupName"],
                db_name,
                db_engine,
                engine_versions[engine_index]["EngineVersion"],
                admin_username,
                admin_password,
            )
            while cluster.get("Status") != "available":
                wait(30)
                cluster = self.aurora_wrapper.get_db_cluster(cluster_name)
            print("Cluster created and available.\n")
        print("Cluster data:")
        pp(cluster)
        print("-" * 88)
        return cluster

    def create_instance(self, cluster):
        """
```

```
        Shows how to create a DB instance in an existing Aurora DB cluster. A new
DB cluster
        contains no DB instances, so you must add one. The first DB instance that
is added
        to a DB cluster defaults to a read-write DB instance.

        :param cluster: The DB cluster where the DB instance is added.
        :return: The newly created DB instance.
        """
        print("Checking for an existing database instance.")
        cluster_name = cluster["DBClusterIdentifier"]
        db_inst = self.aurora_wrapper.get_db_instance(cluster_name)
        if db_inst is None:
            print("Let's create a database instance in your DB cluster.")
            print("First, choose a DB instance type:")
            inst_opts = self.aurora_wrapper.get_orderable_instances(
                cluster["Engine"], cluster["EngineVersion"]
            )
            inst_choices = list({opt["DBInstanceClass"] + ", storage type: " +
opt["StorageType"] for opt in inst_opts})
            inst_index = q.choose(
                "Which DB instance class do you want to use? ", inst_choices
            )
            print(
                f"Creating a database instance. This typically takes several
minutes."
            )
            db_inst = self.aurora_wrapper.create_instance_in_cluster(
                cluster_name, cluster_name, cluster["Engine"],
inst_opts[inst_index]["DBInstanceClass"]
            )
            while db_inst.get("DBInstanceStatus") != "available":
                wait(30)
                db_inst = self.aurora_wrapper.get_db_instance(cluster_name)
        print("Instance data:")
        pp(db_inst)
        print("-" * 88)
        return db_inst

    @staticmethod
    def display_connection(cluster):
        """
        Displays connection information about an Aurora DB cluster and tips on
how to
```

```
        connect to it.

        :param cluster: The DB cluster to display.
        """
        print(
            "You can now connect to your database using your favorite MySql
 client.\n"
            "One way to connect is by using the 'mysql' shell on an Amazon EC2
 instance\n"
            "that is running in the same VPC as your database cluster. Pass the
 endpoint,\n"
            "port, and administrator user name to 'mysql' and enter your password
\n"
            "when prompted:\n"
        )
        print(
            f"\n\tmysql -h {cluster['Endpoint']} -P {cluster['Port']} -u
 {cluster['MasterUsername']} -p\n"
        )
        print(
            "For more information, see the User Guide for Aurora:\n"
            "\thttps://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
CHAP_GettingStartedAurora.CreatingConnecting.Aurora.html#CHAP_GettingStartedAurora.Aurora
        )
        print("-" * 88)

    def create_snapshot(self, cluster_name):
        """
        Shows how to create a DB cluster snapshot and wait until it's available.

        :param cluster_name: The name of a DB cluster to snapshot.
        """
        if q.ask(
            "Do you want to create a snapshot of your DB cluster (y/n)? ",
 q.is_yesno
        ):
            snapshot_id = f"{cluster_name}-{uuid.uuid4()}"
            print(
                f"Creating a snapshot named {snapshot_id}. This typically takes a
 few minutes."
            )
            snapshot = self.aurora_wrapper.create_cluster_snapshot(
                snapshot_id, cluster_name
            )
```

```
            while snapshot.get("Status") != "available":
                wait(30)
                snapshot = self.aurora_wrapper.get_cluster_snapshot(snapshot_id)
            pp(snapshot)
            print("-" * 88)

    def cleanup(self, db_inst, cluster, parameter_group):
        """
        Shows how to clean up a DB instance, DB cluster, and DB cluster parameter
group.
        Before the DB cluster parameter group can be deleted, all associated DB
instances and
        DB clusters must first be deleted.

        :param db_inst: The DB instance to delete.
        :param cluster: The DB cluster to delete.
        :param parameter_group: The DB cluster parameter group to delete.
        """
        cluster_name = cluster["DBClusterIdentifier"]
        parameter_group_name = parameter_group["DBClusterParameterGroupName"]
        if q.ask(
            "\nDo you want to delete the database instance, DB cluster, and
parameter "
            "group (y/n)? ",
            q.is_yesno,
        ):
            print(f"Deleting database instance
{db_inst['DBInstanceIdentifier']}.")

self.aurora_wrapper.delete_db_instance(db_inst["DBInstanceIdentifier"])
            print(f"Deleting database cluster {cluster_name}.")
            self.aurora_wrapper.delete_db_cluster(cluster_name)
            print(
                "Waiting for the DB instance and DB cluster to delete.\n"
                "This typically takes several minutes."
            )
            while db_inst is not None or cluster is not None:
                wait(30)
                if db_inst is not None:
                    db_inst = self.aurora_wrapper.get_db_instance(
                        db_inst["DBInstanceIdentifier"]
                    )
                if cluster is not None:
                    cluster = self.aurora_wrapper.get_db_cluster(
```

```
                        cluster["DBClusterIdentifier"]
                    )
            print(f"Deleting parameter group {parameter_group_name}.")
            self.aurora_wrapper.delete_parameter_group(parameter_group_name)

    def run_scenario(self, db_engine, parameter_group_name, cluster_name,
db_name):
        print("-" * 88)
        print(
            "Welcome to the Amazon Relational Database Service (Amazon RDS) get
started\n"
            "with Aurora DB clusters demo."
        )
        print("-" * 88)

        parameter_group = self.create_parameter_group(db_engine,
parameter_group_name)
        self.set_user_parameters(parameter_group_name)
        cluster = self.create_cluster(cluster_name, db_engine, db_name,
parameter_group)
        wait(5)
        db_inst = self.create_instance(cluster)
        self.display_connection(cluster)
        self.create_snapshot(cluster_name)
        self.cleanup(db_inst, cluster, parameter_group)

        print("\nThanks for watching!")
        print("-" * 88)


if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
    try:
        scenario = AuroraClusterScenario(AuroraWrapper.from_client())
        scenario.run_scenario(
            "aurora-mysql",
            "doc-example-cluster-parameter-group",
            "doc-example-aurora",
            "docexampledb",
        )
    except Exception:
        logging.exception("Something went wrong with the demo.")
```

Aurora アクションを管理するためにシナリオによって呼び出される関数を定義します。

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_parameter_group(self, parameter_group_name):
        """
        Gets a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to retrieve.
        :return: The requested parameter group.
        """
        try:
            response = self.rds_client.describe_db_cluster_parameter_groups(
                DBClusterParameterGroupName=parameter_group_name
            )
            parameter_group = response["DBClusterParameterGroups"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBParameterGroupNotFound":
                logger.info("Parameter group %s does not exist.",
 parameter_group_name)
            else:
                logger.error(
                    "Couldn't get parameter group %s. Here's why: %s: %s",
                    parameter_group_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
```

```
                raise
        else:
            return parameter_group


    def create_parameter_group(
        self, parameter_group_name, parameter_group_family, description
    ):
        """
        Creates a DB cluster parameter group that is based on the specified
parameter group
        family.

        :param parameter_group_name: The name of the newly created parameter
group.
        :param parameter_group_family: The family that is used as the basis of
the new
                                       parameter group.
        :param description: A description given to the parameter group.
        :return: Data about the newly created parameter group.
        """
        try:
            response = self.rds_client.create_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                DBParameterGroupFamily=parameter_group_family,
                Description=description,
            )
        except ClientError as err:
            logger.error(
                "Couldn't create parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def delete_parameter_group(self, parameter_group_name):
        """
        Deletes a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to delete.
```

```
        :return: Data about the parameter group.
        """
        try:
            response = self.rds_client.delete_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name
            )
        except ClientError as err:
            logger.error(
                "Couldn't delete parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def get_parameters(self, parameter_group_name, name_prefix="", source=None):
        """
        Gets the parameters that are contained in a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to query.
        :param name_prefix: When specified, the retrieved list of parameters is
filtered
                            to contain only parameters that start with this
prefix.
        :param source: When specified, only parameters from this source are
retrieved.
                       For example, a source of 'user' retrieves only parameters
that
                       were set by a user.
        :return: The list of requested parameters.
        """
        try:
            kwargs = {"DBClusterParameterGroupName": parameter_group_name}
            if source is not None:
                kwargs["Source"] = source
            parameters = []
            paginator =
self.rds_client.get_paginator("describe_db_cluster_parameters")
            for page in paginator.paginate(**kwargs):
                parameters += [
                    p
```

```
                    for p in page["Parameters"]
                    if p["ParameterName"].startswith(name_prefix)
                ]
        except ClientError as err:
            logger.error(
                "Couldn't get parameters for %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return parameters


    def update_parameters(self, parameter_group_name, update_parameters):
        """
        Updates parameters in a custom DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to update.
        :param update_parameters: The parameters to update in the group.
        :return: Data about the modified parameter group.
        """
        try:
            response = self.rds_client.modify_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                Parameters=update_parameters,
            )
        except ClientError as err:
            logger.error(
                "Couldn't update parameters in %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def get_db_cluster(self, cluster_name):
        """
        Gets data about an Aurora DB cluster.
```

```
        :param cluster_name: The name of the DB cluster to retrieve.
        :return: The retrieved DB cluster.
        """
        try:
            response = self.rds_client.describe_db_clusters(
                DBClusterIdentifier=cluster_name
            )
            cluster = response["DBClusters"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBClusterNotFoundFault":
                logger.info("Cluster %s does not exist.", cluster_name)
            else:
                logger.error(
                    "Couldn't verify the existence of DB cluster %s. Here's why:
%s: %s",
                    cluster_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return cluster


    def create_db_cluster(
        self,
        cluster_name,
        parameter_group_name,
        db_name,
        db_engine,
        db_engine_version,
        admin_name,
        admin_password,
    ):
        """
        Creates a DB cluster that is configured to use the specified parameter
group.
        The newly created DB cluster contains a database that uses the specified
engine and
        engine version.

        :param cluster_name: The name of the DB cluster to create.
        :param parameter_group_name: The name of the parameter group to associate
with
```

```
                                    the DB cluster.
        :param db_name: The name of the database to create.
        :param db_engine: The database engine of the database that is created,
such as MySql.
        :param db_engine_version: The version of the database engine.
        :param admin_name: The user name of the database administrator.
        :param admin_password: The password of the database administrator.
        :return: The newly created DB cluster.
        """
        try:
            response = self.rds_client.create_db_cluster(
                DatabaseName=db_name,
                DBClusterIdentifier=cluster_name,
                DBClusterParameterGroupName=parameter_group_name,
                Engine=db_engine,
                EngineVersion=db_engine_version,
                MasterUsername=admin_name,
                MasterUserPassword=admin_password,
            )
            cluster = response["DBCluster"]
        except ClientError as err:
            logger.error(
                "Couldn't create database %s. Here's why: %s: %s",
                db_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return cluster


    def delete_db_cluster(self, cluster_name):
        """
        Deletes a DB cluster.

        :param cluster_name: The name of the DB cluster to delete.
        """
        try:
            self.rds_client.delete_db_cluster(
                DBClusterIdentifier=cluster_name, SkipFinalSnapshot=True
            )
            logger.info("Deleted DB cluster %s.", cluster_name)
        except ClientError:
```

```
            logger.exception("Couldn't delete DB cluster %s.", cluster_name)
            raise


    def create_cluster_snapshot(self, snapshot_id, cluster_id):
        """
        Creates a snapshot of a DB cluster.

        :param snapshot_id: The ID to give the created snapshot.
        :param cluster_id: The DB cluster to snapshot.
        :return: Data about the newly created snapshot.
        """
        try:
            response = self.rds_client.create_db_cluster_snapshot(
                DBClusterSnapshotIdentifier=snapshot_id,
DBClusterIdentifier=cluster_id
            )
            snapshot = response["DBClusterSnapshot"]
        except ClientError as err:
            logger.error(
                "Couldn't create snapshot of %s. Here's why: %s: %s",
                cluster_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot


    def get_cluster_snapshot(self, snapshot_id):
        """
        Gets a DB cluster snapshot.

        :param snapshot_id: The ID of the snapshot to retrieve.
        :return: The retrieved snapshot.
        """
        try:
            response = self.rds_client.describe_db_cluster_snapshots(
                DBClusterSnapshotIdentifier=snapshot_id
            )
            snapshot = response["DBClusterSnapshots"][0]
        except ClientError as err:
            logger.error(
```

```
                "Couldn't get DB cluster snapshot %s. Here's why: %s: %s",
                snapshot_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot


    def create_instance_in_cluster(
        self, instance_id, cluster_id, db_engine, instance_class
    ):
        """
        Creates a database instance in an existing DB cluster. The first database
that is
        created defaults to a read-write DB instance.

        :param instance_id: The ID to give the newly created DB instance.
        :param cluster_id: The ID of the DB cluster where the DB instance is
created.
        :param db_engine: The database engine of a database to create in the DB
instance.
                          This must be compatible with the configured parameter
group
                          of the DB cluster.
        :param instance_class: The DB instance class for the newly created DB
instance.
        :return: Data about the newly created DB instance.
        """
        try:
            response = self.rds_client.create_db_instance(
                DBInstanceIdentifier=instance_id,
                DBClusterIdentifier=cluster_id,
                Engine=db_engine,
                DBInstanceClass=instance_class,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't create DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
```

```
            )
            raise
        else:
            return db_inst


    def get_engine_versions(self, engine, parameter_group_family=None):
        """
        Gets database engine versions that are available for the specified engine
        and parameter group family.

        :param engine: The database engine to look up.
        :param parameter_group_family: When specified, restricts the returned
list of
                                       engine versions to those that are
compatible with
                                       this parameter group family.
        :return: The list of database engine versions.
        """
        try:
            kwargs = {"Engine": engine}
            if parameter_group_family is not None:
                kwargs["DBParameterGroupFamily"] = parameter_group_family
            response = self.rds_client.describe_db_engine_versions(**kwargs)
            versions = response["DBEngineVersions"]
        except ClientError as err:
            logger.error(
                "Couldn't get engine versions for %s. Here's why: %s: %s",
                engine,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return versions


    def get_orderable_instances(self, db_engine, db_engine_version):
        """
        Gets DB instance options that can be used to create DB instances that are
        compatible with a set of specifications.

        :param db_engine: The database engine that must be supported by the DB
instance.
```

```
        :param db_engine_version: The engine version that must be supported by
the DB instance.
        :return: The list of DB instance options that can be used to create a
compatible DB instance.
        """
        try:
            inst_opts = []
            paginator = self.rds_client.get_paginator(
                "describe_orderable_db_instance_options"
            )
            for page in paginator.paginate(
                Engine=db_engine, EngineVersion=db_engine_version
            ):
                inst_opts += page["OrderableDBInstanceOptions"]
        except ClientError as err:
            logger.error(
                "Couldn't get orderable DB instances. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return inst_opts


    def get_db_instance(self, instance_id):
        """
        Gets data about a DB instance.

        :param instance_id: The ID of the DB instance to retrieve.
        :return: The retrieved DB instance.
        """
        try:
            response = self.rds_client.describe_db_instances(
                DBInstanceIdentifier=instance_id
            )
            db_inst = response["DBInstances"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBInstanceNotFound":
                logger.info("Instance %s does not exist.", instance_id)
            else:
                logger.error(
                    "Couldn't get DB instance %s. Here's why: %s: %s",
                    instance_id,
```

```
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return db_inst


    def delete_db_instance(self, instance_id):
        """
        Deletes a DB instance.

        :param instance_id: The ID of the DB instance to delete.
        :return: Data about the deleted DB instance.
        """
        try:
            response = self.rds_client.delete_db_instance(
                DBInstanceIdentifier=instance_id,
                SkipFinalSnapshot=True,
                DeleteAutomatedBackups=True,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't delete DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- API の詳細については、「AWS SDK for Python (Boto3) API リファレンス」の以下のトピックを参照してください。
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot

- CreateDBInstance
- DeleteDBCluster
- DeleteDBClusterParameterGroup
- DeleteDBInstance
- DescribeDBClusterParameterGroups
- DescribeDBClusterParameters
- DescribeDBClusterSnapshots
- DescribeDBClusters
- DescribeDBEngineVersions
- DescribeDBInstances
- DescribeOrderableDBInstanceOptions
- ModifyDBClusterParameterGroup

Rust

SDK for Rust

> **Note**
>
> GitHub には、その他のリソースもあります。用例一覧を検索し、AWS コード例リポジトリでの設定と実行の方法を確認してください。

Aurora シナリオのシナリオ固有の関数を含むライブラリ。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use phf::{phf_set, Set};
use secrecy::SecretString;
use std::{collections::HashMap, fmt::Display, time::Duration};

use aws_sdk_rds::{
    error::ProvideErrorMetadata,

 operation::create_db_cluster_parameter_group::CreateDbClusterParameterGroupOutput,
```

```
    types::{DbCluster, DbClusterParameterGroup, DbClusterSnapshot, DbInstance,
 Parameter},
};
use sdk_examples_test_utils::waiter::Waiter;
use tracing::{info, trace, warn};

const DB_ENGINE: &str = "aurora-mysql";
const DB_CLUSTER_PARAMETER_GROUP_NAME: &str =
 "RustSDKCodeExamplesDBParameterGroup";
const DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION: &str =
    "Parameter Group created by Rust SDK Code Example";
const DB_CLUSTER_IDENTIFIER: &str = "RustSDKCodeExamplesDBCluster";
const DB_INSTANCE_IDENTIFIER: &str = "RustSDKCodeExamplesDBInstance";

static FILTER_PARAMETER_NAMES: Set<&'static str> = phf_set! {
    "auto_increment_offset",
    "auto_increment_increment",
};

#[derive(Debug, PartialEq, Eq)]
struct MetadataError {
    message: Option<String>,
    code: Option<String>,
}

impl MetadataError {
    fn from(err: &dyn ProvideErrorMetadata) -> Self {
        MetadataError {
            message: err.message().map(String::from),
            code: err.code().map(String::from),
        }
    }
}

impl Display for MetadataError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        let display = match (&self.message, &self.code) {
            (None, None) => "Unknown".to_string(),
            (None, Some(code)) => format!("({code})"),
            (Some(message), None) => message.to_string(),
            (Some(message), Some(code)) => format!("{message} ({code})"),
        };
        write!(f, "{display}")
    }
```

```
}

#[derive(Debug, PartialEq, Eq)]
pub struct ScenarioError {
    message: String,
    context: Option<MetadataError>,
}

impl ScenarioError {
    pub fn with(message: impl Into<String>) -> Self {
        ScenarioError {
            message: message.into(),
            context: None,
        }
    }

    pub fn new(message: impl Into<String>, err: &dyn ProvideErrorMetadata) ->
 Self {
        ScenarioError {
            message: message.into(),
            context: Some(MetadataError::from(err)),
        }
    }
}

impl std::error::Error for ScenarioError {}
impl Display for ScenarioError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match &self.context {
            Some(c) => write!(f, "{}: {}", self.message, c),
            None => write!(f, "{}", self.message),
        }
    }
}

// Parse the ParameterName, Description, and AllowedValues values and display
 them.
#[derive(Debug)]
pub struct AuroraScenarioParameter {
    name: String,
    allowed_values: String,
    current_value: String,
}
```

```
impl Display for AuroraScenarioParameter {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(
            f,
            "{}: {} (allowed: {})",
            self.name, self.current_value, self.allowed_values
        )
    }
}

impl From<aws_sdk_rds::types::Parameter> for AuroraScenarioParameter {
    fn from(value: aws_sdk_rds::types::Parameter) -> Self {
        AuroraScenarioParameter {
            name: value.parameter_name.unwrap_or_default(),
            allowed_values: value.allowed_values.unwrap_or_default(),
            current_value: value.parameter_value.unwrap_or_default(),
        }
    }
}

pub struct AuroraScenario {
    rds: crate::rds::Rds,
    engine_family: Option<String>,
    engine_version: Option<String>,
    instance_class: Option<String>,
    db_cluster_parameter_group: Option<DbClusterParameterGroup>,
    db_cluster_identifier: Option<String>,
    db_instance_identifier: Option<String>,
    username: Option<String>,
    password: Option<SecretString>,
}

impl AuroraScenario {
    pub fn new(client: crate::rds::Rds) -> Self {
        AuroraScenario {
            rds: client,
            engine_family: None,
            engine_version: None,
            instance_class: None,
            db_cluster_parameter_group: None,
            db_cluster_identifier: None,
            db_instance_identifier: None,
            username: None,
            password: None,
```

```
        }
    }

    // snippet-start:[rust.aurora.get_engines.usage]
    // Get available engine families for Aurora MySql.
rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
    pub async fn get_engines(&self) -> Result<HashMap<String, Vec<String>>,
ScenarioError> {
        let describe_db_engine_versions =
self.rds.describe_db_engine_versions(DB_ENGINE).await;
        trace!(versions=?describe_db_engine_versions, "full list of versions");

        if let Err(err) = describe_db_engine_versions {
            return Err(ScenarioError::new(
                "Failed to retrieve DB Engine Versions",
                &err,
            ));
        };

        let version_count = describe_db_engine_versions
            .as_ref()
            .map(|o| o.db_engine_versions().len())
            .unwrap_or_default();
        info!(version_count, "got list of versions");

        // Create a map of engine families to their available versions.
        let mut versions = HashMap::<String, Vec<String>>::new();
        describe_db_engine_versions
            .unwrap()
            .db_engine_versions()
            .iter()
            .filter_map(
                |v| match (&v.db_parameter_group_family, &v.engine_version) {
                    (Some(family), Some(version)) => Some((family.clone(),
version.clone())),
                    _ => None,
                },
            )
            .for_each(|(family, version)|
versions.entry(family).or_default().push(version));

        Ok(versions)
    }
```

```
    // snippet-end:[rust.aurora.get_engines.usage]

    // snippet-start:[rust.aurora.get_instance_classes.usage]
    pub async fn get_instance_classes(&self) -> Result<Vec<String>,
ScenarioError> {
        let describe_orderable_db_instance_options_items = self
            .rds
            .describe_orderable_db_instance_options(
                DB_ENGINE,
                self.engine_version
                    .as_ref()
                    .expect("engine version for db instance options")
                    .as_str(),
            )
            .await;

        describe_orderable_db_instance_options_items
            .map(|options| {
                options
                    .iter()
                    .map(|o|
o.db_instance_class().unwrap_or_default().to_string())
                    .collect::<Vec<String>>()
            })
            .map_err(|err| ScenarioError::new("Could not get available instance
classes", &err))
    }
    // snippet-end:[rust.aurora.get_instance_classes.usage]

    // snippet-start:[rust.aurora.set_engine.usage]
    // Select an engine family and create a custom DB cluster parameter group.
rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
    pub async fn set_engine(&mut self, engine: &str, version: &str) -> Result<(),
ScenarioError> {
        self.engine_family = Some(engine.to_string());
        self.engine_version = Some(version.to_string());
        let create_db_cluster_parameter_group = self
            .rds
            .create_db_cluster_parameter_group(
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION,
                engine,
            )
            .await;
```

```
        match create_db_cluster_parameter_group {
            Ok(CreateDbClusterParameterGroupOutput {
                db_cluster_parameter_group: None,
                ..
            }) => {
                return Err(ScenarioError::with(
                    "CreateDBClusterParameterGroup had empty response",
                ));
            }
            Err(error) => {
                if error.code() == Some("DBParameterGroupAlreadyExists") {
                    info!("Cluster Parameter Group already exists, nothing to
do");
                } else {
                    return Err(ScenarioError::new(
                        "Could not create Cluster Parameter Group",
                        &error,
                    ));
                }
            }
            _ => {
                info!("Created Cluster Parameter Group");
            }
        }

        Ok(())
    }
    // snippet-end:[rust.aurora.set_engine.usage]

    pub fn set_instance_class(&mut self, instance_class: Option<String>) {
        self.instance_class = instance_class;
    }

    pub fn set_login(&mut self, username: Option<String>, password:
Option<SecretString>) {
        self.username = username;
        self.password = password;
    }

    pub async fn connection_string(&self) -> Result<String, ScenarioError> {
        let cluster = self.get_cluster().await?;
        let endpoint = cluster.endpoint().unwrap_or_default();
        let port = cluster.port().unwrap_or_default();
```

```
        let username = cluster.master_username().unwrap_or_default();
        Ok(format!("mysql -h {endpoint} -P {port} -u {username} -p"))
    }

    // snippet-start:[rust.aurora.get_cluster.usage]
    pub async fn get_cluster(&self) -> Result<DbCluster, ScenarioError> {
        let describe_db_clusters_output = self
            .rds
            .describe_db_clusters(
                self.db_cluster_identifier
                    .as_ref()
                    .expect("cluster identifier")
                    .as_str(),
            )
            .await;
        if let Err(err) = describe_db_clusters_output {
            return Err(ScenarioError::new("Failed to get cluster", &err));
        }

        let db_cluster = describe_db_clusters_output
            .unwrap()
            .db_clusters
            .and_then(|output| output.first().cloned());

        db_cluster.ok_or_else(|| ScenarioError::with("Did not find the cluster"))
    }
    // snippet-end:[rust.aurora.get_cluster.usage]

    // snippet-start:[rust.aurora.cluster_parameters.usage]
    // Get the parameter group. rds.DescribeDbClusterParameterGroups
    // Get parameters in the group. This is a long list so you will have to
paginate. Find the auto_increment_offset and auto_increment_increment parameters
(by ParameterName). rds.DescribeDbClusterParameters
    // Parse the ParameterName, Description, and AllowedValues values and display
them.
    pub async fn cluster_parameters(&self) ->
Result<Vec<AuroraScenarioParameter>, ScenarioError> {
        let parameters_output = self
            .rds
            .describe_db_cluster_parameters(DB_CLUSTER_PARAMETER_GROUP_NAME)
            .await;

        if let Err(err) = parameters_output {
            return Err(ScenarioError::new(
```

```
                format!("Failed to retrieve parameters for
{DB_CLUSTER_PARAMETER_GROUP_NAME}"),
                &err,
            ));
        }

        let parameters = parameters_output
            .unwrap()
            .into_iter()
            .flat_map(|p| p.parameters.unwrap_or_default().into_iter())
            .filter(|p|
FILTER_PARAMETER_NAMES.contains(p.parameter_name().unwrap_or_default()))
            .map(AuroraScenarioParameter::from)
            .collect::<Vec<_>>();

        Ok(parameters)
    }
    // snippet-end:[rust.aurora.cluster_parameters.usage]

    // snippet-start:[rust.aurora.update_auto_increment.usage]
    // Modify both the auto_increment_offset and auto_increment_increment
parameters in one call in the custom parameter group. Set their ParameterValue
fields to a new allowable value. rds.ModifyDbClusterParameterGroup.
    pub async fn update_auto_increment(
        &self,
        offset: u8,
        increment: u8,
    ) -> Result<(), ScenarioError> {
        let modify_db_cluster_parameter_group = self
            .rds
            .modify_db_cluster_parameter_group(
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value(format!("{offset}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value(format!("{increment}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ],
```

```
            )
            .await;

        if let Err(error) = modify_db_cluster_parameter_group {
            return Err(ScenarioError::new(
                "Failed to modify cluster parameter group",
                &error,
            ));
        }

        Ok(())
    }
    // snippet-end:[rust.aurora.update_auto_increment.usage]

    // snippet-start:[rust.aurora.start_cluster_and_instance.usage]
    // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
    // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
    // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
    // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

    // Create a database instance in the cluster.
    // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
    pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
        if self.password.is_none() {
            return Err(ScenarioError::with(
                "Must set Secret Password before starting a cluster",
            ));
        }
        let create_db_cluster = self
            .rds
            .create_db_cluster(
                DB_CLUSTER_IDENTIFIER,
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_ENGINE,
                self.engine_version.as_deref().expect("engine version"),
                self.username.as_deref().expect("username"),
```

```
                self.password
                    .replace(SecretString::new("".to_string()))
                    .expect("password"),
            )
            .await;
        if let Err(err) = create_db_cluster {
            return Err(ScenarioError::new(
                "Failed to create DB Cluster with cluster group",
                &err,
            ));
        }

        self.db_cluster_identifier = create_db_cluster
            .unwrap()
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }
```

```
        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
```

```
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }
    // snippet-end:[rust.aurora.start_cluster_and_instance.usage]

    // snippet-start:[rust.aurora.snapshot.usage]
    // Create a snapshot of the DB cluster. rds.CreateDbClusterSnapshot.
    // Wait for the snapshot to create. rds.DescribeDbClusterSnapshots until
Status == 'available'.
    pub async fn snapshot(&self, name: &str) -> Result<DbClusterSnapshot,
ScenarioError> {
        let id = self.db_cluster_identifier.as_deref().unwrap_or_default();
        let snapshot = self
            .rds
            .snapshot_cluster(id, format!("{id}_{name}").as_str())
```

```
            .await;
        match snapshot {
            Ok(output) => match output.db_cluster_snapshot {
                Some(snapshot) => Ok(snapshot),
                None => Err(ScenarioError::with("Missing Snapshot")),
            },
            Err(err) => Err(ScenarioError::new("Failed to create snapshot",
&err)),
        }
    }
    // snippet-end:[rust.aurora.snapshot.usage]

    // snippet-start:[rust.aurora.clean_up.usage]
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
```

```
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
```

```
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }
```

```
        // Delete the DB cluster parameter group.
 rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
 DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }
    // snippet-end:[rust.aurora.clean_up.usage]
}

#[cfg(test)]
pub mod tests;
```

RDS Client ラッパーのオートモックを使用してライブラリをテストします。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use crate::rds::MockRdsImpl;

use super::*;
```

```
use std::io::{Error, ErrorKind};

use assert_matches::assert_matches;
use aws_sdk_rds::{
    error::SdkError,
    operation::{
        create_db_cluster::{CreateDBClusterError, CreateDbClusterOutput},
        create_db_cluster_parameter_group::CreateDBClusterParameterGroupError,
        create_db_cluster_snapshot::{CreateDBClusterSnapshotError,
 CreateDbClusterSnapshotOutput},
        create_db_instance::{CreateDBInstanceError, CreateDbInstanceOutput},
        delete_db_cluster::DeleteDbClusterOutput,
        delete_db_cluster_parameter_group::DeleteDbClusterParameterGroupOutput,
        delete_db_instance::DeleteDbInstanceOutput,
        describe_db_cluster_endpoints::DescribeDbClusterEndpointsOutput,
        describe_db_cluster_parameters::{
            DescribeDBClusterParametersError, DescribeDbClusterParametersOutput,
        },
        describe_db_clusters::{DescribeDBClustersError,
 DescribeDbClustersOutput},
        describe_db_engine_versions::{
            DescribeDBEngineVersionsError, DescribeDbEngineVersionsOutput,
        },
        describe_db_instances::{DescribeDBInstancesError,
 DescribeDbInstancesOutput},

 describe_orderable_db_instance_options::DescribeOrderableDBInstanceOptionsError,
        modify_db_cluster_parameter_group::{
            ModifyDBClusterParameterGroupError,
 ModifyDbClusterParameterGroupOutput,
        },
    },
    types::{
        error::DbParameterGroupAlreadyExistsFault, DbClusterEndpoint,
 DbEngineVersion,
        OrderableDbInstanceOption,
    },
};
use aws_smithy_runtime_api::http::{Response, StatusCode};
use aws_smithy_types::body::SdkBody;
use mockall::predicate::eq;
use secrecy::ExposeSecret;
```

```
// snippet-start:[rust.aurora.set_engine.test]
#[tokio::test]
async fn test_scenario_set_engine() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert_eq!(set_engine, Ok(()));
    assert_eq!(Some("aurora-mysql"), scenario.engine_family.as_deref());
    assert_eq!(Some("aurora-mysql8.0"), scenario.engine_version.as_deref());
}

#[tokio::test]
async fn test_scenario_set_engine_not_create() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _|
 Ok(CreateDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
```

```
    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}

#[tokio::test]
async fn test_scenario_set_engine_param_group_exists() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .withf(|_, _, _| true)
        .return_once(|_, _, _| {
            Err(SdkError::service_error(

 CreateDBClusterParameterGroupError::DbParameterGroupAlreadyExistsFault(
                    DbParameterGroupAlreadyExistsFault::builder().build(),
                ),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}
// snippet-end:[rust.aurora.set_engine.test]

// snippet-start:[rust.aurora.get_engines.test]
#[tokio::test]
async fn test_scenario_get_engines() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Ok(DescribeDbEngineVersionsOutput::builder()
```

```
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1a")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1b")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f2")
                        .engine_version("f2a")
                        .build(),
                )
                .db_engine_versions(DbEngineVersion::builder().build())
                .build())
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;

    assert_eq!(
        versions_map,
        Ok(HashMap::from([
            ("f1".into(), vec!["f1a".into(), "f1b".into()]),
            ("f2".into(), vec!["f2a".into()])
        ]))
    );
}

#[tokio::test]
async fn test_scenario_get_engines_failed() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Err(SdkError::service_error(
```

```
                DescribeDBEngineVersionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_engine_versions error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;
    assert_matches!(
        versions_map,
        Err(ScenarioError { message, context: _ }) if message == "Failed to
 retrieve DB Engine Versions"
    );
}
// snippet-end:[rust.aurora.get_engines.test]

// snippet-start:[rust.aurora.get_instance_classes.test]
#[tokio::test]
async fn test_scenario_get_instance_classes() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Ok(vec![
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t2")
```

```
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t3")
                    .build(),
            ])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario
        .set_engine("aurora-mysql", "aurora-mysql8.0")
        .await
        .expect("set engine");

    let instance_classes = scenario.get_instance_classes().await;

    assert_eq!(
        instance_classes,
        Ok(vec!["t1".into(), "t2".into(), "t3".into()])
    );
}

#[tokio::test]
async fn test_scenario_get_instance_classes_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Err(SdkError::service_error(

 DescribeOrderableDBInstanceOptionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_orderable_db_instance_options_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_family = Some("aurora-mysql".into());
    scenario.engine_version = Some("aurora-mysql8.0".into());
```

```
    let instance_classes = scenario.get_instance_classes().await;

    assert_matches!(
        instance_classes,
        Err(ScenarioError {message, context: _}) if message == "Could not get
 available instance classes"
    );
}
// snippet-end:[rust.aurora.get_instance_classes.test]

// snippet-start:[rust.aurora.get_cluster.test]
#[tokio::test]
async fn test_scenario_get_cluster() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;

    assert!(cluster.is_ok());
}

#[tokio::test]
async fn test_scenario_get_cluster_missing_cluster() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });
```

```
    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| Ok(DescribeDbClustersOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;

    assert_matches!(cluster, Err(ScenarioError { message, context: _ }) if
 message == "Did not find the cluster");
}

#[tokio::test]
async fn test_scenario_get_cluster_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_clusters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;
```

```
    assert_matches!(cluster, Err(ScenarioError { message, context: _ }) if
 message == "Failed to get cluster");
}
// snippet-end:[rust.aurora.get_cluster.test]

#[tokio::test]
async fn test_scenario_connection_string() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .endpoint("test_endpoint")
                        .port(3306)
                        .master_username("test_username")
                        .build(),
                )
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let connection_string = scenario.connection_string().await;

    assert_eq!(
        connection_string,
        Ok("mysql -h test_endpoint -P 3306 -u test_username -p".into())
    );
}

// snippet-start:[rust.aurora.cluster_parameters.test]
#[tokio::test]
async fn test_scenario_cluster_parameters() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Ok(vec![DescribeDbClusterParametersOutput::builder()
```

```
                .parameters(Parameter::builder().parameter_name("a").build())
                .parameters(Parameter::builder().parameter_name("b").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("c").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("d").build())
                .build()])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());

    let params = scenario.cluster_parameters().await.expect("cluster params");
    let names: Vec<String> = params.into_iter().map(|p| p.name).collect();
    assert_eq!(
        names,
        vec!["auto_increment_offset", "auto_increment_increment"]
    );
}

#[tokio::test]
async fn test_scenario_cluster_parameters_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClusterParametersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_cluster_parameters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
```

```
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let params = scenario.cluster_parameters().await;
    assert_matches!(params, Err(ScenarioError { message, context: _ }) if message
 == "Failed to retrieve parameters for RustSDKCodeExamplesDBParameterGroup");
}
// snippet-end:[rust.aurora.cluster_parameters.test]

// snippet-start:[rust.aurora.update_auto_increment.test]
#[tokio::test]
async fn test_scenario_update_auto_increment() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .withf(|name, params| {
            assert_eq!(name, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(
                params,
                &vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value("10")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value("20")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ]
            );
            true
        })
        .return_once(|_, _|
 Ok(ModifyDbClusterParameterGroupOutput::builder().build()));

    let scenario = AuroraScenario::new(mock_rds);

    scenario
        .update_auto_increment(10, 20)
        .await
```

```
        .expect("update auto increment");
}

#[tokio::test]
async fn test_scenario_update_auto_increment_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .return_once(|_, _| {
            Err(SdkError::service_error(

 ModifyDBClusterParameterGroupError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "modify_db_cluster_parameter_group_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let update = scenario.update_auto_increment(10, 20).await;
    assert_matches!(update, Err(ScenarioError { message, context: _}) if message
 == "Failed to modify cluster parameter group");
}
// snippet-end:[rust.aurora.update_auto_increment.test]

// snippet-start:[rust.aurora.start_cluster_and_instance.test]
#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
```

```
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
```

```
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

  .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

```
#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
```

```
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
```

```
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
```

```
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
```

```
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
// snippet-end:[rust.aurora.start_cluster_and_instance.test]

// snippet-start:[rust.aurora.clean_up.test]
#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
```

```
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
```

```
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
```

```
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
```

```
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.get(0), Some(ScenarioError {message, context: _}) if
 message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
// snippet-end:[rust.aurora.clean_up.test]

// snippet-start:[rust.aurora.snapshot.test]
```

```
#[tokio::test]
async fn test_scenario_snapshot() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
        .times(1)
        .return_once(|_, _| {
            Ok(CreateDbClusterSnapshotOutput::builder()
                .db_cluster_snapshot(
                    DbClusterSnapshot::builder()
                        .db_cluster_identifier("MockCluster")

 .db_cluster_snapshot_identifier("MockCluster_MockSnapshot")
                        .build(),
                )
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert!(create_snapshot.is_ok());
}

#[tokio::test]
async fn test_scenario_snapshot_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
        .times(1)
        .return_once(|_, _| {
            Err(SdkError::service_error(
                CreateDBClusterSnapshotError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create snapshot error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });
```

```
    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert_matches!(create_snapshot, Err(ScenarioError { message, context: _}) if
 message == "Failed to create snapshot");
}

#[tokio::test]
async fn test_scenario_snapshot_invalid() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
        .times(1)
        .return_once(|_, _|
 Ok(CreateDbClusterSnapshotOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert_matches!(create_snapshot, Err(ScenarioError { message, context: _}) if
 message == "Missing Snapshot");
}
// snippet-end:[rust.aurora.snapshot.test]
```

ユーザーが一部の決定を下せるように inquirer を使用してシナリオを最初から最後まで実行するためのバイナリ。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use std::fmt::Display;

use anyhow::anyhow;
use aurora_code_examples::{
    aurora_scenario::{AuroraScenario, ScenarioError},
    rds::Rds as RdsClient,
};
use aws_sdk_rds::Client;
use inquire::{validator::StringValidator, CustomUserError};
```

```
use secrecy::SecretString;
use tracing::warn;

#[derive(Default, Debug)]
struct Warnings(Vec<String>);

impl Warnings {
    fn new() -> Self {
        Warnings(Vec::with_capacity(5))
    }

    fn push(&mut self, warning: &str, error: ScenarioError) {
        let formatted = format!("{warning}: {error}");
        warn!("{formatted}");
        self.0.push(formatted);
    }

    fn is_empty(&self) -> bool {
        self.0.is_empty()
    }
}

impl Display for Warnings {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        writeln!(f, "Warnings:")?;
        for warning in &self.0 {
            writeln!(f, "{: >4}- {warning}", "")?;
        }
        Ok(())
    }
}

fn select(
    prompt: &str,
    choices: Vec<String>,
    error_message: &str,
) -> Result<String, anyhow::Error> {
    inquire::Select::new(prompt, choices)
        .prompt()
        .map_err(|error| anyhow!("{error_message}: {error}"))
}

// Prepare the Aurora Scenario. Prompt for several settings that are optional to
 the Scenario, but that the user should choose for the demo.
```

```
// This includes the engine, engine version, and instance class.
async fn prepare_scenario(rds: RdsClient) -> Result<AuroraScenario,
 anyhow::Error> {
    let mut scenario = AuroraScenario::new(rds);

    // Get available engine families for Aurora MySql.
 rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
 'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
    let available_engines = scenario.get_engines().await;
    if let Err(error) = available_engines {
        return Err(anyhow!("Failed to get available engines: {}", error));
    }
    let available_engines = available_engines.unwrap();

    // Select an engine family and create a custom DB cluster parameter group.
 rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
    let engine = select(
        "Select an Aurora engine family",
        available_engines.keys().cloned().collect::<Vec<String>>(),
        "Invalid engine selection",
    )?;

    let version = select(
        format!("Select an Aurora engine version for {engine}").as_str(),
        available_engines.get(&engine).cloned().unwrap_or_default(),
        "Invalid engine version selection",
    )?;

    let set_engine = scenario.set_engine(engine.as_str(),
 version.as_str()).await;
    if let Err(error) = set_engine {
        return Err(anyhow!("Could not set engine: {}", error));
    }

    let instance_classes = scenario.get_instance_classes().await;
    match instance_classes {
        Ok(classes) => {
            let instance_class = select(
                format!("Select an Aurora instance class for {engine}").as_str(),
                classes,
                "Invalid instance class selection",
            )?;
            scenario.set_instance_class(Some(instance_class))
        }
```

```
        Err(err) => return Err(anyhow!("Failed to get instance classes for
 engine: {err}")),
    }

    Ok(scenario)
}

// Prepare the cluster, creating a custom parameter group overriding some group
 parameters based on user input.
async fn prepare_cluster(scenario: &mut AuroraScenario, warnings: &mut Warnings)
 -> Result<(), ()> {
    show_parameters(scenario, warnings).await;

    let offset = prompt_number_or_default(warnings, "auto_increment_offset", 5);
    let increment = prompt_number_or_default(warnings,
 "auto_increment_increment", 3);

    // Modify both the auto_increment_offset and auto_increment_increment
 parameters in one call in the custom parameter group. Set their ParameterValue
 fields to a new allowable value. rds.ModifyDbClusterParameterGroup.
    let update_auto_increment = scenario.update_auto_increment(offset,
 increment).await;

    if let Err(error) = update_auto_increment {
        warnings.push("Failed to update auto increment", error);
        return Err(());
    }

    // Get and display the updated parameters. Specify Source of 'user' to get
 just the modified parameters. rds.DescribeDbClusterParameters(Source='user')
    show_parameters(scenario, warnings).await;

    let username = inquire::Text::new("Username for the database (default
 'testuser')")
        .with_default("testuser")
        .with_initial_value("testuser")
        .prompt();

    if let Err(error) = username {
        warnings.push(
            "Failed to get username, using default",
            ScenarioError::with(format!("Error from inquirer: {error}")),
        );
        return Err(());
```

```
    }
    let username = username.unwrap();

    let password = inquire::Text::new("Password for the database (minimum 8
 characters)")
        .with_validator(|i: &str| {
            if i.len() >= 8 {
                Ok(inquire::validator::Validation::Valid)
            } else {
                Ok(inquire::validator::Validation::Invalid(
                    "Password must be at least 8 characters".into(),
                ))
            }
        })
        .prompt();

    let password: Option<SecretString> = match password {
        Ok(password) => Some(SecretString::from(password)),
        Err(error) => {
            warnings.push(
                "Failed to get password, using none (and not starting a DB)",
                ScenarioError::with(format!("Error from inquirer: {error}")),
            );
            return Err(());
        }
    };

    scenario.set_login(Some(username), password);

    Ok(())
}

// Start a single instance in the cluster,
async fn run_instance(scenario: &mut AuroraScenario) -> Result<(), ScenarioError>
 {
    // Create an Aurora DB cluster database cluster that contains a MySql
 database and uses the parameter group you created.
    // Create a database instance in the cluster.
    // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
 for DBInstanceStatus == 'available'.
    scenario.start_cluster_and_instance().await?;

    let connection_string = scenario.connection_string().await?;
```

```
    println!("Database ready: {connection_string}",);

    let _ = inquire::Text::new("Use the database with the connection string. When
 you're finished, press enter key to continue.").prompt();

    // Create a snapshot of the DB cluster. rds.CreateDbClusterSnapshot.
    // Wait for the snapshot to create. rds.DescribeDbClusterSnapshots until
 Status == 'available'.
    let snapshot_name = inquire::Text::new("Provide a name for the snapshot")
        .prompt()
        .unwrap_or(String::from("ScenarioRun"));
    let snapshot = scenario.snapshot(snapshot_name.as_str()).await?;
    println!(
        "Snapshot is available: {}",
        snapshot.db_cluster_snapshot_arn().unwrap_or("Missing ARN")
    );

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), anyhow::Error> {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::from_env().load().await;
    let client = Client::new(&sdk_config);
    let rds = RdsClient::new(client);
    let mut scenario = prepare_scenario(rds).await?;

    // At this point, the scenario has things in AWS and needs to get cleaned up.
    let mut warnings = Warnings::new();

    if prepare_cluster(&mut scenario, &mut warnings).await.is_ok() {
        println!("Configured database cluster, starting an instance.");
        if let Err(err) = run_instance(&mut scenario).await {
            warnings.push("Problem running instance", err);
        }
    }

    // Clean up the instance, cluster, and parameter group, waiting for the
 instance and cluster to delete before moving on.
    let clean_up = scenario.clean_up().await;
    if let Err(errors) = clean_up {
        for error in errors {
            warnings.push("Problem cleaning up scenario", error);
```

```
        }
    }

    if warnings.is_empty() {
        Ok(())
    } else {
        println!("There were problems running the scenario:");
        println!("{warnings}");
        Err(anyhow!("There were problems running the scenario"))
    }
}

#[derive(Clone)]
struct U8Validator {}
impl StringValidator for U8Validator {
    fn validate(&self, input: &str) -> Result<inquire::validator::Validation,
 CustomUserError> {
        if input.parse::<u8>().is_err() {
            Ok(inquire::validator::Validation::Invalid(
                "Can't parse input as number".into(),
            ))
        } else {
            Ok(inquire::validator::Validation::Valid)
        }
    }
}

async fn show_parameters(scenario: &AuroraScenario, warnings: &mut Warnings) {
    let parameters = scenario.cluster_parameters().await;

    match parameters {
        Ok(parameters) => {
            println!("Current parameters");
            for parameter in parameters {
                println!("\t{parameter}");
            }
        }
        Err(error) => warnings.push("Could not find cluster parameters", error),
    }
}

fn prompt_number_or_default(warnings: &mut Warnings, name: &str, default: u8) ->
 u8 {
    let input = inquire::Text::new(format!("Updated {name}:").as_str())
```

```
        .with_validator(U8Validator {})
        .prompt();

    match input {
        Ok(increment) => match increment.parse::<u8>() {
            Ok(increment) => increment,
            Err(error) => {
                warnings.push(
                    format!("Invalid updated {name} (using {default}
 instead)").as_str(),
                    ScenarioError::with(format!("{error}")),
                );
                default
            }
        },
        Err(error) => {
            warnings.push(
                format!("Invalid updated {name} (using {default}
 instead)").as_str(),
                ScenarioError::with(format!("{error}")),
            );
            default
        }
    }
}
```

テストのオートモッキングを可能にする Amazon RDS サービスのラッパー。

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use aws_sdk_rds::{
    error::SdkError,
    operation::{
        create_db_cluster::{CreateDBClusterError, CreateDbClusterOutput},
        create_db_cluster_parameter_group::CreateDBClusterParameterGroupError,
        create_db_cluster_parameter_group::CreateDbClusterParameterGroupOutput,
        create_db_cluster_snapshot::{CreateDBClusterSnapshotError,
 CreateDbClusterSnapshotOutput},
        create_db_instance::{CreateDBInstanceError, CreateDbInstanceOutput},
        delete_db_cluster::{DeleteDBClusterError, DeleteDbClusterOutput},
        delete_db_cluster_parameter_group::{
```

```
            DeleteDBClusterParameterGroupError,
 DeleteDbClusterParameterGroupOutput,
        },
        delete_db_instance::{DeleteDBInstanceError, DeleteDbInstanceOutput},
        describe_db_cluster_endpoints::{
            DescribeDBClusterEndpointsError, DescribeDbClusterEndpointsOutput,
        },
        describe_db_cluster_parameters::{
            DescribeDBClusterParametersError, DescribeDbClusterParametersOutput,
        },
        describe_db_clusters::{DescribeDBClustersError,
 DescribeDbClustersOutput},
        describe_db_engine_versions::{
            DescribeDBEngineVersionsError, DescribeDbEngineVersionsOutput,
        },
        describe_db_instances::{DescribeDBInstancesError,
 DescribeDbInstancesOutput},

 describe_orderable_db_instance_options::DescribeOrderableDBInstanceOptionsError,
        modify_db_cluster_parameter_group::{
            ModifyDBClusterParameterGroupError,
 ModifyDbClusterParameterGroupOutput,
        },
    },
    types::{OrderableDbInstanceOption, Parameter},
    Client as RdsClient,
};
use secrecy::{ExposeSecret, SecretString};

#[cfg(test)]
use mockall::automock;

#[cfg(test)]
pub use MockRdsImpl as Rds;
#[cfg(not(test))]
pub use RdsImpl as Rds;

pub struct RdsImpl {
    pub inner: RdsClient,
}

#[cfg_attr(test, automock)]
impl RdsImpl {
    pub fn new(inner: RdsClient) -> Self {
```

```
        RdsImpl { inner }
    }

    // snippet-start:[rust.aurora.describe_db_engine_versions.wrapper]
    pub async fn describe_db_engine_versions(
        &self,
        engine: &str,
    ) -> Result<DescribeDbEngineVersionsOutput,
SdkError<DescribeDBEngineVersionsError>> {
        self.inner
            .describe_db_engine_versions()
            .engine(engine)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.describe_db_engine_versions.wrapper]

    // snippet-start:[rust.aurora.describe_orderable_db_instance_options.wrapper]
    pub async fn describe_orderable_db_instance_options(
        &self,
        engine: &str,
        engine_version: &str,
    ) -> Result<Vec<OrderableDbInstanceOption>,
SdkError<DescribeOrderableDBInstanceOptionsError>>
    {
        self.inner
            .describe_orderable_db_instance_options()
            .engine(engine)
            .engine_version(engine_version)
            .into_paginator()
            .items()
            .send()
            .try_collect()
            .await
    }
    // snippet-end:[rust.aurora.describe_orderable_db_instance_options.wrapper]

    // snippet-start:[rust.aurora.create_db_cluster_parameter_group.wrapper]
    pub async fn create_db_cluster_parameter_group(
        &self,
        name: &str,
        description: &str,
        family: &str,
```

```
    ) -> Result<CreateDbClusterParameterGroupOutput,
SdkError<CreateDBClusterParameterGroupError>>
    {
        self.inner
            .create_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .description(description)
            .db_parameter_group_family(family)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.create_db_cluster_parameter_group.wrapper]

    // snippet-start:[rust.aurora.describe_db_clusters.wrapper]
    pub async fn describe_db_clusters(
        &self,
        id: &str,
    ) -> Result<DescribeDbClustersOutput, SdkError<DescribeDBClustersError>> {
        self.inner
            .describe_db_clusters()
            .db_cluster_identifier(id)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.describe_db_clusters.wrapper]

    // snippet-start:[rust.aurora.describe_db_cluster_parameters.wrapper]
    pub async fn describe_db_cluster_parameters(
        &self,
        name: &str,
    ) -> Result<Vec<DescribeDbClusterParametersOutput>,
SdkError<DescribeDBClusterParametersError>>
    {
        self.inner
            .describe_db_cluster_parameters()
            .db_cluster_parameter_group_name(name)
            .into_paginator()
            .send()
            .try_collect()
            .await
    }
    // snippet-end:[rust.aurora.describe_db_cluster_parameters.wrapper]

    // snippet-start:[rust.aurora.modify_db_cluster_parameter_group.wrapper]
```

```
    pub async fn modify_db_cluster_parameter_group(
        &self,
        name: &str,
        parameters: Vec<Parameter>,
    ) -> Result<ModifyDbClusterParameterGroupOutput,
SdkError<ModifyDBClusterParameterGroupError>>
    {
        self.inner
            .modify_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .set_parameters(Some(parameters))
            .send()
            .await
    }
    // snippet-end:[rust.aurora.modify_db_cluster_parameter_group.wrapper]

    // snippet-start:[rust.aurora.create_db_cluster.wrapper]
    pub async fn create_db_cluster(
        &self,
        name: &str,
        parameter_group: &str,
        engine: &str,
        version: &str,
        username: &str,
        password: SecretString,
    ) -> Result<CreateDbClusterOutput, SdkError<CreateDBClusterError>> {
        self.inner
            .create_db_cluster()
            .db_cluster_identifier(name)
            .db_cluster_parameter_group_name(parameter_group)
            .engine(engine)
            .engine_version(version)
            .master_username(username)
            .master_user_password(password.expose_secret())
            .send()
            .await
    }
    // snippet-end:[rust.aurora.create_db_cluster.wrapper]

    // snippet-start:[rust.aurora.create_db_instance.wrapper]
    pub async fn create_db_instance(
        &self,
        cluster_name: &str,
        instance_name: &str,
```

```
        instance_class: &str,
        engine: &str,
    ) -> Result<CreateDbInstanceOutput, SdkError<CreateDBInstanceError>> {
        self.inner
            .create_db_instance()
            .db_cluster_identifier(cluster_name)
            .db_instance_identifier(instance_name)
            .db_instance_class(instance_class)
            .engine(engine)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.create_db_instance.wrapper]

    // snippet-start:[rust.aurora.describe_db_instance.wrapper]
    pub async fn describe_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner
            .describe_db_instances()
            .db_instance_identifier(instance_identifier)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.describe_db_instance.wrapper]

    // snippet-start:[rust.aurora.create_db_cluster_snapshot.wrapper]
    pub async fn snapshot_cluster(
        &self,
        db_cluster_identifier: &str,
        snapshot_name: &str,
    ) -> Result<CreateDbClusterSnapshotOutput,
SdkError<CreateDBClusterSnapshotError>> {
        self.inner
            .create_db_cluster_snapshot()
            .db_cluster_identifier(db_cluster_identifier)
            .db_cluster_snapshot_identifier(snapshot_name)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.create_db_cluster_snapshot.wrapper]

    // snippet-start:[rust.aurora.describe_db_instances.wrapper]
```

```
    pub async fn describe_db_instances(
        &self,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner.describe_db_instances().send().await
    }
    // snippet-end:[rust.aurora.describe_db_instances.wrapper]

    // snippet-start:[rust.aurora.describe_db_cluster_endpoints.wrapper]
    pub async fn describe_db_cluster_endpoints(
        &self,
        cluster_identifier: &str,
    ) -> Result<DescribeDbClusterEndpointsOutput,
SdkError<DescribeDBClusterEndpointsError>> {
        self.inner
            .describe_db_cluster_endpoints()
            .db_cluster_identifier(cluster_identifier)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.describe_db_cluster_endpoints.wrapper]

    // snippet-start:[rust.aurora.delete_db_instance.wrapper]
    pub async fn delete_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DeleteDbInstanceOutput, SdkError<DeleteDBInstanceError>> {
        self.inner
            .delete_db_instance()
            .db_instance_identifier(instance_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.delete_db_instance.wrapper]

    // snippet-start:[rust.aurora.delete_db_cluster.wrapper]
    pub async fn delete_db_cluster(
        &self,
        cluster_identifier: &str,
    ) -> Result<DeleteDbClusterOutput, SdkError<DeleteDBClusterError>> {
        self.inner
            .delete_db_cluster()
            .db_cluster_identifier(cluster_identifier)
            .skip_final_snapshot(true)
```

```
            .send()
            .await
    }
    // snippet-end:[rust.aurora.delete_db_cluster.wrapper]

    // snippet-start:[rust.aurora.delete_db_cluster_parameter_group.wrapper]
    pub async fn delete_db_cluster_parameter_group(
        &self,
        name: &str,
    ) -> Result<DeleteDbClusterParameterGroupOutput,
 SdkError<DeleteDBClusterParameterGroupError>>
    {
        self.inner
            .delete_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.delete_db_cluster_parameter_group.wrapper]
}
```

このシナリオで使用した依存関係を含む Cargo.toml。

```
[package]
name = "aurora-code-examples"
authors = [
  "David Souther <dpsouth@amazon.com>",
]
edition = "2021"
version = "0.1.0"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/
reference/manifest.html

[dependencies]
anyhow = "1.0.75"
assert_matches = "1.5.0"
aws-config = { version = "1.0.1", features = ["behavior-version-latest"] }
aws-smithy-types = { version = "1.0.1" }
aws-smithy-runtime-api = { version = "1.0.1" }
aws-sdk-rds = { version = "1.3.0" }
inquire = "0.6.2"
```

```
mockall = "0.11.4"
phf = { version = "0.11.2", features = ["std", "macros"] }
sdk-examples-test-utils = { path = "../../test-utils" }
secrecy = "0.8.0"
tokio = { version = "1.20.1", features = ["full", "test-util"] }
tracing = "0.1.37"
tracing-subscriber = { version = "0.3.15", features = ["env-filter"] }
```

- API の詳細については、「AWS SDK for Rust API リファレンス」の以下のトピックを参照してください。
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeOrderableDBInstanceOptions
  - ModifyDBClusterParameterGroup

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

AWS SDK を使用した Aurora のクロスサービスの例

次のサンプルアプリケーションでは、AWS SDK を使用して Aurora を他の AWS のサービス と組み合わせます。それぞれの例には、GitHub へのリンクがあり、アプリケーションを設定および実行する方法についての説明を参照できます。

例

- 貸出ライブラリ REST API を作成する
- Aurora Serverless 作業項目トラッカーの作成

貸出ライブラリ REST API を作成する

以下のコード例は、Amazon Aurora データベースでバックアップされた REST API を使用して、常連客が書籍の貸出と返却できる貸出ライブラリを作成する方法を示しています。

Python

SDK for Python (Boto3)

AWS SDK for Python (Boto3) を Amazon Relational Database Service (Amazon RDS) API および AWS Chalice で使用して、Amazon Aurora データベースに基づく REST API を作成する方法を示します。Web サービスは完全にサーバーレスであり、常連客が本を借りたり返却したりできるシンプルな貸し出しライブラリを表しています。以下ではその方法を説明しています。

- サーバーレス Aurora データベースクラスターを作成および管理します。
- AWS Secrets Manager を使用してデータベース認証情報を管理します。
- Amazon RDS を使用してデータをデータベースに出し入れするデータストレージレイヤーを実装します。
- AWS Chalice を使用して、サーバーレス REST API を Amazon API Gateway および AWS Lambda にデプロイします。
- リクエストパッケージを使用して、Web サービスにリクエストを送信します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- API Gateway

- Aurora
- Lambda
- Secrets Manager

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Aurora Serverless 作業項目トラッカーの作成

次のコード例は、Amazon Aurora Serverless データベースの作業項目を追跡し、Amazon Simple Email Service (Amazon SES) を使用してレポートを送信するウェブアプリケーションを作成する方法を示しています。

.NET

AWS SDK for .NET

AWS SDK for .NET を使用して Amazon Aurora データベースの作業項目を追跡し、Amazon Simple Email Service (Amazon SES) を使用してレポートを E メールで送信するウェブアプリケーションを作成する方法を示します。この例では、React.js で構築されたフロントエンドを使用して RESTful .NET バックエンドと対話します。

- React ウェブアプリケーションを AWS のサービスと統合します。
- Aurora テーブルの項目を一覧表示、更新、削除します。
- Amazon SES を使用して、フィルター処理された作業項目の E メールレポートを送信します。
- 付属の AWS CloudFormation スクリプトでサンプルリソースをデプロイおよび管理します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Aurora
- Amazon RDS
- Amazon RDS データサービス

- Amazon SES

C++

SDK for C++

Amazon Aurora Serverless データベースに保存されている作業項目を追跡して報告するウェブアプリケーションを作成する方法を説明します。

Amazon Aurora Serverless データをクエリし、React アプリケーションで使用するための C++ REST API の完全なソースコードと設定方法については、GitHub にある完全な例を参照してください。

この例で使用されているサービス

- Aurora
- Amazon RDS
- Amazon RDS データサービス
- Amazon SES

Java

SDK for Java 2.x

Amazon RDS データベースに保存されている作業項目を追跡およびレポートするウェブアプリケーションを作成する方法を説明します。

Amazon Aurora サーバーレスデータをクエリする Spring REST API と React アプリケーションで使用するための完全なソースコードと設定方法については、GitHub にある完全な例を参照してください。

完全なソースコードと JDBC API を使用する例のセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Aurora
- Amazon RDS
- Amazon RDS データサービス

- Amazon SES

JavaScript

SDK for JavaScript (v3)

AWS SDK for JavaScript (v3) を使用して Amazon Aurora データベースの作業項目を追跡し、Amazon Simple Email Service (Amazon SES) を使用してレポートを E メールで送信するウェブアプリケーションを作成する方法を示します。この例では、React.js で構築されたフロントエンドを使用して Express Node.js バックエンドと対話します。

- React.js ウェブアプリケーションを AWS のサービス と統合します。
- Aurora テーブルの項目を一覧表示、追加、更新します。
- Amazon SES を使用して、フィルター処理された作業項目の E メールレポートを送信します。
- 付属の AWS CloudFormation スクリプトでサンプルリソースをデプロイおよび管理します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Aurora
- Amazon RDS
- Amazon RDS データサービス
- Amazon SES

Kotlin

SDK for Kotlin

Amazon RDS データベースに保存されている作業項目を追跡およびレポートするウェブアプリケーションを作成する方法を説明します。

Amazon Aurora サーバーレスデータをクエリする Spring REST API と React アプリケーションで使用するための完全なソースコードと設定方法については、GitHub にある完全な例を参照してください。

この例で使用されているサービス

- Aurora
- Amazon RDS
- Amazon RDS データサービス
- Amazon SES

PHP

SDK for PHP

AWS SDK for PHP を使用して Amazon RDS データベースの作業項目を追跡し、Amazon Simple Email Service (Amazon SES) を使用してレポートを E メールで送信するウェブアプリケーションを作成する方法を示します。この例では、React.js で構築されたフロントエンドを使用して RESTful PHP バックエンドと対話します。

- React.js ウェブアプリケーションを AWS のサービスと統合します。
- Amazon RDS テーブル内の項目の一覧表示、追加、更新、削除を行います。
- Amazon SES を使用して、フィルター処理された作業項目の E メールレポートを送信します。
- 付属の AWS CloudFormation スクリプトでサンプルリソースをデプロイおよび管理します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Aurora
- Amazon RDS
- Amazon RDS データサービス
- Amazon SES

Python

SDK for Python (Boto3)

AWS SDK for Python (Boto3) を使用して Amazon Aurora Serverless データベースの作業項目を追跡し、Amazon Simple Email Service (Amazon SES) を使用してレポートを E メールで送

信する REST サービスを作成する方法を示します。この例では、Flask ウェブフレームワークを使用して HTTP ルーティングを処理し、React ウェブページと統合して完全に機能するウェブアプリケーションを提供します。

- AWS のサービス と統合する Flask REST サービスを構築します。
- Aurora Serverless データベースに保存されている作業項目の読み取り、書き込み、更新を行います。
- データベース認証情報を含む AWS Secrets Manager シークレットを作成し、それを使用してデータベースへの呼び出しを認証します。
- Amazon SES を使用して作業項目のレポートを E メールで送信します。

完全なソースコードとセットアップおよび実行の手順については、GitHub で完全な例を参照してください。

この例で使用されているサービス

- Aurora
- Amazon RDS
- Amazon RDS データサービス
- Amazon SES

AWS SDK デベロッパーガイドとコード例の完全なリストについては、「このサービスを AWS SDK で使用する」を参照してください。このトピックには、使用開始方法に関する情報と、以前の SDK バージョンの詳細も含まれています。

Amazon Aurora を使用する際のベストプラクティス

以下に、データを Amazon Aurora DB クラスターに使用または移行するための一般的なベストプラクティスとオプションに関する情報を示します。

Amazon Aurora のベストプラクティスのいくつかは、特定のデータベースエンジンに固有です。データベースエンジンに固有の Aurora のベストプラクティスの詳細については、以下を参照してください。

データベースエンジン	ベストプラクティス
Amazon Aurora MySQL	「Amazon Aurora MySQL を使用する際のベストプラクティス」を参照してください。
Amazon Aurora PostgreSQL	「Amazon Aurora PostgreSQL を使用する際のベストプラクティス」を参照してください。

> **Note**
>
> Aurora の一般的な推奨事項については、Amazon Aurora の推奨事項の表示とこれらに対する対応 を参照してください。

トピック

- Amazon Aurora の基本的な操作のガイドライン
- DB インスタンスの RAM の推奨事項
- AWS データベースドライバー
- Amazon Aurora のモニタリング
- DB パラメータグループおよび DB クラスターパラメータグループを使用する
- Amazon Aurora のベストプラクティスビデオ

Amazon Aurora の基本的な操作のガイドライン

以下に示しているのは、基本的な操作のガイドラインであり、Amazon Aurora の使用時にすべてのユーザーが従う必要があります。Amazon RDS のサービスレベルアグリーメント (SLA) では、以下のガイドラインに従う必要があります。

- メモリ、CPU、ストレージの使用状況をモニタリングする。Amazon CloudWatch は、使用パターンが変更されたり、デプロイメントの最大容量に近づいたりすると、通知するように設定できます。このようにして、システムのパフォーマンスと可用性を維持できます。
- クライアントアプリケーションが DB インスタンスのドメインネームサービス (DNS) のデータをキャッシュしている場合には、有効期限 (TTL) の値を 30 秒未満に設定します。DB インスタンスの基になる IP アドレスは、フェイルオーバー後に変更されている可能性があります。したがって、長時間キャッシュされている DNS データがあると、使用されなくなった IP アドレスにアプリケーションが接続しようとして、接続に失敗することがあります。複数のリードレプリカがある Aurora DB クラスターでは、接続でリーダーエンドポイントを使用していて、いずれかのリードレプリカインスタンスがメンテナンスや削除された場合にも、接続エラーが発生する場合があります。
- DB クラスターのフェイルオーバーをテストすることで、そのプロセスでユースケースにかかる時間を把握します。フェイルオーバーをテストすることで、DB クラスターにアクセスするアプリケーションがフェイルオーバー後に新しい DB クラスターに自動的に接続されることを確認できます。

DB インスタンスの RAM の推奨事項

パフォーマンスを最適化するために、作業セットをほぼすべてメモリに保持できるように十分な RAM を割り当てます。作業セットがほぼすべてメモリ内にあるかどうかを確認するには、Amazon CloudWatch の以下のメトリクスを調べます。

- `VolumeReadIOPS` – クラスターボリュームからの読み取り I/O オペレーションの平均回数を測定し、5 分間隔で報告します。`VolumeReadIOPS` の値は小さく、安定している必要があります。場合によっては、読み取り I/O が急に増えたり、通常よりも増えたりすることがあります。その場合は、DB クラスターの DB インスタンスを調査して、I/O 増加の原因となっている DB インスタンスを特定してください。

> **Tip**
>
> Aurora MySQL クラスターが並列クエリを使用している場合、VolumeReadIOPS 値が増加することがあります。パラレルクエリでは、バッファプールは使用されません。したがって、クエリは高速ですが、この最適化された処理により、読み取り操作とそれに関連する料金が増加する可能性があります。

- BufferCacheHitRatio – このメトリクスは、DB クラスター内の DB インスタンスのバッファキャッシュによって処理されるリクエストの割合を測定します。このメトリクスにより、メモリから提供されているデータの量についての洞察を得られます。

  ヒット率が高い場合、DB インスタンスで十分なメモリが使用可能であることを示します。ヒット率が低い場合、この DB インスタンスのクエリが、頻繁にディスクに書き込まれていることを示します。ワークロードを調査して、この動作の原因となっているクエリを特定します。

ワークロードを調査した後、より多くのメモリが必要であることがわかった場合は、DB インスタンスクラスをより多くの RAM を備えたクラスにスケールアップすることを検討してください。その後、上記のメトリクスを調査し、必要に応じてスケールアップを続けることができます。Aurora クラスターが 40 TB より大きい場合は、db.t2、db.t3、または db.t4g インスタンスクラスを使用しないでください。

詳細については、「Amazon Aurora の Amazon CloudWatch メトリクス」を参照してください。

AWS データベースドライバー

アプリケーション接続には、AWS のドライバースイートを推奨します。ドライバーは、スイッチオーバーとフェイルオーバーの時間の短縮のほか、AWS Secrets Manager、AWS Identity and Access Management (IAM)、フェデレーティッド ID での認証をサポートするように設計されています。AWS ドライバーは、DB クラスターステータスをモニタリングし、クラストポロジを認識して新しいライターを決定することを前提としています。このアプローチにより、スイッチオーバーとフェイルオーバーの時間が 1 桁秒に短縮されます (オープンソースドライバーの場合は数十秒)。

新しいサービス機能が導入されるにあたって、こうしたサービス機能を標準でサポートすることが AWS のドライバースイートの目標です。

詳細については、「AWS ドライバーを使用した Aurora DB クラスターへの接続」を参照してください。

Amazon Aurora のモニタリング

Amazon Aurora には、さまざまなメトリクスとインサイトが用意されており、Aurora DB クラスターの状態とパフォーマンスをモニタリングして判断することができます。Aurora メトリクスは、AWS Management Console、AWS CLI、CloudWatch API などのさまざまなツールを使用して表示できます。Performance Insights と CloudWatch メトリクスの組み合わせを Performance Insights ダッシュボードで表示し、DB インスタンスをモニタリングできます。このモニタリングビューを使用するには、DB インスタンスの Performance Insights がオンになっている必要があります。このモニタリングビューの詳細については、「Amazon RDS コンソールでの組み合わせたメトリクスの表示」を参照してください。

特定の期間のパフォーマンス分析レポートを作成して、特定されたインサイトと問題を解決するための推奨事項を確認できます。詳細については、「パフォーマンス分析レポートの作成」を参照してください。

DB パラメータグループおよび DB クラスターパラメータグループを使用する

パラメータグループの変更を本稼働 DB クラスターに適用する前に、DB パラメータグループおよび DB クラスターパラメータグループの変更を、テスト DB クラスターで試すことをお勧めします。不適切な設定の DB エンジンパラメータがあると、パフォーマンスの低下やシステムの不安定化など、予期しない悪影響が生じることがあります。

DB エンジンパラメータの変更時には常に注意が必要です。DB パラメータグループを変更する前に必ず DB クラスターをバックアップしてください。DB クラスターのバックアップの詳細については、「Amazon Aurora DB クラスターのバックアップと復元」を参照してください。

Amazon Aurora のベストプラクティスビデオ

AWS Online Tech Talks チャネルでは、セキュリティと可用性のより高い Amazon Aurora DB クラスターを作成および設定するためのベストプラクティスに関するプレゼンテーションが行われました。「Amazon Aurora の高可用性のベストプラクティス」を参照してください。

Amazon Aurora の PoC (概念実証) の実行

Aurora の PoC (概念実証) を設定して実行する方法について以下に説明します。PoC (概念実証) は、Aurora がアプリケーションに適合しているかどうかを判断するために実施する調査です。PoC (概念実証) は、独自のデータベースアプリケーションのコンテキストにおける Aurora の機能と、現在のデータベース環境に対する Aurora の適合性を理解するのに役立ちます。また、データの移動、SQL コードの移植、パフォーマンスのチューニングを行い、現在の管理手順を適応させるために必要となる労力の目安を提供します。

このトピックでは、PoC (概念実証) を実行するための大まかなステップと意思決定について、ステップバイステップの概要を以下に提供します。詳細な手順については、リンクに従って主題別の詳細なドキュメントを参照してください。

Aurora の PoC (概念実証) の概要

Amazon Aurora の PoC (概念実証) を実行すると、既存のデータや SQL アプリケーションを Aurora に移植するために何が必要であるかを判断できます。本番環境に特徴的なデータ量やアクティビティを使用して Aurora の重要な機能を広範に試します。この目的は、以前のデータベースインフラストラクチャでは対応できなくなった課題に対して、Aurora の強力な機能で対応できることを確認することにあります。PoC (概念実証) を完了すると、より大規模なパフォーマンスのベンチマークとアプリケーションのテストを行うための堅実な計画を立てることができます。この時点で、本番稼働用デプロイに必要な最大の作業項目が明らかになります。

以下のベストプラクティスに関するアドバイスを参考にして、ベンチマーク時の問題の原因となる一般的な間違いを避けることができます。ただし、このトピックでは、ベンチマークやパフォーマンスチューニングのステップバイステップのステップは提供しません。これらの手順は、ワークロードや実際に使用する Aurora 機能に応じて異なります。詳細については、「Aurora DB クラスターのパフォーマンスとスケーリングの管理」、「Amazon Aurora MySQL パフォーマンスの拡張」、「Amazon Aurora PostgreSQL の管理」、「Amazon Aurora での Performance Insights を使用した DB 負荷のモニタリング」などのパフォーマンス関連のドキュメントを参照してください。

このトピックの説明は、貴社がコードを記述してスキーマを設計するアプリケーション、MySQL や PostgreSQL のオープンソースデータベースエンジンをサポートするアプリケーションに主に該当します。商用アプリケーションや、アプリケーションフレームワークで生成されたコードをテストする場合、貴社が柔軟に対応できず、一部のガイドラインを適用できないことがあります。このような場

合は、アプリケーションのタイプに応じた Aurora のベストプラクティスや導入事例があるかどうかを AWS の担当者にお問い合わせください。

1. 目標を明確にする

PoC (概念実証) の一環として Aurora を評価する場合は、実施する測定の種類と成功の評価方法を選択します。

アプリケーションのすべての機能が Aurora と互換性があることを確認する必要があります。Aurora メジャーバージョンは、対応する MySQL および PostgreSQL のメジャーバージョンとワイヤ互換性があるため、これらのエンジン用に開発されたほとんどのアプリケーションは、Auroraとも互換性があります。ただし、その場合でも、アプリケーション別に互換性を検証する必要があります。

例えば、Aurora クラスターの設定時に選択するオプションのいくつかは、特定のデータベース機能を使用できるかどうか、使用すべきかどうかに影響します。まず、最も汎用的な Aurora クラスターであるプロビジョンドクラスターからスタートできます。次に、サーバーレスやパラレルクエリなどの特化された設定が、ワークロードに利点をもたらすかどうかを判断できます。

目標を明確にして定量化するには、以下の問いが役立ちます。

- Aurora はワークロードのすべての機能ユースケースをサポートするか?
- 必要なデータセットサイズやロードレベルは何であるか? このレベルまでスケールできるか?
- クエリスループットやレイテンシーの具体的な要件は何であるか? これらの要件を達成できるか?
- ワークロードの予定されたダウンタイムまたは予定外のダウンタイムの最小許容時間はどれくらいであるか? これを達成できるか?
- 業務効率に関して必要なメトリクスは何であるか? これらのメトリクスを正確にモニタリングできるか?
- Aurora は特定のビジネス目標 (コストの削減、デプロイの拡張、プロビジョニングのスピードなど) をサポートするか? これらの目標を定量化する方法があるか?
- ワークロードに関するすべてのセキュリティおよびコンプライアンス要件を満たすことができるか?

少し時間をかけて Aurora データベースエンジンとプラットフォームの機能に関する知識を深め、サービスドキュメントを確認します。必要な結果を達成するために役立つすべての機能に注目します。これらの機能の 1 つとしてワークロードの統合が挙げられます。詳細については、AWS Database Blog 記事「ワークロードの統合に向けた Amazon Aurora と MySQL の互換性の計画と最

適化の方法」を参照してください。別の機能としては、需要に基づくスケーリングが挙げられます。詳細については、Amazon Aurora ユーザーガイド の「Aurora レプリカでの Amazon Aurora Auto Scaling の使用」を参照してください。その他の機能として、パフォーマンスの向上やデータベース操作の簡素化などが挙げられます。

2. ワークロードの特性を理解する

意図したユースケースに則って Aurora を評価します。Aurora は、オンライントランザクション処理 (OLTP) ワークロードに適しています。別個のデータウェアハウスクラスターをプロビジョンせずに、リアルタイムの OLTP データを保持するクラスターでレポートを実行することもできます。ユースケースが上記のカテゴリに該当するかどうかは、以下の特性をチェックすることで判断できます。

- 数十、数百、または数千の同時クライアントを伴う高い同時実行。
- 大量の低レイテンシークエリ (ミリ秒単位 ~ 秒単位)。
- 短い、リアルタイムのトランザクション。
- 高度に選択的なクエリパターン (インデックスベースのルックアップを使用)。
- HTAP の場合、Aurora のパラレルクエリを利用する分析クエリ。

データベースの選択に影響する主な要因の 1 つは、データの速度です。高速では、データが非常に頻繁に挿入されて更新されます。このようなシステムでは、数千の接続や数十万の同時クエリがデータベースに対して読み書きされる場合があります。高速システムのクエリは、通常、比較的少数の行に影響します。また、通常、同じ行の複数の列にアクセスします。

Aurora は、高速データを処理するように設計されています。ワークロードに応じて異なりますが、単一の r4.16xlarge DB インスタンスを持つ Aurora クラスターの場合、1 秒あたり 600,000 個を超える SELECT ステートメントを処理できます。ワークロードに応じて、このようなクラスターは、INSERT、UPDATE、および DELETE ステートメントを、1 秒あたり 200,000 回処理できます Aurora は行指向のストアデータベースであり、高ボリューム、高スループット、および高度にパラレル化された OLTP ワークロードに最適です。

また、Aurora は、OLTP ワークロードを処理する同じクラスターで、レポートクエリを実行することもできます。Aurora は最大 15 個のレプリカをサポートします。各レプリカは平均して、プライマリインスタンスの 10 ~ 20ミリ秒以内を消費します。アナリストは、データを別個のデータウェアハウスクラスターにコピーせずに、リアルタイムで OLTP データにクエリを実行できます。パラレルクエリ機能を使用する Aurora クラスターでは、処理、フィルター処理、および集約作業の多くを、広く分散された Aurora ストレージシステムにオフロードできます。

この計画フェーズで、Aurora の機能、他の AWS のサービス、AWS Management Console、および AWS CLI に関する知識を深めてください。また、これらが、PoC (概念実証) で使用予定の他のツールとどのように連携するかも確認します。

3. AWS Management Console または AWS CLI を試す

次のステップとして、AWS Management Console または AWS CLI の実践を通じ、これらのツールや Aurora に習熟します。

AWS Management Console を試す

Aurora データベースクラスターに関する以下の初期アクティビティでは、主と s h ちえ、AWS Management Console 環境に慣れながら Aurora クラスターの設定と変更を試すことができます。MySQL 互換および PostgreSQL 互換のデータベースエンジンを Amazon RDS で使用している場合は、Aurora を使用する際にその知識を活かすことができます。

Aurora の共有ストレージモデルや機能 (レプリケーションやスナップショットなど) を利用することで、データベースクラスター全体を一種の自由に操作可能なオブジェクトとして扱うことができます。Aurora クラスターの容量は、PoC (概念実証) 中に頻繁に設定、削除、変更できます。容量、データベース設定、物理データレイアウトに関する初期の選択に縛られることはありません。

使用をスタートするには、空の Aurora クラスターを設定します。初期に試すプロビジョンド容量タイプとリージョンの場所を選択します。

SQL コマンドラインアプリケーションなどのクライアントプログラムを使用して、このクラスターに接続します。初期は、クラスターエンドポイントを使用して接続します。このエンドポイントに接続して、データ定義言語 (DDL) ステートメントや、抽出、変換、読み取り (ETL) プロセスなどの書き込みオペレーションを実行します。PoC (概念実証) の後半では、リーダーエンドポイントを使用してクエリが集中するセッションを接続します。これにより、クエリのワークロードがクラスター内の複数の DB インスタンスに分散されます。

Aurora レプリカの数を増やしてクラスターをスケールアウトします。詳しい手順については、「Amazon Aurora でのレプリケーション」を参照してください。AWS インスタンスクラスを変更することで DB インスタンスをスケールアップ/ダウンします。Aurora では、この種のオペレーションが簡素化されているため、システム容量に関する初期の見積りが不正確でも、すべてをやり直さずに後で調整できます。

スナップショットを作成して別のクラスターに復元します。

クラスターのメトリクスを調べて、時間の経過に伴うアクティビティと、メトリクスがクラスター内の DB インスタンスにどのように適用されるかを確認します。

初期の段階で以上のオペレーションを AWS Management Console で実行する方法に慣れることが役立ちます。Aurora で実行できるオペレーションを理解したら、これらのオペレーションを AWS CLI で自動化する手法に進みます。以下のセクションでは、PoC (概念実証) 時における以上のアクティビティの手順とベストプラクティスについて詳しく説明します。

AWS CLI を試す

デプロイと管理の手順は、PoC (概念実証) の設定段階においても自動化することをお勧めします。これを行うには、AWS CLI の使い方に慣れます (まだ慣れていない場合)。MySQL 互換および PostgreSQL 互換のデータベースエンジンを Amazon RDS で使用している場合は、Aurora を使用する際にその知識を活かすことができます。

通常、Aurora では、DB インスタンスのグループがクラスターとして編成されます。したがって、多くのオペレーションでは、どの DB インスタンスがクラスターに関連付けられているかを判断し、すべてのインスタンスに対して管理オペレーションをループで実行します。

例えば、Aurora クラスターの作成などのステップを自動化し、スケールアップしてインスタンスのサイズを大きくしたり、スケールアウトして DB インスタンス数を増やしたりできます。これにより、PoC (概念実証) のステージを繰り返し、Aurora クラスターの異なる種類や異なる設定で what-if シナリオを試すことができます。

インフラストラクチャのデプロイツール (AWS CloudFormation など) の機能と制限を確認します。PoC (概念実証) のコンテキストで実行するアクティビティが、本番稼働用として適していないことが判明する場合があります。例えば、AWS CloudFormation による変更動作では、新しいインスタンスを作成して現在のもの (データを含む) を削除します。この動作の詳細については、AWS CloudFormation ユーザーガイドの「スタックのリソースの更新動作」を参照してください。

4. Aurora クラスターを作成する

Aurora では、what-if シナリオを試すために、DB インスタンスをクラスターに追加し、これらの DB インスタンスをより強力なインスタンスクラスにスケールアップできます。また、構成設定が異なる複数のクラスターを作成して、同じワークロードをパラレルして実行することもできます。Aurora では、DB クラスターを非常に柔軟に設定、削除、および再設定できます。このため、これらの手法を PoC (概念実証) プロセスの初期ステージで身に付けることが役立ちます。Aurora クラスターの一般的な作成手順については、「Amazon Aurora DB クラスターの作成」を参照してください。

可能であれば、以下の設定を使用してクラスターをスタートします。このステップをスキップするのは、特定のユースケースを決めている場合に限ります。例えば、特化された種類の Aurora クラスターがユースケースで必要な場合は、このステップをスキップできます。また、特定のデータベースエンジンとバージョンの組み合わせが必要な場合にも、スキップできます。

- [Easy create (簡易作成)] を無効にします。PoC (概念実証) では、同じクラスターやわずかに異なるクラスターを後で作成できるように、すべての選択した設定を把握しておくことをお勧めします。
- 最新の DB エンジンバージョンを使用します。データベースエンジンとバージョンのこれらの組み合わせは、Aurora の他の機能との互換性が高く、本番アプリケーションとしてカスタマーに広く使用されています。
  - Aurora MySQL バージョン 3.x (MySQL 8.0 互換)
  - Aurora PostgreSQL バージョン 15.x または 16.x
- [開発/テスト] テンプレートを選択します。このオプションは、PoC (概念実証) のアクティビティでは重要ではありません。
- [DB instance class (DB インスタンスクラス)] で、[Memory optimized classes (メモリ最適化クラス)] を選択し、[xlarge] インスタンスクラスのいずれかを選択します。インスタンスクラスは、スケールアップ/ダウンできます。
- [Multi-AZ Deployment (マルチ AZ 配置)] で、[Create an Aurora Replica or リーダー node in a different AZ (別の AZ に Aurora レプリカまたはリーダーノードを作成)] を選択します。Aurora の最も役立つ機能の多くでは、複数の DB インスタンスを持つクラスターを使用します。すべての新しいクラスターは、最低 2 つの DB インスタンスで常にスタートするのが合理的です。2 つ目の DB インスタンスに別のアベイラビリティーゾーンを使用することで、複数の異なる高可用性シナリオをテストできます。
- DB インスタンスに名前を付ける場合は、一般的な命名規則に従います。クラスターの DB インスタンスを「ライター」として参照しないようにします。これらのロールは、必要に応じて、複数の異なる DB インスタンスが引き受ける場合があるためです。`clustername-az-serialnumber` (`myprodappdb-a-01` など) を使用することをお勧めします。これらの名前は、DB インスタンスとその配置を一意に識別します。
- Aurora クラスターのバックアップの保存期間を [高] に設定します。保存期間を長くすると、最大 35 日間にわたってポイントインタイムリカバリ (PITR) を実行できます。DDL やデータ操作言語 (DML) のステートメントを伴うテストを実行したら、データベースを既知の状態にリセットできます。データを間違えて削除または変更した場合は、復旧することもできます。
- クラスターの作成時に、追加の復旧、ログ記録、モニタリング機能を有効にします。[バックトラック]、[Performance Insights]、[モニタリング]、および [ログのエクスポート] のすべてのオプ

ションをオンにします。これらの機能を有効にすると、ワークロードのバックトラック、拡張モニタリング、Performance Insights などの機能の適合性をテストできます。PoC (概念実証) 中に、簡単にパフォーマンスを調査してトラブルシューティングを実行することもできます。

5. スキーマを設定する

Aurora クラスターで、アプリケーション用のデータベース、テーブル、インデックス、外部キーなどのスキーマオブジェクトを設定します。別の MySQL 互換または PostgreSQL 互換のデータベースシステムから移行する場合、これは簡単明快なステージとなります。使い慣れている同じ SQL 構文やコマンドラインまたは他のクライアントアプリケーションをデータベースエンジンで使用します。

クラスターで SQL ステートメントを発行するには、そのクラスターエンドポイントを見つけて、その値を接続パラメータとしてクライアントアプリケーションに渡します。クラスターエンドポイントは、クラスターの詳細ページの [Connectivity (接続)] タブで見つけることができます。クラスターエンドポイントには、[書き込み] というラベルが付いています。[リーダー] というラベルが付いた他のエンドポイントは、読み取り専用接続であり、レポートや他の読み取り専用クエリを実行するエンドユーザーに渡すことができます。クラスターへの接続問題のヘルプについては、「Amazon Aurora DB クラスターへの接続」を参照してください。

スキーマやデータを別のデータベースシステムから移植する場合は、この時点でスキーマの変更を行う必要があります。これらのスキーマの変更は、Aurora の SQL 構文や機能と一致させる必要があります。この時点で、特定の列、制約、トリガーなどのスキーマオブジェクトを除外できます。これは、Aurora との互換性のために、これらのオブジェクトの手直しが必要な場合に特に役立つ場合があります。ただし、PoC (概念実証) の目的には重要ではありません。

基盤となるエンジンが Aurora のものとは異なるデータベースシステムから移行する場合は、AWS Schema Conversion Tool (AWS SCT) を使用してプロセスを簡素化することを検討してください。詳細については、AWS Schema Conversion Tool ユーザーガイドを参照してください。移行および移植アクティビティの一般的な詳細については、AWS ホワイトペーパー「Amazon Aurora へのデータベースの移行」を参照してください。

このステージでは、スキーマ設定に非効率が存在するかどうかを評価できます。例えば、インデックス作成戦略や他のテーブル構造 (パーティショニングされたテーブルなど) の非効率を評価します。このような非効率は、アプリケーションのデプロイ先のクラスターに複数の DB インスタンスや重いワークロードがあると、増幅される場合があります。このようなパフォーマンス面の微調整を今すぐ行うか、後でフルベンチマークテストなどのアクティビティ時に行うかを判断します。

6. データをインポートする

PoC (概念実証) では、以前のデータベースシステムからデータ全体または代表的なサンプルをインポートします。可能であれば、テーブルごとに少なくとも一部のデータを設定します。これにより、すべてのデータ型とスキーマ機能の互換性をテストできます。基本的な Aurora 機能を実行したら、データ量をスケールアップします。PoC (概念実証) が完了するまでに、正確な結論を引き出せるだけのサイズのデータセットを使用して、ETL ツール、クエリ、およびワークロード全体をテストする必要があります。

物理または論理的なバックアップデータを Aurora にインポートするには、いくつかの手法を利用できます。詳細については、PoC (概念実証) で使用するデータベースエンジンに応じて、「Amazon Aurora MySQL DB クラスターへのデータの移行」または「PostgreSQL と互換性がある Amazon Aurora にデータを移行する」を参照してください。

検討している ETL ツールやテクノロジーを試します。どれがニーズに最適であるかを判断します。スループットと柔軟性の両方を検討します。例えば、ETL ツールによっては 1 回の転送を実行するものと、古いシステムから Aurora への継続的なレプリケーションを実行するものがあります。

MySQL 互換システムから Aurora MySQL に移行する場合は、ネイティブのデータ転送ツールを使用できます。PostgreSQL 互換システムから Aurora PostgreSQL に移行する場合も同様です。基盤となるエンジンが Aurora のものとは異なるデータベースシステムから移行する場合は、AWS Database Migration Service (AWS DMS) を試すことができます。AWS DMS の詳細については、AWS Database Migration Service ユーザーガイドを参照してください。

移行および移植アクティビティの詳細については、AWS ホワイトペーパー「Aurora 移行ハンドブック」を参照してください。

7. SQL コードを移植する

SQL および関連アプリケーションを試すには、ケース別に異なる労力レベルが必要になります。特に、MySQL 互換/PostgreSQL 互換システムまたは別の種類から移行する場合は、労力レベルが異なります。

- RDS for MySQL や RDS for PostgreSQL から移行する場合、SQL の変更は小規模であり、Aurora で元の SQL コードを試して必要な変更を手動で反映できます。
- 同様に、MySQL や PostgreSQL と互換性があるオンプレミスデータベースから移行する場合は、元の SQL コードを試して手動で変更を反映できます。

- 別の商用データベースから移行する場合、SQL に必要な変更はより広範囲に及びます。このような場合は、AWS SCT の使用を検討します。

このステージでは、スキーマ設定に非効率が存在するかどうかを評価できます。例えば、インデックス作成戦略や他のテーブル構造 (パーティショニングされたテーブルなど) の非効率を評価します。このようなパフォーマンス面の微調整を今すぐ行うか、後でフルベンチマークテストなどのアクティビティ時に行うかを判断します。

アプリケーションでデータベース接続ロジックを検証できます。Aurora の分散処理を利用するには、必要に応じて、読み取りオペレーションと書き込みオペレーションに別個の接続を使用します。また、クエリオペレーションには比較的短いセッションを使用します。接続の詳細については、「9. Aurora に接続する」を参照してください

プロダクションデータベースの問題を回避するために妥協やトレードオフが必要であったかどうかを考えます。PoC (概念実証) のスケジュール内に、スキーマの設計とクエリを改善するための時間を組み入れます。パフォーマンス、運用コスト、スケーラビリティを簡単に改善できるかどうかを判断するには、元のアプリケーションと変更後のアプリケーションを異なる Aurora クラスターでパラレル実行します。

移行および移植アクティビティの詳細については、AWS ホワイトペーパー「Aurora 移行ハンドブック」を参照してください。

8. 構成設定を指定する

Aurora の PoC (概念実証) の一環として、データベースの設定パラメータを見直すこともできます。MySQL や PostgreSQL の構成設定は、現在の環境のパフォーマンスやスケーラビリティに応じて、すでにチューニングされている場合があります。Aurora のストレージサブシステムは、高速のストレージサブシステムを備えた分散型のクラウドベース環境に適応するようにチューニングされています。そのため、多くの以前のデータベースエンジン設定は該当しません。初期の実験は Aurora のデフォルトの構成設定で実施することをお勧めします。現在の環境の設定を再適用するのは、パフォーマンスやスケーラビリティのボトルネックが生じた場合のみとします。この主題に関心がある場合は、AWS データベースブログの記事「Aurora ストレージエンジンの紹介」で詳細を参照してください。

Aurora では、特定のアプリケーションやユースケースの最適な構成設定を簡単に再利用できます。DB インスタンスごとに別個の設定ファイルを編集せずに、パラメータセットを管理し、クラスター全体や DB インスタンス別に割り当てます。例えば、タイムゾーンの設定はクラスター内のすべ

ての DB インスタンスに適用されます。ページのキャッシュサイズ設定は DB インスタンス別に調整できます。

デフォルトのパラメータセットのいずれかを使用してスタートし、微調整する必要があるパラメータにのみ変更を適用します。パラメータグループの使用の詳細については、「Amazon Aurora の DB クラスターパラメータと DB インスタンスパラメータ」を参照してください。Aurora クラスターに適用される、または適用されない構成設定については、使用しているデータベースエンジンに応じて「Aurora MySQL 設定パラメータ」または「Amazon Aurora PostgreSQL のパラメータ」を参照してください。

9. Aurora に接続する

初期のスキーマやデータを設定したり、サンプルクエリを実行したりする場合に、Aurora クラスター内の複数の異なるエンドポイントに接続できます。どのエンドポイントを使用するかは、オペレーションが読み取りであるか (SELECT ステートメントなど)、書き込みであるか (CREATE や INSERT ステートメントなど) に応じて異なります。Aurora クラスターでワークロードを増やして Aurora 機能を試す場合、アプリケーションで各オペレーションを適切なエンドポイントに割り当てることが重要です。

書き込みオペレーション用のクラスターエンドポイントを使用すると、クラスター内で読み取り/書き込み機能を持つ DB インスタンスに常に接続されます。デフォルトでは、Aurora クラスター内で読み取り/書き込み機能を持つ DB インスタンスは 1 つのみです。この DB インスタンスは、プライマリインスタンスと呼ばれます。プライマリインスタンスが使用不能になると、Aurora はフェイルオーバー機構をアクティブ化して別の DB インスタンスをプライマリとして昇格させます。

同様に、SELECT ステートメントをリーダーエンドポイントに振り向けると、処理クエリの作業がクラスター内の複数の DB インスタンスに分散されます。各リーダー接続は、ラウンドロビン DNS 解像度を使用して異なる DB インスタンスに割り当てられます。ほとんどのクエリ作業を読み取り専用の DB Aurora レプリカで処理することで、プライマリインスタンスに対する負荷が減り、プライマリインスタンスの解放された容量ーで DDL ステートメントや DML ステートメントを処理できます。

これらのエンドポイントを使用すると、ハードコードされたホスト名に対する依存が減り、アプリケーションは DB インスタンスの障害から迅速に復旧できます。

> **Note**
>
> Aurora では、独自に作成したカスタムエンドポイントも使用できます。通常、これらのエンドポイントは PoC (概念実証) に不要です。

Aurora レプリカはレプリカラグを伴います。ただし、通常、ラグは 10～20 ミリ秒です。レプリケーションラグをモニタリングして、それがデータの整合性要件の範囲内であるかどうかを判断できます。場合によっては、読み取りクエリで強力な読み取りの整合性 (書き込み直後の読み取りの整合性) が必要になることがあります。このような場合は、リーダーエンドポイントを使用せずに、引き続きクラスターエンドポイントを使用できます。

分散型のパラレル実行で Aurora の機能を最大限に活用するには、接続ロジックの変更が必要になる場合があります。目的は、すべての読み取りリクエストをプライマリインスタンスに送信するのを避けることにあります。読み取り専用の Aurora レプリカはスタンバイしています。すべての同じデータを使用し、SELECT ステートメントを処理する準備ができています。オペレーションの種類別に適切なエンドポイントを使用するようにアプリケーションロジックのコードを記述します。以下の一般的なガイドラインに従ってください。

- すべてのデータベースセッションに 1 つのハードコードされた接続文字列を使用するのを避けます。
- できれば、DDL ステートメントや DML ステートメントなどの書き込みオペレーションをクライアントアプリケーションコードの関数で囲います。これにより、オペレーションの種類別に異なる接続を使用できます。
- クエリオペレーションのための個別の関数を作成します。Aurora は、リーダーエンドポイントへの新しい接続ごとに異なる Aurora レプリカを割り当てることで、読み取りが集中するアプリケーションの負荷を分散します。
- クエリのセットを使用するオペレーションでは、関連するクエリの各セットが終了したら、リーダーエンドポイントへの接続を閉じて再び開きます。この機能をソフトウェアスタックで使用できる場合は、接続プーリングを使用します。クエリを複数の異なる接続に振り向けると、Aurora はクラスター内の複数の DB インスタンスに読み取りワークロードを分散しやすくなります。

Aurora の接続の管理とエンドポイントの一般的な情報については、「Amazon Aurora DB クラスターへの接続」を参照してください。この主題の詳細については、「Aurora MySQL データベース管理者のハンドブック - 接続管理」を参照してください。

10. ワークロードを実行する

スキーマ、データ、および構成設定が済んだら、ワークロードを実行することでクラスターを試すことができます。本番稼働用ワークロードの主な要素をミラーリングする PoC (概念実証) のワークロードを使用します。sysbench や TPC-C などの合成ベンチマークを使用せずに、常に実際のテストやワークロードを使用してパフォーマンスに関する意思決定を行うことをお勧めします。その際は、独自のスキーマ、クエリパターン、使用ボリュームに基づく測定値をできるだけ収集します。

可能な限り、アプリケーションが実行される実際の条件を再現します。例えば、アプリケーションコードは、一般的に Aurora クラスターと同じ AWS リージョンと Virtual Private Cloud (VPC) の Amazon EC2 インスタンスで実行します。本番稼働用アプリケーションが複数のアベイラビリティーゾーンにまたがる複数の EC2 インスタンスで実行される場合は、PoC (概念実証) の環境も同じように設定します。AWS リージョンの詳細については、Amazon RDS ユーザーガイドの「リージョンとアベイラビリティーゾーン」を参照してください。Amazon VPC サービスの詳細については、Amazon VPC ユーザーガイドの「Amazon VPC とは」を参照してください。

アプリケーションの基本的な機能が動作することを確認し、Aurora を通じてデータにアクセスできたら、Aurora クラスターの機能を試すことができます。同時接続とロードバランシング、同時実行トランザクション、自動レプリケーションなどの機能を試すことができます。

この時点までには、データ転送機構に慣れているはずなので、サンプルデータを増やしてテストを実行できます。

このステージでは、メモリの制限や接続の制限などの構成設定を変更した結果を確認できます。「8. 構成設定を指定する」で試した手順を再度参照してください。

スナップショットの作成や復元などの機構を試すこともできます。例えば、AWS インスタンスクラスを変えたり、AWS レプリカの数を変えたりしてクラスターを作成できます。次に、各クラスターで、スキーマとすべてのデータを含む同じスナップショットを復元できます。このサイクルの詳細については、「DB クラスタースナップショットの作成」と「DB クラスターのスナップショットからの復元」を参照してください。

11. パフォーマンスを測定する

当領域のベストプラクティスでは、すべての適切なツールとプロセスが、ワークロードオペレーション中の異常な動作をすばやく隔離するように設定されていることを確認します。また、これらのツールやプロセスが該当する原因を確実に特定できるように設定されていることを確認します。

[モニタリング] タブを参照することで、常にクラスターの現在の状態を確認したり、時間の経過に伴う傾向を調べたりできます。このタブは、Aurora クラスターや DB インスタンスごとにコンソールの詳細ページから利用できます。Amazon CloudWatch モニタリングサービスのメトリクスがグラフ形式で表示されます。メトリクスは、名前、DB インスタンス、および期間をフィルターとして絞り込むことができます。

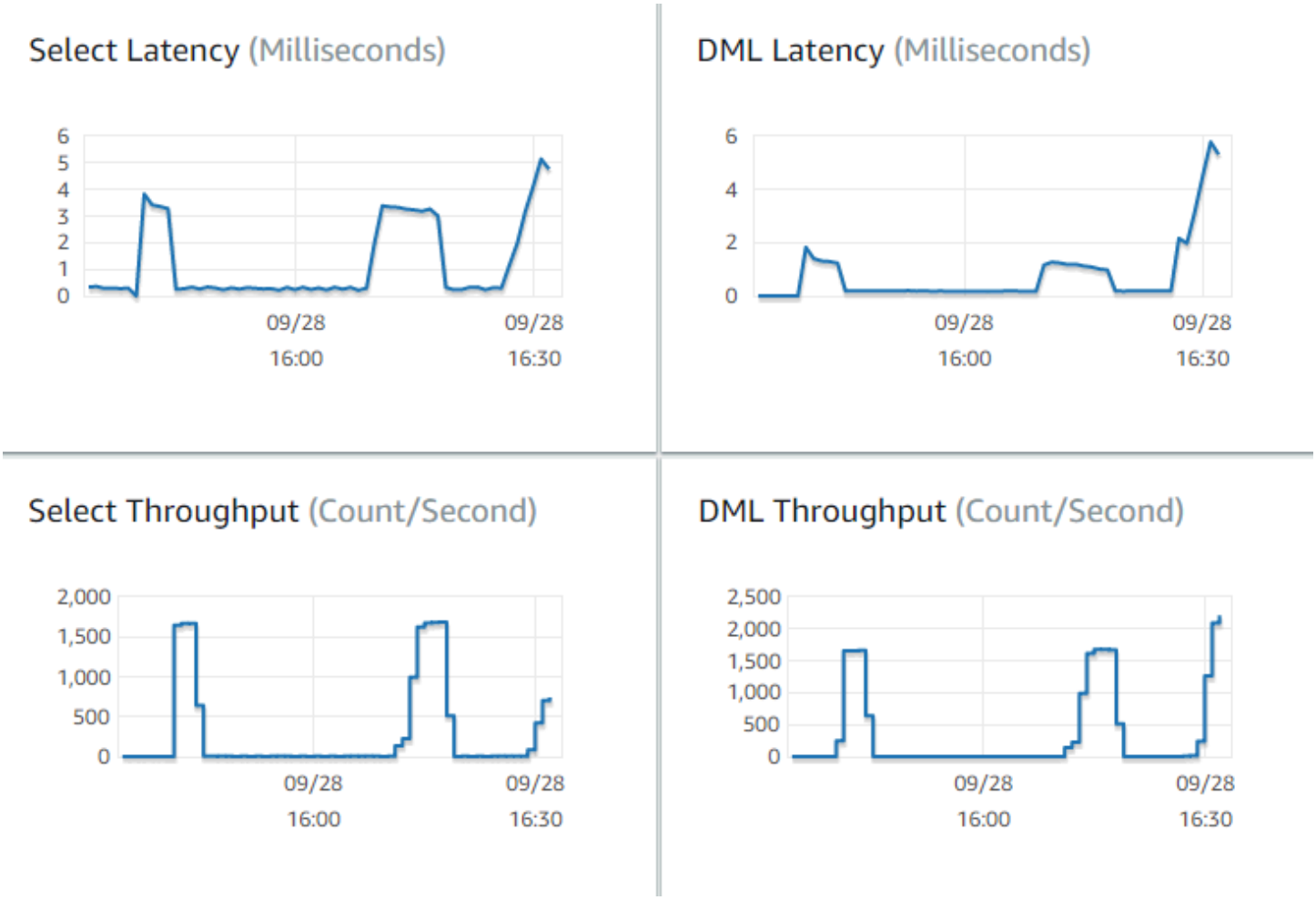
[モニタリング] タブで使用できるオプションを増やすには、クラスター設定で拡張モニタリングと Performance Insights を有効にします。これらのオプションは、クラスターの設定時に選択しなかった場合、後で有効にすることもできます。

パフォーマンスを測定するには、Aurora クラスター全体のアクティビティを示すグラフに主に依存します。Aurora レプリカ間で負荷や応答時間が類似しているかどうかを確認できます。読み取り/書き込みプライマリインスタンスと読み取り専用 Aurora レプリカの間で作業がどのように分担されているかも確認できます。複数の DB インスタンス間に不均衡がある場合や、1 つの DB インスタンスにのみ影響する問題がある場合は、該当するインスタンスについて [モニタリング] で調査できます。

本番稼働用アプリケーションをエミュレートするように環境と実際のワークロードを設定したら、Aurora がどれだけ適切に動作しているかを測定できます。ここで最も重要な問いは以下のとおりです。

- Aurora で処理している 1 秒あたりのクエリ数は? オペレーションの種類別の数については、[スループット] メトリクスで確認できます。
- Aurora で特定のクエリを処理する平均所要時間は? オペレーションの種類別の数については、[レイテンシー] メトリクスで確認できます。

これを行うには、次に示すように、Amazon RDS コンソールで該当する Aurora クラスターの [モニタリング] タブを開きます。



できれば、これらのメトリクスのベースライン値を現在の環境で確立します。これができない場合は、本番稼働用アプリケーションと同等のワークロードを実行して Aurora クラスターでベースラインを構築します。例えば、同数の同時ユーザーおよびクエリを使用して Aurora ワークロードを実行します。次に、異なるインスタンスクラス、クラスターサイズ、構成設定などを試すたびに、どのように値が変わるかを確認します。

スループットの数値が期待より低い場合は、ワークロードのデータベースパフォーマンスに影響する要因をさらに調査します。同様に、レイテンシーの数値が期待より高い場合は、さらに調査します。これを行うには、DB サーバーのセカンダリメトリクス (CPU やメモリ など) をモニタリングします。DB インスタンスがその制限に近づいているかどうかを確認できます。同時実行クエリやより大きなテーブルに対するクエリなどを処理するための DB インスタンスの追加の容量を確認することもできます。

> **Tip**
>
> 想定範囲から外れているメトリクス値を検出するには、CloudWatch のアラームを設定します。

最適な Aurora クラスターサイズと容量を評価する場合は、リソースを過剰にプロビジョニングせずに、アプリケーションのピークパフォーマンスを達成する設定を確認できます。1 つの重要な要素は、Aurora クラスター内の DB インスタンスの適切なサイズを確認することです。まず、現在の本番環境と類似した CPU およびメモリ容量を持つインスタンスサイズを選択します。このインスタンスサイズで、ワークロードのスループットとレイテンシーの数値を収集します。次に、インスタンスを次のより大きなサイズにスケールアップします。スループットとレイテンシーの数値が改善したかどうかを確認します。また、サイズをスケールダウンして、スループットとレイテンシーの数値が同じままであるかどうかを確認します。目標は、できる限り最小のインスタンスで最高のスループットと最低のレイテンシーを達成することにあります。

> **Tip**
>
> Aurora クラスターおよび関連する DB インスタンスのサイズを変更して、既存の容量で突然の想定外のトラフィックスパイクを十分に処理できるようにします。ミッションクリティカルなデータベースの場合は、少なくとも 20% のスペア CPU とメモリ容量を残します。

ウォームな安定した状態でデータベースパフォーマンスを測定できるほどの時間にかけて、パフォーマンステストを実行します。この安定した状態に達するまでに数分または数時間にわたり、ワークロードを実行する必要がある場合もあります。実行のスタート時には、通常、いくらかの変動があります。この変動が発生するのは、各 Aurora レプリカが処理対象の SELECT クエリに応じてキャッシュをウォームアップするためです。

Aurora は、複数の同時実行ユーザーおよびクエリが関わるトランザクションワークロードで最適に動作します。最適なパフォーマンスに応じた負荷を駆動していることを確認するには、マルチスレッドを使用するベンチマークを実行するか、パフォーマンステストの複数のインスタンスを同時に実行します。数百または数千の同時実行クライアントスレッドを使用してパフォーマンスを測定します。本番環境で想定される同時実行スレッドの数をシミュレートします。スレッド数を増やして追加のストレステストを実行し、Aurora のスケーラビリティを測定することもできます。

12. Aurora の高可用性を試す

Aurora の主要な機能では、高可用性を使用します。これらの機能には、自動レプリケーション、自動フェイルオーバー、自動バックアップとポイントインタイム復元、クラスターに DB インスタンスを追加する機能などが含まれます。このような機能の安全性と信頼性は、ミッションクリティカルなアプリケーションにとって重要です。

これらの機能を評価するには考え方を変える必要があります。これまでのアクティビティ (パフォーマンスの測定など) では、すべてが正常に動作した場合を想定して、システムのパフォーマンスを確認しました。高可用性のテストでは、最悪の動作を想定する必要があります。様々な障害 (めったに発生しない障害も含めて) を検討する必要があります。システムが正常かつ迅速に復旧することを確認するために、意図的に問題を作り出す場合もあります。

> **Tip**
>
> 概念実証の段階では、同じ AWS インスタンスクラスを使用して Aurora クラスター内のすべての DB インスタンスを設定します。これにより、DB インスタンスをオフラインにして障害をシミュレートする場合に、パフォーマンスやスケーラビリティの大きな変更なしに、Aurora の可用性の機能を試すことができます。

Aurora クラスターごとに少なくとも 2 つのインスタンスを使用することをお勧めします。Aurora クラスターの DB インスタンスは、最大 3 つのアベイラビリティーゾーン (AZ) にまたがることができます。初期の 2 つまたは 3 つの DB インスタンスのそれぞれを異なる AZ に配置します。より大きなクラスターを使い始めた場合は、AWS リージョンのすべての AZ に DB インスタンスを分散します。これにより、耐障害性機能が強化されます。問題が AZ 全体に影響する場合でも、Aurora は別の AZ の DB インスタンスにフェイルオーバーできます。実行するクラスター内のインスタンス数が 3 つを超えている場合は、3 つすべての AZ にできるだけ均等に DB インスタンスを分散します。

> **Tip**
>
> Aurora クラスターのストレージは、DB インスタンスから独立しています。各 Aurora クラスターのストレージは常に 3 つの AZ にまたがります。
> 高可用性機能をテストする場合は、テストクラスター内の各 DB インスタンスの容量を必ず同じサイズにします。これにより、1 つの DB インスタンスが別のインスタンスに取って代わった場合に、パフォーマンスやレイテンシーなどに想定外の変更が起きないようにします。

障害の条件をシミュレートして高可用性機能をテストする方法については、「障害挿入クエリを使用した Amazon Aurora MySQL のテスト」を参照してください。

PoC (概念実証) における 1 つの目的は、DB インスタンスの最適な数と、これらの DB インスタンスの最適なインスタンスクラスを確認することにあります。これには、高可用性の要件とパフォーマンスの要件の間でバランスを取る必要があります。

Aurora では、クラスター内の DB インスタンス数が増えるほど、高可用性の利点が大きくなります。DB インスタンス数が増えるほど、読み取りが集中するアプリケーションのスケーラビリティも向上します。Aurora は、SELECT クエリのための複数の接続を、読み取り専用の Aurora レプリカ間で分散することができます。

一方、DB インスタンス数を制限すると、プライマリノードからのレプリケーショントラフィックが減少します。レプリケーショントラフィックは、ネットワークの帯域幅を消費します。これは全体的なパフォーマンスとスケーラビリティに伴う別の側面です。したがって、書き込みが集中する OLTP アプリケーションの場合、多数の小さい DB インスタンスを使用しないで、少数の大きな DB インスタンスを使用することを優先します。

代表的な Aurora クラスターでは、1 つの DB インスタンス (プライマリインスタンス) ですべての DDL ステートメントと DML ステートメントを処理します。他の DB インスタンス (Aurora レプリカ) は、SELECT ステートメントのみを処理します。各 DB インスタンスが正確に同じ量の作業を行うわけではありませんが、クラスター内のすべての DB インスタンスに同じインスタンスクラスを使用することをお勧めします。これにより、障害が発生して、Aurora が読み取り専用 DB インスタンスの 1 つを新しいプライマリインスタンスとして昇格させた場合、このプライマリインスタンスの容量は以前と同じになります。

同じクラスター内の DB インスタンス間で異なる容量を使用する必要がある場合は、DB インスタンスのフェイルオーバー階層を設定します。これらの階層は、フェイルオーバー機構で Aurora レプリカを昇格させる順番を決定します。DB インスタンスが他の DB インスタンスより大きいか、より小さいほど、より低いフェイルオーバー階層に配置します。これにより、これらのインスタンスの昇格順位は後回しになります。

Aurora のデータ復旧機能 (自動ポイントインタイプ復元、手動スナップショットと復元、クラスターのバックトラッキングなど) を試します。必要に応じて、スナップショットを他の AWS リージョンにコピーしたり、他の AWS リージョン内に復元したりして、DR シナリオを模倣します。

貴社の目標復旧時間 (RTO)、目標復旧時点 (RPO)、地理的冗長性に関する要件を確認します。ほとんどの組織では、これらの項目を災害対策という大まかなカテゴリに一括しています。このセクションで説明した Aurora の高可用性機能を災害対策プロセスのコンテキストで評価し、RTO と RPO の要件を満たしていることを確認します。

13. 次のステップ

PoC (概念実証) プロセスが正常に完了したら、想定されるワークロードに基づいて Aurora が適切なソリューションであることを確認します。これまでのプロセス全体を通じて、Aurora が実際の運用環境でどのように動作するかを確認し、成功の条件に照らして測定を行いました。

Aurora を使用してデータベース環境を起動して稼動中の状態になったら、より詳細な評価ステップに移動して、最終的な移行と本番稼働用デプロイに備えることができます。実際の状況に応じて、これらの他のステップは PoC (概念実証) プロセスに含まれる場合と含まれない場合があります。移行および移植アクティビティの詳細については、AWS ホワイトペーパー「Aurora 移行ハンドブック」を参照してください。

また、別のステップとして、本番環境のセキュリティ要件を満たすように設計された、ワークロード関連のセキュリティ設定を検討します。Aurora クラスターのマスターユーザー認証情報に対するアクセスを保護するために、どの制御を実装するかを計画します。Aurora クラスターに保存されたデータに対するアクセス制御でデータベースユーザーが果たすロールと責任を定義します。アプリケーション、スクリプト、およびサードパーティー製のツールやサービスに対するデータベースアクセス要件を考慮します。AWS Secrets Manager や AWS Identity and Access Management (IAM) 認証などの AWS のサービスや機能を試します。

この時点では、Aurora でベンチマークテストを実行するための手順とベストプラクティスを理解しているはずです。追加のパフォーマンスチューニングが必要であることが判明する場合があります。詳細については、「Aurora DB クラスターのパフォーマンスとスケーリングの管理」、「Amazon Aurora MySQL パフォーマンスの拡張」、「Amazon Aurora PostgreSQL の管理」および「Amazon Aurora での Performance Insights を使用したDB 負荷のモニタリング」を参照してください。追加のチューニングを行う場合は、PoC (概念実証) 中に収集したメトリクスをよく理解していることを確認してください。次のステップとして、構成設定、データベースエンジン、およびデータベースバージョンの異なるオプションを選択して新しいクラスターを作成できます。または、特定のユースケースのニーズを満たすために特化された種類の Aurora クラスターを作成できます。

例えば、ハイブリッドトランザクション/分析処理 (HTAP) アプリケーション用の Aurora パラレルクエリクラスターを試すことができます。災害対策の目的またはレイテンシーを短縮する目的で、広範な地理的分散が不可欠である場合は、Aurora のグローバルデータベースを試すことができます。ワークロードが断続的である場合や、開発/テストシナリオで Aurora を使用している場合は、Aurora Serverless のクラスターを試すことができます。

本番稼働用クラスターでは、大量の着信接続の処理が必要になる場合もあります。これらの処理の詳細については、AWS ホワイトペーパーのAurora MySQL データベース管理者のハンドブック - 接続管理を参照してください。

概念実証が完了し、ユースケースが Aurora に適していないと判断した場合は、以下に示す他の AWS のサービスを検討します。

- 純粋に分析的なユースケースの場合、OLAP ワークロードにより適した列ストレージ形式や他の機能をワークロードに有効に利用できます。このようなユースケースに対応する AWS サービスには、次のものがあります。
  - Amazon Redshift
  - Amazon EMR
  - Amazon Athena
- 多くのワークロードは、Aurora と上記サービスの 1 つ以上との組み合わせから利点を得られます。これらのサービス間でデータを移動するには、以下を使用できます。
  - AWS Glue
  - AWS DMS
  - Amazon S3 からのインポート (Amazon Aurora ユーザーガイド を参照)
  - Amazon S3 へのエクスポート (Amazon Aurora ユーザーガイド を参照)
  - その他多くの一般的な ETL ツール

Amazon Aurora でのセキュリティ

AWS では、クラウドのセキュリティが最優先事項です。AWS のお客様は、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャから利点を得られます。

セキュリティは、AWS とお客様の間の共有責任です。責任共有モデルでは、この責任がクラウドのセキュリティおよびクラウド内のセキュリティとして説明されています。

- クラウドのセキュリティ - AWS は、AWS クラウドで AWS のサービスを実行するインフラストラクチャを保護する責任を負います。また、AWS は、使用するサービスを安全に提供します。AWS コンプライアンスプログラムの一環として、サードパーティーの監査が定期的にセキュリティの有効性をテストおよび検証しています。Amazon Aurora (Aurora) に適用するコンプライアンスプログラムの詳細については、「コンプライアンスプログラムによる AWS 対象範囲内のサービス」を参照してください。
- クラウド内のセキュリティ - お客様の責任は、使用する AWS のサービスに応じて異なります。また、お客様は、お客様のデータの機密性、組織の要件、および適用可能な法律および規制などの他の要因についても責任を担います。

このドキュメントは、Amazon Aurora を使用する際に共有責任モデルを適用する方法を理解するのに役立ちます。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達成するために Amazon Aurora を設定する方法を示します。また Amazon Aurora リソースのモニタリングやセキュリティ確保に役立つ他の AWS サービスの使用方法についても確認頂けます。

DB のクラスター上の Amazon Aurora リソースとデータベースへのアクセスを管理できます。アクセスの管理に使用する方法は、ユーザーが Amazon Aurora で実行する必要のあるタスクのタイプによって異なります。

- Amazon VPC サービスに基づき、Virtual Private Cloud (VPC) 内で DB クラスターを実行して、ネットワークアクセス制御を最大限に拡張します。VPC での DB クラスターの作成の詳細については、「Amazon VPC VPC とAmazon Aurora」を参照してください。
- AWS Identity and Access Management (IAM) ポリシーを使用して、どのユーザーが Amazon Aurora リソースの管理を許可されるかを決定するアクセス許可を割り当てます。例えば、IAM を使用して、いずれのユーザーが DB のクラスターの作成、情報入手、変更、削除、リソースのタグ付け、セキュリティグループの変更を許可されるかを決定します。

IAM ポリシーの例を確認するには、「Amazon Aurora のアイデンティティベースのポリシーの例」を参照してください。

- セキュリティグループを使用して、どの IP アドレスまたは Amazon EC2 インスタンスが DB のクラスター上のデータベースに接続できるかを制御します。DB のクラスターを初めて作成すると、そのインスタンスのファイアウォールにより、関連付けられるセキュリティグループによって指定されたルールに従ったアクセスを除き、データベースへのアクセスはすべて禁止されます。
- Aurora MySQL または Aurora PostgreSQL を実行している DB クラスターと Secure Socket Layer (SSL) または Transport Layer Security (TLS) の接続を使用します。DB クラスターと SSL/TLS を使用する方法の詳細については、「SSL/TLS を使用した DB クラスターへの接続の暗号化」を参照してください。
- Amazon Aurora 暗号化を使用して、DB クラスターおよび保管時のスナップショットのセキュリティを確保します。Amazon Aurora 暗号化は、業界スタンダードの AES-256 暗号化アルゴリズムを使用して、DB クラスターをホストしているサーバーでデータを暗号化します。詳細については、「Amazon Aurora リソースの暗号化」を参照してください。
- DB エンジンのセキュリティ機能を使用して、DB クラスターのデータベースにログインできるユーザーを制御します。これらの機能は、データベースがローカルネットワーク上にあるかのように動作します。

  Aurora MySQL のセキュリティについては、「Amazon Aurora MySQL でのセキュリティ」を参照してください。Aurora PostgreSQL のセキュリティについては、「Amazon Aurora PostgreSQL でのセキュリティ」を参照してください。

Aurora は、マネージド型データベースサービスである Amazon Relational Database Service (Amazon RDS) の一部です。Amazon RDS は、クラウドでリレーショナルデータベースを簡単に設定、運用、および拡張することができるウェブサービスです。Amazon RDS にまだ慣れていない場合は、Amazon RDS ユーザーガイドを参照してください。

Aurora には、高性能のストレージサブシステムが含まれています。MySQL と PostgreSQL との互換性のあるデータベースエンジンは、その高速分散ストレージを利用するようにカスタマイズされています。また、Aurora はデータベースのクラスター化とレプリケーションを自動化しスタンダード化します。通常、これらはデータベースの設定と管理に伴う最も困難な作業に属します。

Amazon RDS と Aurora ではいずれも、プログラムで RDS API にアクセスすることができます。また、AWS CLI を使用して、インタラクティブに RDS API にアクセスすることもできます。一部の RDS API オペレーションと AWS CLI コマンドは、Amazon RDS および Aurora に適用されるのに対し、それ以外は Amazon RDS と Aurora のいずれかにのみ適用されます。RDS API オペレーショ

ンについては、Amazon RDS API リファレンスを参照してください。AWS CLI の詳細については、「Amazon RDS の AWS Command Line Interface リファレンス」を参照してください。

> **Note**
>
> 目的のユースケースに対してのみ、セキュリティを設定する必要があります。Amazon Aurora で管理されるプロセス用にセキュリティアクセスを設定する必要はありません。このプロセスには、バックアップの作成、自動フェイルオーバーなどがあります。

Amazon Aurora リソースや DB クラスター上のデータベースに対するアクセスの管理の詳細については、以下のトピックを参照してください。

トピック

- Amazon Aurora でのデータベース認証
- Amazon Aurora および AWS Secrets Manager によるパスワード管理
- Amazon RDS でのデータ保護
- Amazon Aurora での Identity and Access Management
- Amazon Aurora でのログ記録とモニタリング
- Amazon Aurora のコンプライアンス検証
- Amazon Aurora の耐障害性
- Amazon Aurora でのインフラストラクチャセキュリティ
- Amazon RDS API とインターフェイス VPC エンドポイント (AWS PrivateLink)
- Amazon Aurora のセキュリティのベストプラクティス
- セキュリティグループによるアクセス制御
- マスターユーザーアカウント権限
- Amazon Aurora のサービスにリンクされたロールの使用
- Amazon VPC VPC とAmazon Aurora

Amazon Aurora でのデータベース認証

Amazon Aurora は、データベースユーザを認証するいくつかの方法をサポートしています。

パスワード認証は、すべての DB クラスターでデフォルトで使用できます。Aurora MySQL および Aurora PostgreSQL の場合、同じ DB クラスターに IAM データベース認証と Kerberos 認証のいずれかまたは両方を追加することもできます。

パスワード、Kerberos、および IAM データベース認証では、データベースに対する認証にはさまざまな方法が使用されます。したがって、特定のユーザーは、1 つの認証方法のみを使用してデータベースにログインできます。

PostgreSQL の場合は、特定のデータベースのユーザーに対して、次のロール設定の 1 つだけを使用します。

- IAM データベース認証を使用するには、`rds_iam`ロールをユーザーに割り当てます。
- Kerberos 認証を使用するには、`rds_ad`ロールをユーザーに割り当てます。
- パスワード認証を使用するには、`rds_iam` または `rds_ad` ロールをユーザーに割り当てないでください。

ネストされた許可アクセスによって直接的または間接的に PostgreSQL データベースのユーザーに `rds_iam` ロールと `rds_ad` ロールを両方を割り当てないでください。`rds_iam`ロールがマスターユーザーに追加されると、IAM 認証はパスワード認証よりも優先されるため、マスターユーザーは IAM ユーザーとしてログインする必要があります。

> **Important**
>
> アプリケーションではマスターユーザーを直接使用しないことを強くお勧めします。代わりに、アプリケーションに必要な最小の特権で作成されたデータベースユーザーを使用するというベストプラクティスに従ってください。

トピック

パスワード認証

パスワード認証を使用すると、データベースがユーザーアカウントのすべての管理を行います。DB エンジンがパスワードを指定するのに必要な正しい句を使用して、`CREATE USER` などの SQL 文で

ユーザーを作成します。例えば、MySQL の文は `CREATE USER ## IDENTIFIED BY #####` となりますが、PostgreSQLでは `CREATE USER ## WITH PASSWORD #####` となります。

パスワード認証を使用すると、データベースがユーザーアカウントを制御および認証します。DB エンジンに強力なパスワード管理機能がある場合は、セキュリティを強化できます。ユーザーコミュニティが小規模である場合は、パスワード認証を使用すると、データベース認証が管理しやすくなります。この場合、クリアテキストパスワードが生成されるため、AWS Secrets Manager との統合によってセキュリティが強化されます。

Amazon Aurora での Secrets Manager の使用については、AWS Secrets Manager ユーザーガイドの「基本シークレットの作成」と「サポートされている Amazon RDS データベースのシークレットのローテーション」を参照してください。カスタムアプリケーションにおいてシークレットをプログラムで取得する方法については、AWS Secrets Manager ユーザーガイドの「シークレット値の取得」を参照してください。

IAM データベース認証

AWS Identity and Access Management (IAM) データベース認証を使用して、DB クラスターを認証できます。この認証方法では、DB クラスターに接続するときにパスワードを使用する必要はありません。代わりに、認証トークンを使用します。

特定の DB エンジンの可用性など、IAM データベース認証の詳細については、「 の IAM データベース認証」を参照してください。

Kerberos 認証

Amazon Aurora で、Kerberos と Microsoft Active Directory を使用した、データベースユーザーの外部認証がサポートされるようになりました。Kerberos は、ネットワーク経由でパスワードを送信する必要をなくすためにチケットと対称キー暗号化を使用するネットワーク認証プロトコルです。Kerberos は Active Directory に組み込まれており、データベースなどのネットワークリソースに対するユーザー認証を行えるように設計されています。

Amazon Aurora での Kerberos と Active Directory のサポートにより、データベースユーザーのシングルサインオンおよび一元化認証という利点が得られます。ユーザー資格情報を Active Directory に保持できます。Active Directory には、複数の DB クラスターの資格情報を保存し、管理する一元的な場所が用意されています。

データベースユーザーが DB クラスターに対して認証できるようにするには、2 つの方法があります。AWS Directory Service for Microsoft Active Directory またはオンプレミスの Active Directory に格納されている資格情報を使用できます。

Aurora は、Aurora MySQL および Aurora PostgreSQL DB クラスターについて Kerberos 認証をサポートしています。Aurora MySQL の Kerberos 認証の詳細については、「Aurora MySQL での Kerberos 認証の使用」を参照してください。

Aurora PostgreSQL DB クラスターは、Kerberos 認証により、一方向および双方向のフォレストの信頼関係をサポートしています。詳細については、「Aurora PostgreSQL で Kerberos 認証を使用する」を参照してください。

Amazon Aurora および AWS Secrets Manager によるパスワード管理

Amazon Aurora は Secrets Manager と統合して、DB クラスターのマスターユーザーパスワードを管理します。

トピック

リージョンとバージョンの可用性

機能の可用性とサポートは、各データベースエンジンの特定のバージョンと AWS リージョン によって異なります。Secrets Manager と Amazon Aurora の統合によるバージョンとリージョンの可用性の詳細については、「Secrets Manager 統合でサポートされているリージョンと Aurora DB エンジン」を参照してください。

Secrets Manager と Amazon Aurora の統合に関する制限事項

Secrets Manager によるマスターユーザーパスワードの管理は、以下の機能ではサポートされていません。

- Amazon RDS ブルー/グリーンデプロイ
- Aurora Global Database の一部である DB クラスター。
- Aurora Serverless v1 DB クラスター
- Aurora MySQL クロスリージョンリードレプリカ

- Secrets Manager でリードレプリカのマスターユーザーパスワードを管理する

AWS Secrets Manager を使用したマスターユーザーパスワード管理の概要

AWS Secrets Manager を使用すると、コード内のハードコードされた認証情報 (データベースパスワードを含む) を Secrets Manager への API コールで置き換えて、プログラムでシークレットを取得することができます。Secrets Manager の詳細については、AWS Secrets Manager ユーザーガイドを参照してください。

データベースシークレットを Secrets Manager に保存すると、AWS アカウント に料金が発生します。料金については、「AWS Secrets Manager 料金表」を参照してください。

次のいずれかのオペレーションを実行するときに、 Aurora が Amazon Amazon Aurora DB クラスターのマスターユーザーパスワードを Secrets Manager で管理するように指定できます。

- DB クラスターを作成する
- DB クラスターを変更する
- Amazon S3 から DB クラスターを復元する (Aurora MySQL のみ)

Aurora が Secrets Manager でマスターユーザーパスワードを管理するように指定すると、 Aurora はパスワードを生成して Secrets Manager に保存します。シークレットを直接操作して、マスターユーザーの認証情報を取得できます。また、カスタマーマネージドキーを指定してシークレットを暗号化したり、Secrets Manager が提供する KMS キーを使用したりすることもできます。

Aurora はシークレットの設定を管理し、デフォルトで 7 日ごとにシークレットをローテーションします。ローテーションスケジュールなど、一部の設定を変更できます。Secrets Manager でシークレットを管理する DB クラスターを削除すると、シークレットとそれに関連するメタデータも削除されます。

シークレット内の認証情報を使用して DB クラスターに接続するには、Secrets Manager からシークレットを取得します。詳細については、AWS Secrets Manager ユーザーガイドの「AWS Secrets Manager からのシークレットの取得、AWS Secrets Manager シークレットの認証情報を使用して SQL データベースに接続する」を参照してください。

Secrets Manager でマスターユーザーパスワードを管理する利点

Secrets Manager で Aurora マスターユーザーのパスワードを管理することには、次の利点があります。

- Aurora はデータベース認証情報を自動的に生成します。
- Aurora はデータベース認証情報を AWS Secrets Manager に自動的に保存および管理します。
- Aurora は、アプリケーションを変更することなく、データベースの認証情報を定期的にローテーションします。
- Secrets Manager は、データベースの認証情報を人間のアクセスやプレーンテキスト表示から保護します。
- Secrets Manager では、データベース接続用のシークレット内のデータベース認証情報を取得できます。
- Secrets Manager では、IAM を使用してシークレット内のデータベース認証情報へのアクセスをきめ細かく制御できます。
- 必要に応じて、さまざまな KMS キーを使用して、データベースの暗号化を資格情報の暗号化から分離できます。
- データベース認証情報の手動管理やローテーションが不要になります。
- AWS CloudTrail と Amazon CloudWatch を使用すると、データベースの認証情報を簡単にモニタリングできます。

Secrets Manager のメリットの詳細については、「AWS Secrets Manager ユーザーガイド」を参照してください。

Secrets Manager の統合に必要なアクセス許可

Secrets Manager の統合に関連するオペレーションを実行するには、ユーザーが必要なアクセス許可を持っている必要があります。必要な特定のリソースの API オペレーションを実行するためのアクセス許可を付与する IAM ポリシーを作成できます。その後、これらのポリシーを、それらのアクセス許可を必要とする IAM アクセス許可セットまたはロールにアタッチできます。詳細については、「Amazon Aurora での Identity and Access Management」を参照してください。

作成、変更、または復元オペレーションの場合、 Aurora が Secrets Manager でマスターユーザーパスワードを管理するように指定するユーザーには、次のオペレーションを実行するアクセス許可が必要です。

- `kms:DescribeKey`
- `secretsmanager:CreateSecret`
- `secretsmanager:TagResource`

作成、変更、または復元オペレーションの場合、Secrets Manager でシークレットを暗号化するカスタマーマネージドキーを指定するユーザーには、次のオペレーションを実行するアクセス許可が必要です。

- kms:Decrypt
- kms:GenerateDataKey
- kms:CreateGrant

変更オペレーションの場合、Secrets Manager でマスターユーザーパスワードをローテーションするユーザーには、次のオペレーションを実行するアクセス許可が必要です。

- secretsmanager:RotateSecret

AWS Secrets Manager によるマスターユーザーパスワードの Aurora 管理の強化

IAM 条件キーを使用して、AWS Secrets Manager のマスターユーザーパスワードの Aurora 管理を実施できます。次のポリシーでは、マスターユーザーパスワードが Aurora で Secrets Manager で管理されていない限り、ユーザーが DB インスタンスまたは DB クラスターを作成または復元することはできません。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": ["rds:CreateDBInstance", "rds:CreateDBCluster",
 "rds:RestoreDBInstanceFromS3", "rds:RestoreDBClusterFromS3"],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "rds:ManageMasterUserPassword": false
                }
            }
        }
    ]
}
```

> **Note**
>
> このポリシーは、作成時に AWS Secrets Manager でのパスワード管理を強制します。ただし、クラスターを変更することで、Secrets Manager の統合を無効にして、マスターパスワードを手動で設定することができます。
> これを防ぐには、ポリシーのアクションブロックに `rds:ModifyDBInstance`、`rds:ModifyDBCluster` を含めます。これにより、Secrets Manager 統合が有効になっていない既存のクラスターには、以降の変更ができなくなることに注意してください。

IAM ポリシーでの条件キーの使用の詳細については、「Aurora のポリシー条件キー」および「ポリシー例: 条件キーの使用」を参照してください。

Secrets Manager による DB クラスターのマスターユーザーパスワードの管理

以下のアクションを実行すると、Secrets Manager でマスターユーザーパスワードの Aurora 管理を設定できます。

- Amazon Aurora DB クラスターの作成
- Amazon Aurora DB クラスターの変更
- 外部の MySQL データベースから Amazon Aurora MySQL DB クラスターへのデータ移行

これらのアクションを実行するには、RDS コンソール、AWS CLI、または RDS API を使用できます。

コンソール

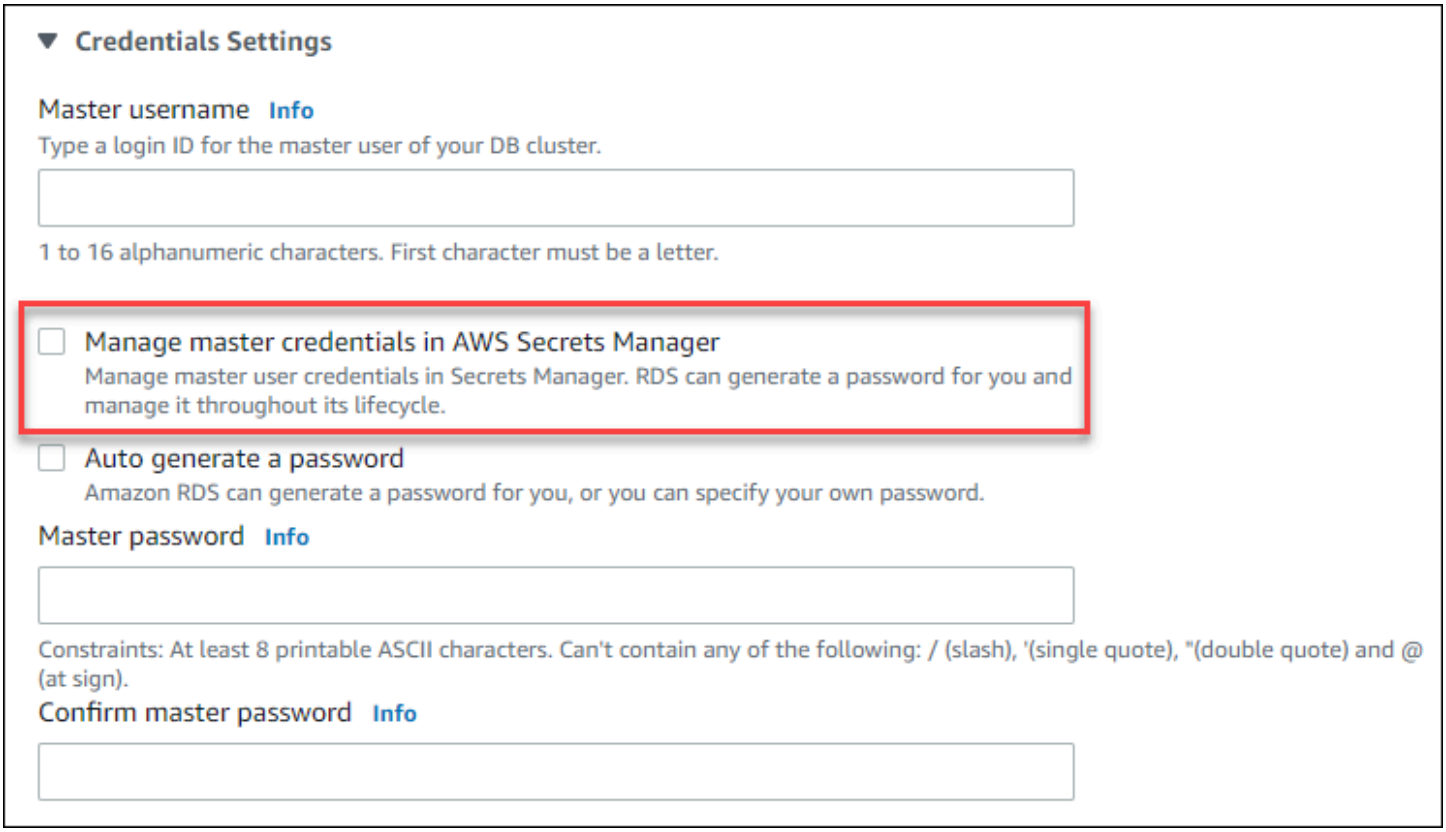
RDS コンソールを使用して DB クラスターを作成または変更する手順に従います。

- DB クラスターの作成
- DB クラスター内の DB インスタンスの変更

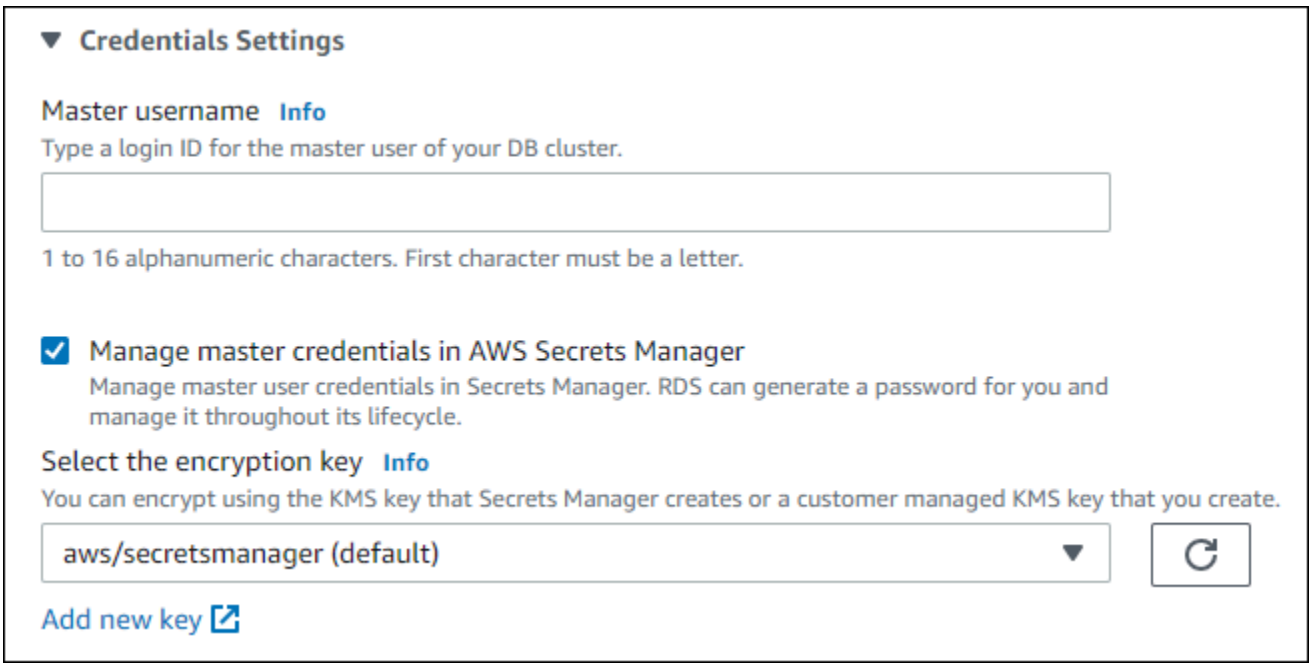
  RDS コンソールでは、任意の DB インスタンスを変更して、DB クラスター全体のマスターユーザーパスワード管理設定を指定できます。
- Amazon S3 バケットからの Amazon Aurora MySQL DB クラスターの復元

RDS コンソールを使用してこれらのオペレーションのいずれかを実行する場合、マスターユーザーパスワードが Aurora で Secrets Manager で管理されるように指定できます。これを行うには、DB クラスターを作成または復元するときに、[Credential settings] (認証情報設定) で [Manage master credentials in AWS Secrets Manager] (でマスター認証情報を管理する) を選択します。DB クラスターを変更する場合は、[Settings] (設定) で [Manage master credentials in AWS Secrets Manager] (でマスター認証情報を管理する) を選択します。

以下の図は、DB インスタンスを作成または復元するときの AWS Secrets Managerでマスター認証情報を管理する設定の例です。



このオプションを選択すると、 Aurora はマスターユーザーパスワードを生成し、そのライフサイクル全体を通じて Secrets Manager で管理します。



シークレットは、Secrets Manager が提供する KMS キーを使用して暗号化するか、自分で作成したカスタマーマネージドキーを使用して暗号化するかを選択できます。 Aurora が DB クラスターのデータベース認証情報を管理した後は、シークレットの暗号化に使用される KMS キーを変更することはできません。

要件に合わせて他の設定を選択できます。

各 DB クラスターの作成時に使用できる設定の詳細については、「Aurora DB クラスターの設定」を参照してください。DB クラスターの変更時に利用できる設定の詳細については、「Amazon Aurora の設定」を参照してください。

AWS CLI

Aurora が Secrets Manager のマスターユーザーパスワードを管理するように指定するには、以下のいずれかの `--manage-master-user-password` コマンドでオプションを指定します。

- create-db-cluster
- modify-db-cluster
- restore-db-cluster-from-s3

これらのコマンドで `--manage-master-user-password` オプションを指定すると、 Aurora はマスターユーザーパスワードを生成し、そのライフサイクル全体を通じて Secrets Manager で管理します。

シークレットを暗号化するには、カスタマーマネージドキーを指定するか、Secrets Manager によって提供されるデフォルトの KMS キーを使用できます。`--master-user-secret-kms-key-id` オプションを使用して、カスタマーマネージドキーを指定します。AWS KMS キー識別子は、KMS キーのキー ARN、キー ID、エイリアス ARN、またはエイリアス名です。別の AWS アカウント で KMS キーを使用するには、キー ARN またはエイリアス ARN を指定します。 Aurora が DB クラスターのデータベース認証情報を管理した後は、シークレットの暗号化に使用される KMS キーを変更することはできません。

要件に合わせて他の設定を選択できます。

各 DB クラスターの作成時に使用できる設定の詳細については、「Aurora DB クラスターの設定」を参照してください。DB クラスターの変更時に利用できる設定の詳細については、「Amazon Aurora の設定」を参照してください。

この例では、 DB クラスターを作成し、 Aurora が Secrets Manager でパスワードを管理するように指定しています。シークレットは、Secrets Manager によって提供される KMS キーを使用して暗号化されます。

Example

Linux、macOS、Unix の場合:

```
aws rds create-db-cluster \
     --db-cluster-identifier sample-cluster \
     --engine aurora-mysql \
     --engine-version 8.0 \
     --master-username admin \
     --manage-master-user-password
```

Windows の場合:

```
aws rds create-db-cluster ^
     --db-cluster-identifier sample-cluster ^
     --engine aurora-mysql ^
     --engine-version 8.0 ^
     --master-username admin ^
     --manage-master-user-password
```

RDS API

Aurora が Secrets Manager のマスターユーザーパスワードを管理するように指定するには、次のいずれかのオペレーションで `ManageMasterUserPassword` パラメータを `true` に設定します。

- CreateDBCluster
- ModifyDBCluster
- RestoreDBClusterFromS3

これらのオペレーションのいずれかで `ManageMasterUserPassword` パラメータを `true` に設定すると、 Aurora はマスターユーザーパスワードを生成し、そのライフサイクル全体を通じて Secrets Manager で管理します。

シークレットを暗号化するには、カスタマーマネージドキーを指定するか、Secrets Manager によって提供されるデフォルトの KMS キーを使用できます。`MasterUserSecretKmsKeyId` パラメータを使用して、カスタマーマネージドキーを指定します。AWS KMS キー識別子は、KMS キーのキー ARN、キー ID、エイリアス ARN、またはエイリアス名です。別の AWS アカウント で KMS キーを使用するには、キー ARN またはエイリアス ARN を指定します。 Aurora が DB クラスターのデータベース認証情報を管理した後は、シークレットの暗号化に使用される KMS キーを変更することはできません。

DB クラスターのマスターユーザーパスワードシークレットのローテーション

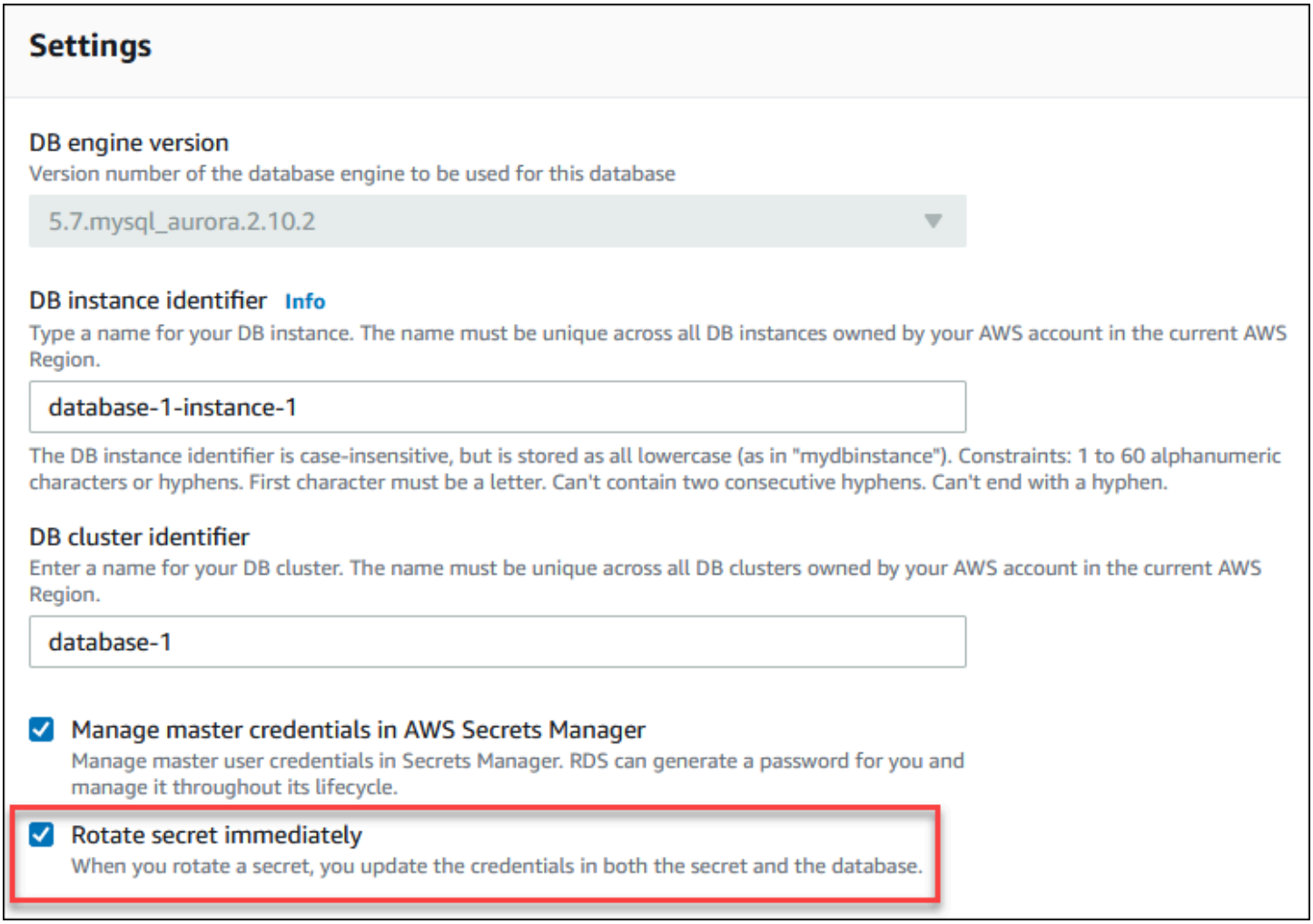
Aurora がマスターユーザーパスワードシークレットをローテーションすると、Secrets Manager は既存のシークレットの新しいシークレットバージョンを生成します。新しいバージョンのシークレットには、新しいマスターユーザーパスワードが含まれています。Aurora は、 DB クラスターのマスターユーザーパスワードを、新しいシークレットバージョンのパスワードと一致するように変更します。

スケジュールされたローテーションを待つ代わりに、シークレットをすぐにローテーションできます。Secrets Manager でマスターユーザーパスワードシークレットをローテーションするには、 DB クラスターを変更します。DB クラスターの変更については、「Amazon Aurora DB クラスターの変更」を参照してください。

RDS コンソール、AWS CLI、または RDS API を使用して、マスターユーザーのパスワードシークレットをすぐに更新できます。新しいパスワードは常に 28 文字で、少なくとも 1 つの大文字と小文字、1 つの数字、1 つの句読点が含まれます。

コンソール

RDS コンソールを使用してマスターユーザーパスワードシークレットをローテーションするには、DB クラスターを変更し、[Settings] (設定) で [Rotate secret immediately] (シークレットを直ちにローテーションする) を選択します。



RDS コンソールを使用して、 および コンソール、CLI、API を使用した DB クラスターの変更 の DB クラスターを変更する手順に従います。確認ページで [Apply immediately] (すぐに適用) を選択する必要があります。

AWS CLI

AWS CLI を使用してマスターユーザーパスワードシークレットをローテーションするには、modify-db-cluster コマンドを使用して `--rotate-master-user-password` オプションを指定します。マスターパスワードをローテーションするときは、`--apply-immediately` オプションを指定する必要があります。

この例では、マスターユーザーパスワードシークレットをローテーションします。

Example

Linux、macOS、Unix の場合:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --rotate-master-user-password \
    --apply-immediately
```

Windows の場合:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --rotate-master-user-password ^
    --apply-immediately
```

RDS API

ModifyDBCluster オペレーションを使用して RotateMasterUserPassword パラメータを true に設定すると、マスターユーザーパスワードシークレットをローテーションできます。マスターパスワードを変更するときは、ApplyImmediately パラメータを true に設定する必要があります。

DB クラスターのシークレットに関する詳細の表示

コンソール (https://console.aws.amazon.com/secretsmanager/) または AWS CLI (get-secret-value Secrets Manager コマンド) を使用して自分のシークレットを取得できます。

Aurora によって管理されているシークレットの Amazon リソースネーム (ARN) は、RDS コンソール、AWS CLI、または RDS API の Secrets Manager で確認できます。

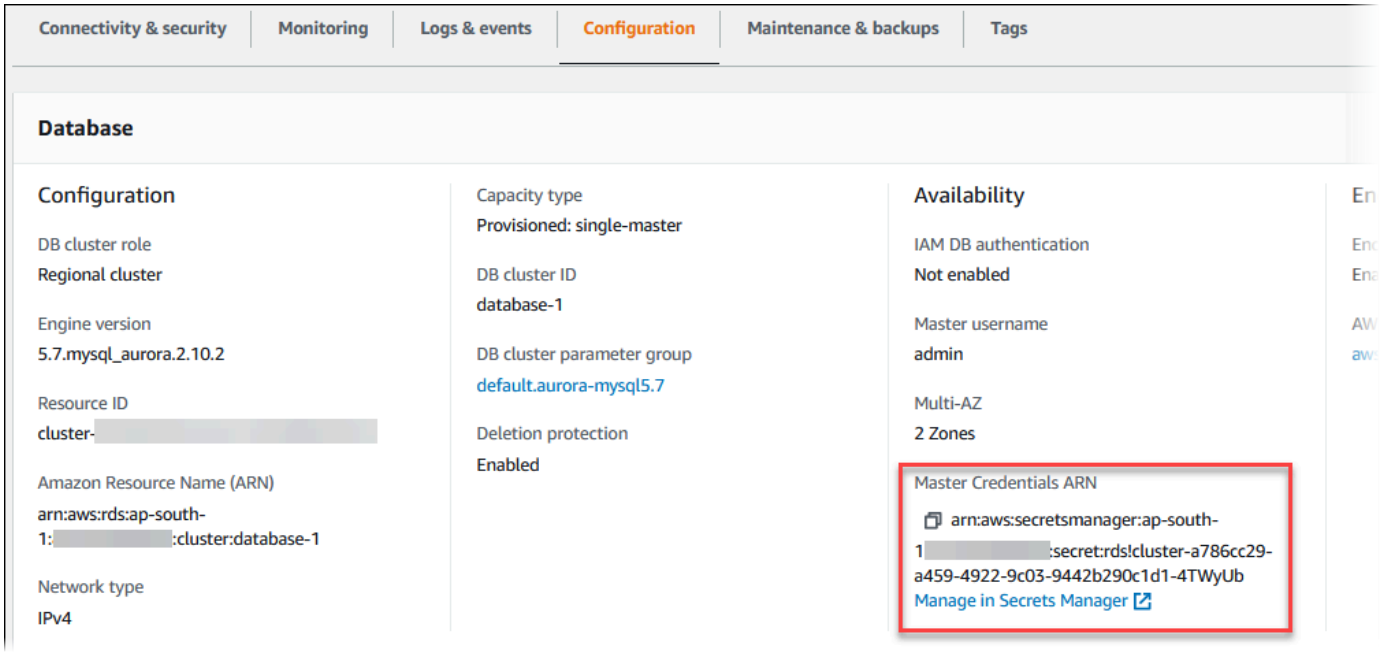
コンソール

Aurora によって管理されているシークレットの詳細を Secrets Manager で表示するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、データベースを選択します。

3. 詳細を表示する DB クラスターの名前を選択します。
4. [設定] タブを選択します。

   [Master Credentials ARN] (マスター認証情報の ARN) では、シークレット ARN を表示できます。



   [Manage in Secrets Manager] (Secrets Managerで管理) で管理リンクをクリックすると、Secrets Manager コンソールでシークレットを表示および管理できます。

AWS CLI

RDS AWS CLI describe-db-cluster コマンドを使用すると、Secrets Manager で Aurora によって管理されているシークレットに関する次の情報を検索できます。

- `SecretArn` – シークレットの ARN
- `SecretStatus` – シークレットのステータス

  設定可能なステータス値は以下のとおりです。

  - `creating` – シークレットは作成中です。
  - `active` – シークレットは通常の使用とローテーションで利用可能です。
  - `rotating` – シークレットはローテーション中です。
  - `impaired` - シークレットはデータベースの認証情報へのアクセスに使用できますが、ローテーションはできません。例えば、アクセス許可が変更されて RDS がシークレットやシークレット

の KMS キーにアクセスできなくなった場合、シークレットがこのステータスになる可能性があります。

シークレットのステータスがこの場合は、ステータスの原因となった状態を修正できます。ステータスの原因となった条件を修正すると、ステータスは次のローテーションまで `impaired` のままです。または、DB クラスターを変更してデータベース認証情報の自動管理をオフにしてから、DB クラスターを再度変更してデータベース認証情報の自動管理をオンにすることもできます。DB クラスターを変更するには、modify-db-cluster コマンドの `--manage-master-user-password` を使用します。

- `KmsKeyId` – シークレットの暗号化に使用する KMS キーの ARN。

特定の DB クラスターの出力を表示する `--db-cluster-identifier` オプションを指定します。この例は、DB クラスターが使用するシークレットの出力を示しています。

Example

```
aws rds describe-db-clusters --db-cluster-identifier mydbcluster
```

以下は、シークレットの出力例を示しています。

```
"MasterUserSecret": {
                "SecretArn": "arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx",
                "SecretStatus": "active",
                "KmsKeyId": "arn:aws:kms:eu-
west-1:123456789012:key/0987dcba-09fe-87dc-65ba-ab0987654321"
            }
```

シークレット ARN がある場合は、get-secret-value Secrets Manager CLI コマンドを使用してシークレットの詳細を表示できます。

この例は、前のサンプル出力のシークレットの詳細を示しています。

Example

Linux、macOS、Unix の場合:

```
aws secretsmanager get-secret-value \
    --secret-id 'arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx'
```

Windows の場合:

```
aws secretsmanager get-secret-value ^
    --secret-id 'arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx'
```

RDS API

Aurora によって管理されているシークレットの ARN、ステータス、および KMS キーは、DescribeDBClusters RDS オペレーションを使用して Secrets Manager で表示し、`DBClusterIdentifier` パラメータを DB クラスター識別子に設定できます。シークレットの詳細は、出力に含まれています。

シークレット ARN がある場合は、GetSecretValue Secrets Manager オペレーションを使用してシークレットの詳細を表示できます。

Amazon RDS でのデータ保護

AWS 責任共有モデルは、Amazon Relational Database Service のデータ保護に適用されます。このモデルで説明されているように、AWS は、AWS クラウド のすべてを実行するグローバルインフラストラクチャを保護する責任を担います。お客様は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。また、使用する AWS のサービスのセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、「データプライバシーのよくある質問」を参照してください。欧州でのデータ保護の詳細については、「AWS セキュリティブログ」に投稿された「AWS 責任共有モデルおよび GDPR」のブログ記事を参照してください。

データを保護するため、AWS アカウント の認証情報を保護し、AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーをセットアップすることをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみを各ユーザーに付与できます。また、次の方法でデータを保護することをおすすめします。

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 が必須です。TLS 1.3 が推奨されます。
- AWS CloudTrail で API とユーザーアクティビティロギングをセットアップします。
- AWS のサービス内でデフォルトである、すべてのセキュリティ管理に加え、AWS の暗号化ソリューションを使用します。

- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API により AWS にアクセスするときに FIPS 140-2 検証済み暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-2」を参照してください。

お客様の E メールアドレスなどの機密情報やセンシティブ情報は、タグや [名前] フィールドなどの自由形式のフィールドに配置しないことを強くお勧めします。これには、コンソール、API、AWS CLI、または AWS SDK を使用して Amazon RDS またはその他の AWS のサービス で作業する場合が含まれます。名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。外部サーバーへの URL を提供する場合は、そのサーバーへのリクエストを検証するための認証情報を URL に含めないように強くお勧めします。

トピック

- 暗号化を使用したデータの保護
- インターネットトラフィックのプライバシー

暗号化を使用したデータの保護

データベースリソースの暗号化を有効にすることができます。また、DB クラスターへの接続を暗号化することもできます。

トピック

- Amazon Aurora リソースの暗号化
- AWS KMS key 管理
- SSL/TLS を使用した DB クラスターへの接続の暗号化
- SSL/TLS 証明書のローテーション

Amazon Aurora リソースの暗号化

Amazon Aurora は Amazon Aurora DB クラスターを暗号化できます。保管時に暗号化されるデータには、DB クラスター、自動バックアップ、リードレプリカ、スナップショット用の基本的なストレージが含まれます。

Amazon Aurora の暗号化された DB クラスターでは、業界スタンダードの AES-256 暗号化アルゴリズムを使用して、Amazon Aurora DB クラスターをホストしているデータをサーバーで暗号化し

ます。データが暗号化されると、Amazon Aurora はパフォーマンスの影響を最小限に抑えながら、データへのアクセスと復号の認証を透過的に処理します。暗号化を使用するために、データベースのクライアントアプリケーションを変更する必要はありません。

> **Note**
>
> 暗号化された/されていない DB のクラスターでは、AWS リージョン間でレプリケートする場合でも、ソースとリードレプリカ間で送信されるデータは暗号化されます。

トピック

Amazon Aurora リソースの暗号化の概要

Amazon Aurora の暗号化された DB クラスターは、基になるストレージへの不正アクセスからデータを保護することによって、データ保護の追加レイヤーを提供します。Amazon Aurora の暗号化を使用して、クラウドにデプロイされるアプリケーションのデータ保護を強化することや、保管時のデータ暗号化に関するコンプライアンスの要件を達成することができます。

Amazon Aurora の暗号化された DB クラスターでは、すべての DB インスタンス、ログ、バックアップ、スナップショットが暗号化されます。Amazon Aurora で暗号化されたクラスターのリードレプリカも暗号化できます。Amazon Aurora は、AWS KMS key を使用して、これらのリソースを暗号化します。KMS キーの詳細については、「AWS Key Management Service デベロッパーガイド」の「AWS KMS keys」と「AWS KMS key 管理」を参照してください。DB クラスター内の各 DB インスタンスは、DB クラスターと同じ KMS キーを使用して暗号化されます。暗号化されたスナップショットをコピーする場合、ソーススナップショットの暗号化に使用した KMS キーとは異なる KMS キーを使用して、ターゲットスナップショットを暗号化できます。

AWS マネージドキー を使用することも、カスタマーマネージドキーを作成することもできます。Amazon Aurora リソースを暗号化および復号するために使用する カスタマーマネージドキーを

管理するには、AWS Key Management Service (AWS KMS) を使用します。AWS KMS は、セキュアで可用性の高いハードウェアとソフトウェアを組み合わせて、クラウド向けにスケールされたキー管理システムを提供します。AWS KMS を使用して、カスタマーマネージドキーを作成し、このカスタマーマネージドキーの使用方法を制御するポリシーを定義できます。AWS KMS は CloudTrail をサポートしているため、KMS キーの使用を監査して、カスタマーマネージドキーが適切に使用されていることを確認できます。カスタマーマネージドキーは、Amazon Aurora およびサポートされている AWS のサービス (Amazon S3、Amazon EBS、Amazon Redshift など) で使用できます。AWS KMS と統合しているサービスのリストについては、「AWS サービス統合」を参照ください。

Amazon Aurora DB クラスターの暗号化

新しい DB クラスターを暗号化するには、コンソールで [Enable encryption] (暗号を有効化) を選択します。DB クラスターの作成については、「Amazon Aurora DB クラスターの作成」を参照してください。

create-db-cluster AWS CLI コマンドを使用して、暗号化された DB クラスターを作成するには、`--storage-encrypted` パラメータを設定します。CreateDBCluster API オペレーションを使用する場合は、`StorageEncrypted` パラメータを true に設定します。

暗号化された DB クラスターを作成するときは、カスタマーマネージドキーまたは Amazon Aurora の AWS マネージドキー を選択して、DB クラスターを暗号化できます。カスタマーマネージドキーのキー識別子を指定しない場合、Amazon Aurora は新しい DB クラスターに AWS マネージドキーを使用します。Amazon Aurora は、Amazon Aurora 用の AWS マネージドキー を AWS アカウントに作成します。AWS アカウントには、AWS リージョンごとに Amazon Aurora の AWS マネージドキー が別々にあります。

KMS キーの詳細については、「AWS Key Management Service デベロッパーガイド」の「AWS KMS keys」を参照してください。

暗号化された DB クラスターを作成したら、その DB クラスターで使用されている KMS キーを変更することはできません。したがって、暗号化された DB クラスターを作成する前に、KMS キーの要件を必ず確認してください。

AWS CLI `create-db-cluster` コマンドを使用して、カスタマーマネージドキーで暗号化された DB クラスターを作成する場合は、`--kms-key-id` パラメータを KMS キーの任意のキー識別子に設定します。Amazon RDS API `CreateDBInstance` オペレーションを使用する場合は、`KmsKeyId` パラメータを KMS キーの任意のキー識別子に設定します。カスタマーマネージドキーを別の AWS アカウントで使用するには、キー ARN またはエイリアス ARN を指定します。

> **Important**
>
> Amazon Aurora が DB クラスター用の KMS キーにアクセスできなくなる場合があります。例えば、Aurora は、KMS キーが有効にならない場合、または Aurora の KMS キーへのアクセス権が失効した場合に、アクセスできなくなります。このような場合、暗号化された DB クラスターは `inaccessible-encryption-credentials-recoverable` 状態になります。DB クラスターのこの状態は 7 日間維持されます。その間に DB クラスターを起動すると、RDS は KMS キーがアクティブかどうかをチェックし、アクティブな場合は DB クラスターを復元します。AWS CLI コマンド start-db-cluster または AWS Management Console を使用して DB クラスターを再起動します。
> DB クラスターが復元されない場合、終了状態 `inaccessible-encryption-credentials` になります。この場合、DB クラスターはバックアップからのみ復元できます。データベース内の暗号化されたデータの消失を防ぐために、暗号化された DB インスタンスのバックアップは常に有効にしておくことを強くお勧めします。

DB クラスターの暗号化が有効になっているかの判別

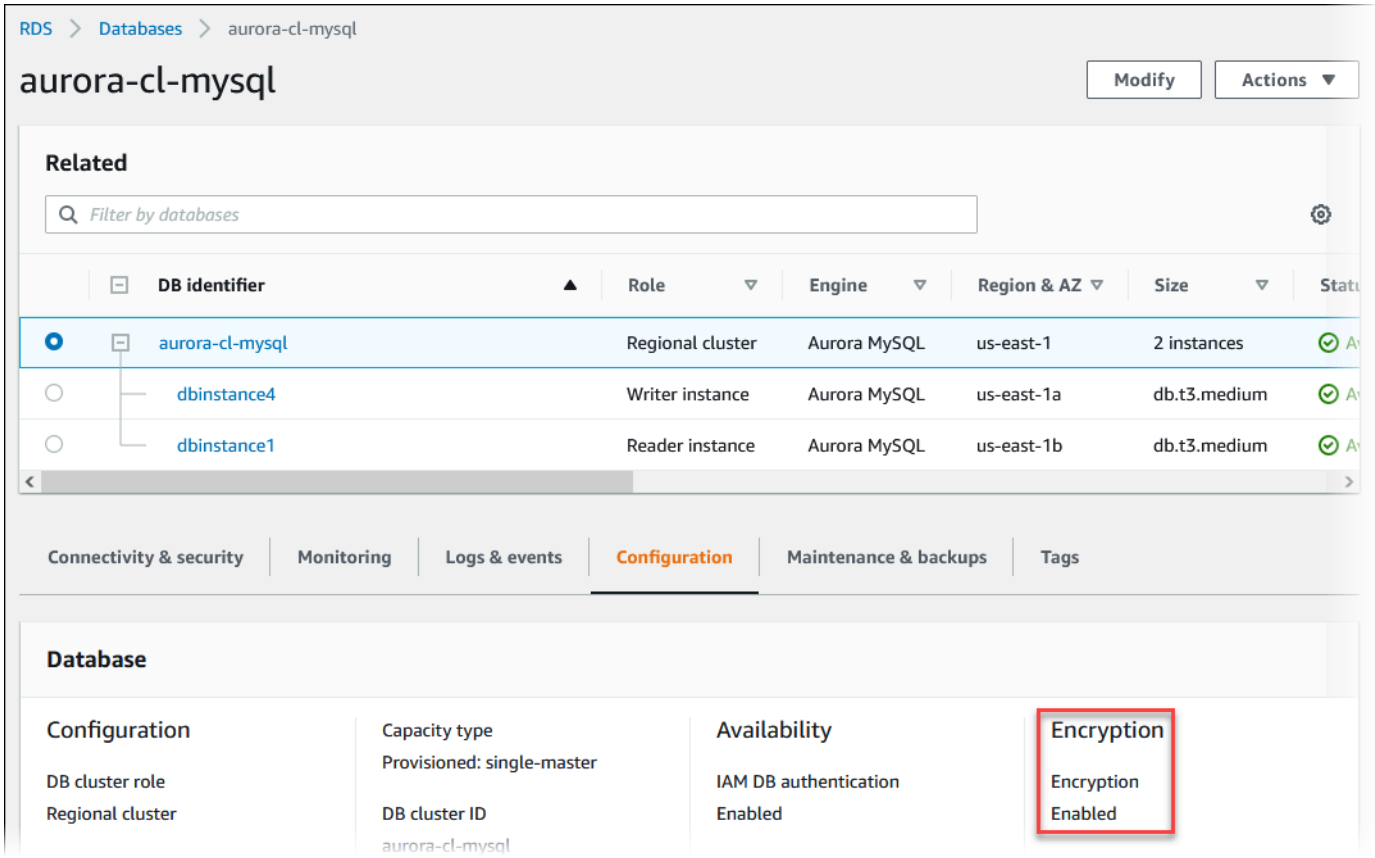
AWS Management Console、AWS CLI、または RDS API を使用して、DB クラスターの保存時の暗号化が有効になっているか判断できます。

コンソール

DB クラスターの保存時の暗号化が有効になっているかどうかを判別するため。

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、データベースを選択します。
3. チェックして詳細を表示したい DB クラスターの名前を選択します。
4. Configuration (設定)タブを選択してEncryption (暗号化)値をチェックします。

   Enable (有効)またはNot enable (無効)のいずれかが表示されています。



AWS CLI

AWS CLI を使用して DB クラスターの保存時の暗号化が有効になっているか判断するには、以下のオプションで describe-db-clusters コマンドを呼び起こします:

- `--db-cluster-identifier` - DB クラスターの名前。

次の例では、mydb DB クラスターの保存時の暗号化に関して TRUE または FALSE のいずれかを返すクエリを使用しています。

Example

```
aws rds describe-db-clusters --db-cluster-identifier mydb --query "*[].
{StorageEncrypted:StorageEncrypted}" --output text
```

RDS API

Amazon RDS API を使用して DB クラスターの保管時の暗号化が有効であるかを判断するには、以下のパラメータで DescribeDBClusters オペレーションを呼び起こします。

- `DBClusterIdentifier` - DB クラスターの名前。

Amazon Aurora の暗号化の可用性

Amazon Aurora 暗号化は、現在 を除く、すべてのデータベースエンジンおよびストレージタイプに使用できます。

> **Note**
>
> Amazon Aurora 暗号化は、db.t2.micro DB インスタンスクラスでは使用できません。

転送中の暗号化

AWS は、すべてのタイプの EC2 インスタンス間において安全でプライベートな接続を提供します。さらに、一部のインスタンスタイプでは、基盤となる Nitro System ハードウェアのオフロード機能を使用して、インスタンス間の転送中のトラフィックを自動的に暗号化します。この暗号化では、256 ビットの暗号化による関連データによる認証暗号化 (AEAD) アルゴリズムを使用します。ネットワークのパフォーマンスには影響しません。インスタンス間でこの追加の転送中トラフィック暗号化をサポートするには、次の要件を満たす必要があります。

- インスタンスは、次のインスタンスタイプを使用します。
  - 汎用: M6i、M6id、M6in、M6idn、 M7g
  - メモリ最適化: R6i、R6id、R6in、R6idn、R7g、X2idn、X2iedn、X2iezn
- 各インスタンスは同じ AWS リージョン に配置します。
- 各インスタンスは同じ VPC 内、あるいはピア接続された VPC 内にあり、トラフィックは仮想ネットワークのデバイスもしくはサービス (ロードバランサーや Transit Gateway など) を通過しないものとします。

Amazon Aurora の暗号化された DB クラスターの制限事項

Amazon Aurora の暗号化された DB クラスターには、以下の制限事項があります。

- 暗号化された DB クラスターの暗号化をオフにすることはできません。
- 暗号化されていない DB クラスターの暗号化されたスナップショットを作成することはできません。

- 暗号化された DB クラスターのスナップショットは、DB クラスターと同じ KMS キーを使用して暗号化する必要があります。
- 暗号化されていない DB クラスターを暗号化された DB クラスターに変換することはできません。ただし、暗号化されていないスナップショットを暗号化された Aurora DB クラスターに復元することはできます。それを行うには、暗号化されていないスナップショットから復元するときに、KMS キーを指定します。
- 暗号化されていない Aurora DB クラスターから、暗号化された Aurora レプリカを作成することはできません。暗号化された Aurora DB クラスターから、暗号化されていない Aurora レプリカを作成することはできません。
- ある AWS リージョンから別のリージョンに暗号化されたスナップショットをコピーするには、送信先 AWS リージョンの KMS キーを指定する必要があります。これは、KMS キーが、作成される AWS リージョンに固有のものであるためです。

  ソーススナップショットはコピープロセス全体で暗号化されたままになります。Amazon Auroraは、コピー処理中にエンベロープ暗号化を使用してデータを保護します。エンベロープ暗号化の仕組みの詳細については、AWS Key Management Service デベロッパーガイドの「エンベロープ暗号化」を参照してください。
- 暗号化された DB クラスターの暗号化を解除することはできません。ただし、暗号化された DB クラスターからデータをエクスポートし、暗号化されていない DB クラスターにデータをインポートすることはできます。

AWS KMS key 管理

Amazon Aurora は、キー管理のために AWS Key Management Service (AWS KMS) と自動的に統合されます。Amazon Aurora では、エンベロープ暗号化が使用されます。エンベロープ暗号化の仕組みの詳細については、AWS Key Management Service デベロッパーガイドの「エンベロープ暗号化」を参照してください。

2 種類の AWS KMS キーを使用して、DB クラスターを暗号化できます。

- KMS キーに対するフル制御の権限が必要な場合は、カスタマーマネージドキーを作成する必要があります。カスタマーマネージドキーの詳細については、AWS Key Management Service デベロッパーガイドの「カスタマーマネージドキー」を参照してください。

  スナップショットを共有する AWS アカウントの AWS マネージドキー を使って暗号化されたスナップショットを共有することはできません。

- AWS マネージドキー は、お客様のアカウントにある KMS キーであり、AWS KMS と統合されている AWS のサービスがお客様に代わって作成し、管理し、使用します。デフォルトでは、RDS AWS マネージドキー (aws/rds) が暗号化に使用されます。RDS AWS マネージドキー の管理、ローテーション、削除はできません。AWS マネージドキー の詳細については、AWS Key Management Service デベロッパーガイドの「AWS マネージドキー 」を参照してください。

Amazon Aurora の暗号化された DB クラスターに使用される KMS キーを管理するには、AWS KMS コンソールの AWS Key Management Service (AWS KMS)、AWS CLI、または AWS KMS API を使用します。AWS マネージドキーまたはカスタマーマネージドキーで実行された、すべてのアクションの監査ログを表示するには、AWS CloudTrail を使用します。キーのローテーションの詳細については、「AWS KMS キーのローテーション」を参照してください。

> **Important**
>
> RDS データベースで使用される KMS キーに対するアクセス許可をオフにする、または取り消すと、KMS キーへのアクセスが必要な際、RDS はユーザーのデータベースをターミナル状態にします。KMS キーへのアクセスを必要とするユースケースに応じて、この変更は即時の場合と遅延する場合があります。この状態では、DB クラスターは使用できなくなり、データベースの現在の状態を復元することができなくなります。DB クラスターを復元するには、RDS 用の KMS キーに対するアクセスを再度有効化し、利用可能な最新のバックアップから DB クラスターを復元します。

カスタマーマネージドキーの使用の承認

Aurora が暗号化オペレーションでカスタマーマネージドキーを使用すると、Aurora リソースを作成または変更するユーザーに代わって動作します。

カスタマーマネージドキーを使用して、Aurora リソースを作成するには、カスタマーマネージドキーで次の操作を呼び出すためのアクセス許可がユーザーに必要になります。

- kms:CreateGrant
- kms:DescribeKey

これらの必要なアクセス許可は、キーポリシーか、キーポリシーで許可されている場合は IAM ポリシーで指定できます。

さまざまな方法で、IAM ポリシー をより厳しくすることができます。例えば、Aurora から発信されるリクエストにのみカスタマーマネージドキーの使用を許可する場合、`rds.<region>.amazonaws.com` 値を指定して kms:ViaService 条件キーを使用します。また、暗号化にカスタマーマネージドキーを使用する条件として、Amazon RDS 暗号化コンテキストでキーまたは値を使用することもできます。

詳細については、「AWS Key Management Service デベロッパーガイド」の「他のアカウントのユーザーに KMS キーの使用を許可する」と「AWS KMS のキーポリシー」を参照してください。

Amazon RDS 暗号化コンテキスト

Aurora が KMS キーを使用する場合、または Amazon EBS が Aurora の代わりに KMS キーを使用する場合、サービスは暗号化コンテキストを指定します。暗号化コンテキストは、データの整合性を保証するために AWS KMS で使用される追加の認証データ (AAD) です。暗号化オペレーションで暗号化コンテキストを指定すると、サービスは復号オペレーションでも同じ暗号化コンテキストを指定する必要があります。そうしないと、復号は失敗します。暗号化コンテキストは AWS CloudTrail ログにも書き込まれるため、特定の KMS キーが使用された理由を理解するのに役立ちます。CloudTrail ログには KMS キーの使用を説明する多くのエントリが含まれている場合があります。各ログエントリの暗号化コンテキストは、その特定の使用理由を判断するのに役立ちます。

Aurora は、少なくとも、次の JSON 形式の例のように、暗号化コンテキストに DB インスタンス ID を常に使用します。

```
{ "aws:rds:db-id": "db-CQYSMDPBRZ7BPMH7Y3RTDG5QY" }
```

この暗号化コンテキストにより、KMS キーが使用された DB インスタンスを識別することができます。

KMS キーが特定の DB インスタンスと特定の Amazon EBS ボリュームに使用されると、次の JSON 形式の例のように、DB インスタンス ID と Amazon EBS ボリューム ID の両方が暗号化コンテキストに使用されます。

```
{
  "aws:rds:db-id": "db-BRG7VYS3SVIFQW7234EJQOM5RQ",
  "aws:ebs:id": "vol-ad8c6542"
}
```

SSL/TLS を使用した DB クラスターへの接続の暗号化

アプリケーションの Secure Socket Layer (SSL) または Transport Layer Security (TLS) を使用して、Aurora MySQL または Aurora PostgreSQL を実行する DB クラスターへの接続を暗号化できます。

SSL/TLS 接続は、クライアントと DB クラスターの間を移動するデータを暗号化することによって、1 つのセキュリティ層を提供します。オプションで、データベースにインストールされたサーバー証明書を検証することで、SSL/TLS 接続でサーバー ID 検証を実行できます。サーバーの ID 検証を必須にするには、次の一般的な手順に従ってください。

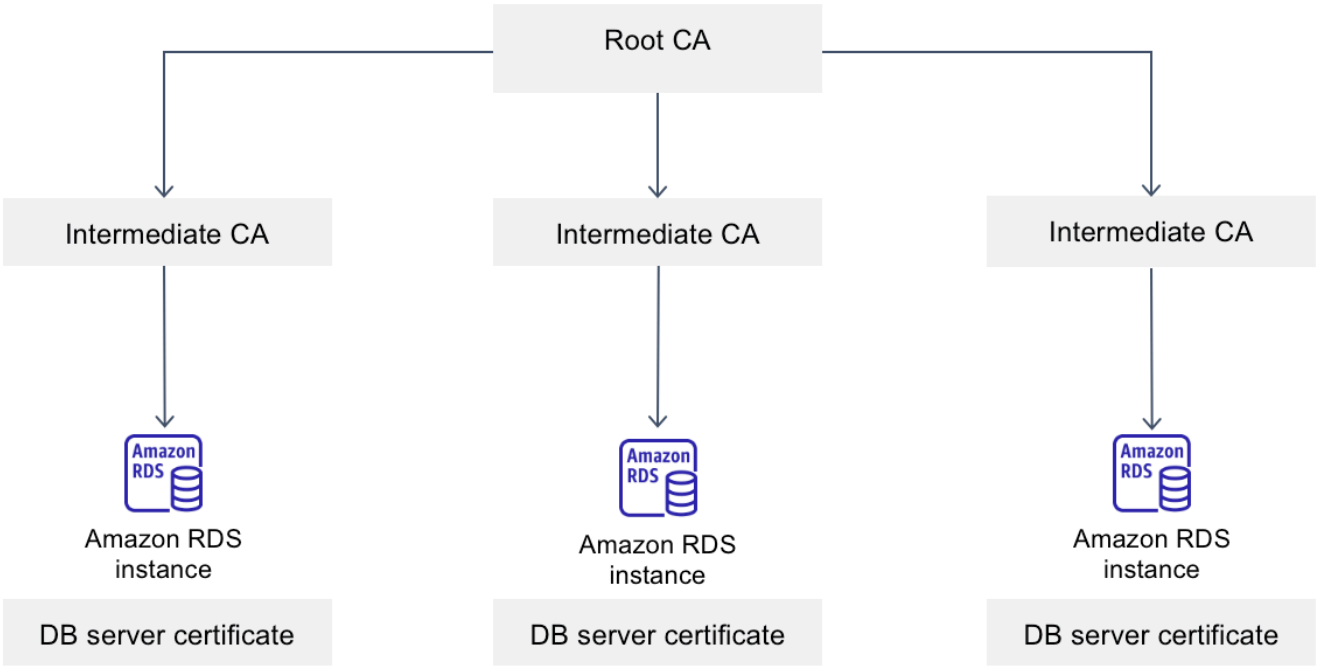
1. データベースの DB サーバー証明書に署名する認証局 (CA) を選択します。認証局の詳細については、「認証局」を参照してください。
2. データベースに接続するときに使用する証明書バンドルをダウンロードします。証明書バンドルをダウンロードするには、 すべての AWS リージョン の証明書バンドルそして特定の AWS リージョン の証明書バンドル。を参照してください。

   > **Note**
   >
   > 証明書はすべて、SSL/TLS 接続を使用したダウンロードでのみ使用可能です。

3. DB エンジンのプロセスで、SSL/TLS Connect を実装するプロセスで、データベースに接続します。各 DB エンジンには SSL/TLS を実装する独自のプロセスがあります。ご使用のデータベースの SSL/TLS の実装方法については、ご使用の DB エンジンに対応する以下のリンクを使用してください。
   - Amazon Aurora MySQL でのセキュリティ
   - Amazon Aurora PostgreSQL でのセキュリティ

認証局

認証局 (CA) は、証明書チェーンの最上位にあるルート CA を識別する証明書です。CA は、各 DB インスタンスにインストールされているサーバー証明書である、DB インスタンス証明書に署名します。DB サーバー証明書は DB インスタンスを信頼できるサーバーとして識別します。



Amazon RDS にはデータベース用の DB サーバー証明書に署名するための、以下の CA が用意されています。

認証局 (CA)	説明
rds-ca-2019	RSA 2048 プライベートキーアルゴリズムと SHA256 署名アルゴリズムを備えた認証局を使用します。この CA は 2024 年に有効期限が切れ、サーバー証明書の自動ローテーションはサポートされていません。この CA を使用して同じ標準を維持したい場合は、rds-ca-rsa2048-g1 CA に切り替えることをお勧めします。
rds-ca-rsa2048-g1	大半の AWS リージョン では、RSA 2048 プライベートキーアルゴリズムと SHA256 署名アルゴリズムを備えた認証局を使用します。 AWS GovCloud (US) Regions では、CA が RSA 2048 プライベートキーアルゴリズムと SHA384 署名アルゴリズムを備えた認証局を使用します。

認証局 (CA)	説明
	この CA は rds-ca-2019 CA よりも長期間有効です。この CA はサーバー証明書の自動ローテーションをサポートします。
rds-ca-rsa4096-g1	RSA 4096 プライベートキーアルゴリズムと SHA384 署名アルゴリズムを備えた認証局を使用します。この CA はサーバー証明書の自動ローテーションをサポートします。
rds-ca-ecc384-g1	ECC 384 プライベートキーアルゴリズムと SHA384 署名アルゴリズムを備えた認証局を使用します。この CA はサーバー証明書の自動ローテーションをサポートします。

> **Note**
>
> AWS CLI を使用している場合は、describe-certificates を使用して、上記の認証局の有効性を確認できます。

これらの CA 証明書は、地域およびグローバル証明書バンドルに含まれています。rds-ca-rsa2048-g1、rds-ca-rsa4096-g1、または rds-ca-ecc384-g1 CA をデータベースで使用すると、RDS はデータベース上で DB サーバー証明書を管理します。RDS は、DB サーバーの証明書の有効期限が切れる前に証明書のローテーションを行います。

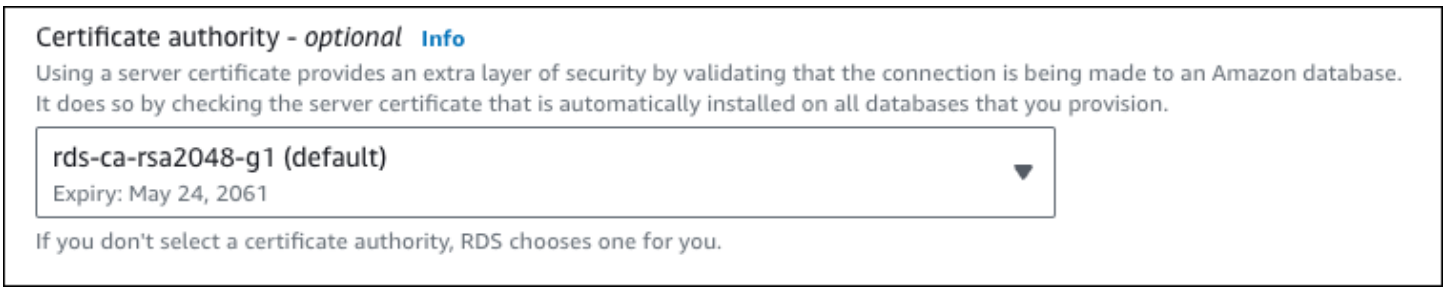
データベースに CA を設定する

データベースの CA は、以下のタスクの実行時に設定できます。

- Aurora DB クラスターの作成 – AWS CLI または RDS API を使用して DB クラスター内で最初の DB インスタンスを作成するとき、Aurora クラスター内の DB インスタンスのための CA を設定できます。現在、RDS コンソールを使用して DB クラスターを作成するときは、DB クラスター内で DB インスタンスの CA は設定できません。手順については、Amazon Aurora DB クラスターの作成 を参照してください。
- DB インスタンスの変更 – DB クラスターで DB インスタンスを変更することにより、CA を設定できます。手順については、DB クラスター内の DB インスタンスの変更 を参照してください。

> **Note**
>
> デフォルトの CA は rds-ca-rsa2048-g1 に設定されています。modify-certificates コマンドを使用して、AWS アカウント のデフォルト CA をオーバーライドできます。

使用可能な CA は、DB エンジンと DB エンジンのバージョンによって異なります。AWS Management Console を使用するとき、次の図に示すように、認証局の設定を使用して CA を選択できます。



コンソールには、DB エンジンおよび DB エンジンのバージョンで利用可能な CA のみが表示されます。AWS CLI を使用しているとき、create-db-instance コマンドまたは modify-db-instance コマンドを使用して DB インスタンスの CA を設定できます。

AWS CLI を使用しているとき、describe-certificates コマンドを使用して、アカウントで使用可能な CA を確認できます。このコマンドでは、出力内の `ValidTill` に各 CA の有効期限も表示されます。describe-db-engine-versions コマンドを使用すると、特定の DB エンジンと DB エンジンのバージョンで使用できる CA を検索できます。

次の例は、PostgreSQL DB エンジンのバージョンのデフォルト RDS で使用できる CA を示しています。

```
aws rds describe-db-engine-versions --default-only --engine postgres
```

以下のような出力が生成されます。使用可能な CA は `SupportedCACertificateIdentifiers` に記載されます。出力には、DB エンジンのバージョンで、`SupportsCertificateRotationWithoutRestart` の再起動なしで証明書のローテーションがサポートされるかどうかも表示されます。

```
{
    "DBEngineVersions": [
        {
```

```
            "Engine": "postgres",
            "MajorEngineVersion": "13",
            "EngineVersion": "13.4",
            "DBParameterGroupFamily": "postgres13",
            "DBEngineDescription": "PostgreSQL",
            "DBEngineVersionDescription": "PostgreSQL 13.4-R1",
            "ValidUpgradeTarget": [],
            "SupportsLogExportsToCloudwatchLogs": false,
            "SupportsReadReplica": true,
            "SupportedFeatureNames": [
                "Lambda"
            ],
            "Status": "available",
            "SupportsParallelQuery": false,
            "SupportsGlobalDatabases": false,
            "SupportsBabelfish": false,
            "SupportsCertificateRotationWithoutRestart": true,
            "SupportedCACertificateIdentifiers": [
                "rds-ca-2019",
                "rds-ca-rsa2048-g1",
                "rds-ca-ecc384-g1",
                "rds-ca-rsa4096-g1"
            ]
        }
    ]
}
```

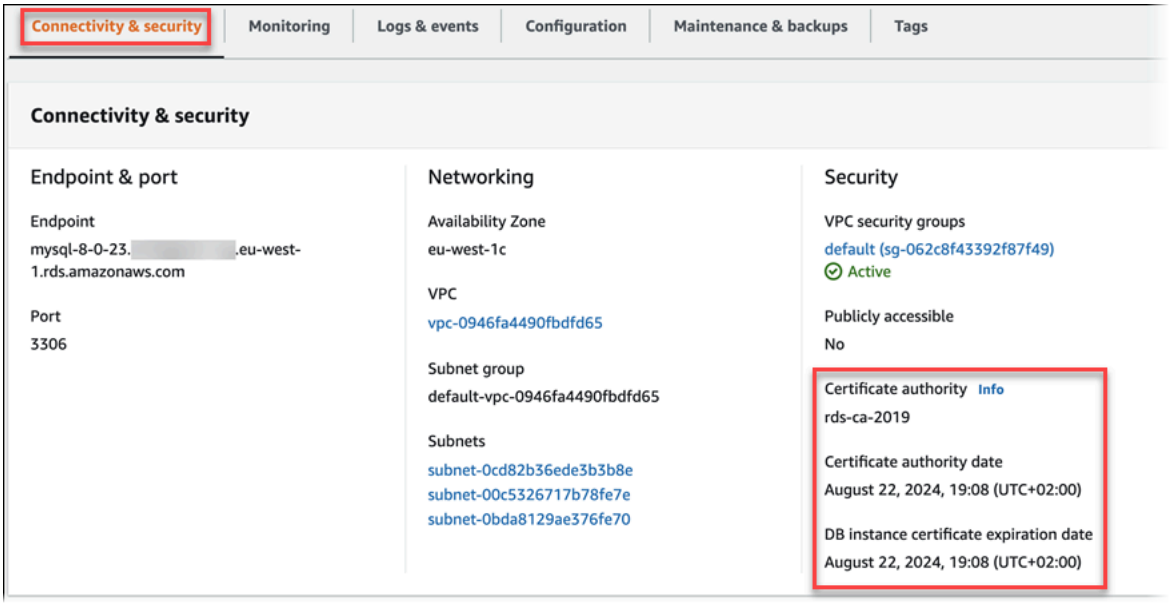
DB サーバーの証明書の有効性

DB サーバー証明書の有効性は、DB エンジンのバージョンと、DB エンジンのバージョンによって異なります。DB エンジンのバージョンで、DB エンジンのバージョンで、DB サーバーのローテーションがサポートされる場合、DB サーバーの証明書の有効期間は 1 年です。それ以外の場合、有効期間は 3 年間です。

DB サーバー証明書ローテーションの詳細については、「サーバー証明書の自動ローテーション」を参照してください。

DB インスタンスの CA の表示

データベースの CA に関する詳細を確認するには、次の画像のように、コンソール内の [接続とセキュリティ] タブを表示します。



AWS CLI を使用している場合は、describe-db-instances コマンドを使用して DB インスタンスの CA の詳細を表示できます。

CA 証明書バンドルの内容を確認するには、次のコマンドを使用します。

```
keytool -printcert -v -file global-bundle.pem
```

すべての AWS リージョン の証明書バンドル

すべての AWS リージョン の証明書バンドルは、https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem からダウンロードできます。

バンドルには、rds-ca-2019 中間証明書とルート証明書の両方が含まれています。またバンドルには、rds-ca-rsa2048-g1、rds-ca-rsa4096-g1、および rds-ca-ecc384-g1 ルート CA 証明書も含まれています。アプリケーション信頼ストアでは、ルート CA 証明書の登録のみが必要です。

アプリケーションが Microsoft Windows にインストールされており、PKCS7 ファイルが必要な場合、PKCS7 証明書バンドルは https://truststore.pki.rds.amazonaws.com/global/global-bundle.p7b からダウンロードできます。

> **Note**
>
> Amazon RDS Proxy および Aurora Serverless v1 は AWS Certificate Manager (ACM) の証明書を使用します。RDS Proxy を使用している場合は、Amazon RDS 証明書をダウンロードしたり、RDS Proxy 接続を使用するアプリケーションを更新したりする必要はありません。詳細については、「RDS Proxy での TLS/SSL の使用」を参照してください。

Aurora Serverless v1 を使用している場合は、Amazon RDS 証明書をダウンロードする必要はありません。詳細については、「Aurora Serverless v1 での TLS/SSL の使用」を参照してください。

特定の AWS リージョン の証明書バンドル

バンドルには、rds-ca-2019 中間証明書とルート証明書の両方が含まれています。またバンドルには、rds-ca-rsa2048-g1、rds-ca-rsa4096-g1、および rds-ca-ecc384-g1 ルート CA 証明書も含まれています。アプリケーション信頼ストアでは、ルート CA 証明書の登録のみが必要です。

AWS リージョン の証明書バンドルを取得するには、次の表の AWS リージョン のリンクからダウンロードします。

AWS リージョン	証明書バンドル (PEM)	証明書バンドル (PKCS7)
米国東部 (バージニア北部)	us-east-1-bundle.pem	us-east-1-bundle.p7b
米国東部 (オハイオ)	us-east-2-bundle.pem	us-east-2-bundle.p7b
米国西部 (北カリフォルニア)	us-west-1-bundle.pem	us-west-1-bundle.p7b
米国西部 (オレゴン)	us-west-2-bundle.pem	us-west-2-bundle.p7b
アフリカ (ケープタウン)	af-south-1-bundle.pem	af-south-1-bundle.p7b
アジアパシフィック (香港)	ap-east-1-bundle.pem	ap-east-1-bundle.p7b
アジアパシフィック (ハイデラバード)	ap-south-2-bundle.pem	ap-south-2-bundle.p7b
アジアパシフィック (ジャカルタ)	ap-southeast-3-bundle.pem	ap-southeast-3-bundle.p7b
アジアパシフィック (メルボルン)	ap-southeast-4-bundle.pem	ap-southeast-4-bundle.p7b
アジアパシフィック (ムンバイ)	ap-south-1-bundle.pem	ap-south-1-bundle.p7b

AWS リージョン	証明書バンドル (PEM)	証明書バンドル (PKCS7)
アジアパシフィック (大阪)	ap-northeast-3-bundle.pem	ap-northeast-3-bundle.p7b
アジアパシフィック (東京)	ap-northeast-1-bundle.pem	ap-northeast-1-bundle.p7b
アジアパシフィック (ソウル)	ap-northeast-2-bundle.pem	ap-northeast-2-bundle.p7b
アジアパシフィック (シンガポール)	ap-southeast-1-bundle.pem	ap-southeast-1-bundle.p7b
アジアパシフィック (シドニー)	ap-southeast-2-bundle.pem	ap-southeast-2-bundle.p7b
カナダ (中部)	ca-central-1-bundle.pem	ca-central-1-bundle.p7b
カナダ西部 (カルガリー)	ca-west-1-bundle.pem	ca-west-1-bundle.p7b
欧州 (フランクフルト)	eu-central-1-bundle.pem	eu-central-1-bundle.p7b
欧州 (アイルランド)	eu-west-1-bundle.pem	eu-west-1-bundle.p7b
欧州 (ロンドン)	eu-west-2-bundle.pem	eu-west-2-bundle.p7b
ヨーロッパ (ミラノ)	eu-south-1-bundle.pem	eu-south-1-bundle.p7b
欧州 (パリ)	eu-west-3-bundle.pem	eu-west-3-bundle.p7b
欧州 (スペイン)	eu-south-2-bundle.pem	eu-south-2-bundle.p7b
欧州 (ストックホルム)	eu-north-1-bundle.pem	eu-north-1-bundle.p7b
欧州 (チューリッヒ)	eu-central-2-bundle.pem	eu-central-2-bundle.p7b
イスラエル (テルアビブ)	il-central-1-bundle.pem	il-central-1-bundle.p7b
中東 (バーレーン)	me-south-1-bundle.pem	me-south-1-bundle.p7b
中東 (アラブ首長国連邦)	me-central-1-bundle.pem	me-central-1-bundle.p7b
南米 (サンパウロ)	sa-east-1-bundle.pem	sa-east-1-bundle.p7b

AWS GovCloud (US) 証明書

AWS GovCloud (US) Region の中間証明書とルート証明書の両方を含む証明書バンドルは、https://truststore.pki.us-gov-west-1.rds.amazonaws.com/global/global-bundle.pem からダウンロードできます。

アプリケーションが Microsoft Windows にインストールされており、PKCS7 ファイルが必要な場合、PKCS7 証明書バンドルは https://truststore.pki.us-gov-west-1.rds.amazonaws.com/global/global-bundle.p7b からダウンロードできます。

バンドルには、`rds-ca-2019` 中間証明書とルート証明書の両方が含まれています。またバンドルには、`rds-ca-rsa2048-g1`、`rds-ca-rsa4096-g1`、および `rds-ca-ecc384-g1` ルート CA 証明書も含まれています。アプリケーション信頼ストアでは、ルート CA 証明書の登録のみが必要です。

AWS GovCloud (US) Region の証明書バンドルを取得するには、次の表の AWS GovCloud (US) Region のリンクからダウンロードします。

AWS GovCloud (US) Region	証明書バンドル (PEM)	証明書バンドル (PKCS7)
AWS GovCloud (米国東部)	us-gov-east-1-bundle.pem	us-gov-east-1-bundle.p7b
AWS GovCloud (米国西部)	us-gov-west-1-bundle.pem	us-gov-west-1-bundle.p7b

SSL/TLS 証明書のローテーション

Amazon RDS 認証局証明書 rds-ca-2019 は、2024 年 8 月に期限切れになるように設定されています。RDS DB インスタンスへの接続に証明書検証付きの Secure Sockets Layer (SSL) または Transport Layer Security (TLS) を使用しているか、使用する予定がある場合は、新しい CA 証明書 rds-ca-rsa2048-g1 の使用を検討してください。現在、証明書検証付きで SSL/TLS を使用していない場合でも、CA 証明書の有効期限が切れている可能性があり、証明書検証付きで SSL/TLS を使用して RDS データベースに接続する予定がある場合は、新しい CA 証明書に更新する必要があります。

更新を行うには、次の手順に従います。新しい CA 証明書を使用するように DB インスタンスを更新する前に、RDS データベースに接続するクライアントまたはアプリケーションを必ず更新します。

Amazon RDS では、AWS セキュリティのベストプラクティスとして、新しい CA 証明書を提供しています。新しい証明書およびサポートしている AWS リージョンに関する詳細は、「SSL/TLS を使用した DB クラスターへの接続の暗号化」を参照してください。

> **Note**
>
> Amazon RDS Proxy および Aurora Serverless v1 は AWS Certificate Manager (ACM) の証明書を使用します。RDS Proxy を使用している場合は、SSL/TLS 証明書を更新するときに、RDS Proxy 接続を使用するアプリケーションを更新する必要はありません。詳細については、「RDS Proxy での TLS/SSL の使用」を参照してください。
> Aurora Serverless v1 を使用している場合は、Amazon RDS 証明書をダウンロードする必要はありません。詳細については、「Aurora Serverless v1 での TLS/SSL の使用」を参照してください。

> **Note**
>
> 2020 年 7 月 28 日より前に作成された、または rds-ca-2019 証明書にアップデートされた DB インスタンスで Go バージョン 1.15 アプリケーションを使用している場合は、証明書を再度更新する必要があります。エンジンに応じて、証明書を rds-ca-rsa2048-g1、rds-ca-rsa4096-g1、または rds-ca-ecc384-g1 に更新してください。新しい CA 証明書識別子を使用して、DB インスタンスの場合は `modify-db-instance` コマンド実行します。`describe-db-engine-versions` コマンドを使用すると、特定の DB エンジンと DB エンジンのバージョンで使用できる CA を検索できます。
> 2020 年 7 月 28 日以降にデータベースを作成したか、その証明書を更新した場合、必要なアクションはありません。詳細については、Go GitHub issue #39568 を参照してください。

トピック

- DB インスタンスの変更による CA 証明書の更新
- メンテナンスの適用による CA 証明書の更新
- サーバー証明書の自動ローテーション
- 証明書を信頼ストアにインポートするためのサンプルスクリプト

DB インスタンスの変更による CA 証明書の更新

次の例では、CA 証明書を RDS CA-2019 から rds-ca-rsa2048-g1 に更新します。別の証明書を選択できます。詳細については、認証局 をご参照ください。

DB インスタンスを変更して CA 証明書を更新するには

1. 「SSL/TLS を使用した DB クラスターへの接続の暗号化」の説明に従って、新しい SSL/TLS 証明書をダウンロードします。
2. 新しい SSL/TLS 証明書を使用するようにアプリケーションを更新します。

   新しい SSL/TLS 証明書のアプリケーションを更新する方法は、特定のアプリケーションにより異なります。アプリケーション開発者と協力して、アプリケーションの SSL/TLS 証明書を更新します。

   SSL/TLS 接続の確認および各 DB エンジン用アプリケーションの更新については、以下のトピックを参照してください。

   - 新しい TLS 証明書を使用して Aurora MySQL DB クラスターに接続するようにアプリケーションを更新する
   - 新しい SSL/TLS 証明書を使用して Aurora PostgreSQL DB クラスターに接続するようにアプリケーションを更新する

   Linux オペレーティングシステムの信頼ストアを更新するサンプルスクリプトについては、「証明書を信頼ストアにインポートするためのサンプルスクリプト」を参照してください。

   > **Note**
   >
   > 証明書バンドルには古い CA と新しい CA の両方の証明書が含まれます。そのため、アプリケーションを安全に更新し、移行期間に接続を維持することができます。AWS Database Migration Service を使用してデータベースを DB クラスターに移行する場合は、移行中の接続を確保するために証明書バンドルを使用することをお勧めします。

3. DB インスタンスを変更して、CA を rds-ca-2019 から rds-ca-rsa2048-g1 に変更します。CA 証明書を更新するためにデータベースの再起動が必要かどうかを確認するには、describe-db-engine-versions コマンドを使用して、`SupportsCertificateRotationWithoutRestart` フラグをチェックします。

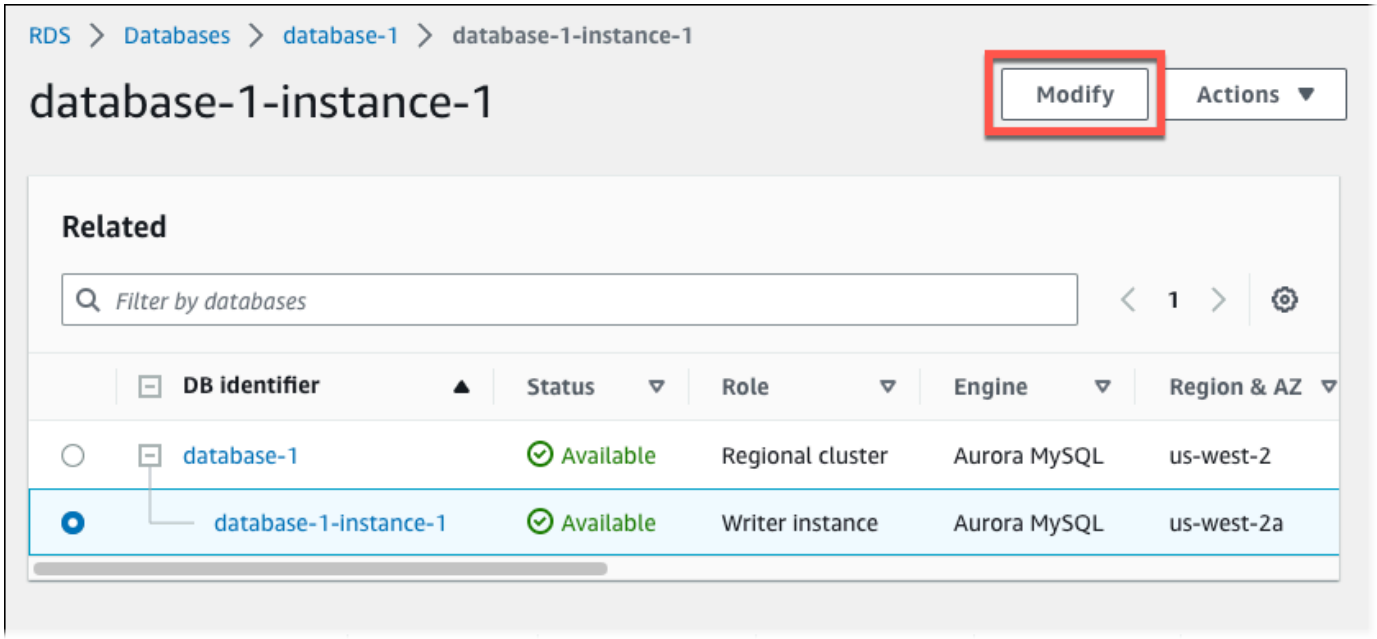
   > **Note**
   >
   > CA 証明書を更新するように変更した後に Babelfish クラスターを再起動します。

> **Important**
>
> 証明書の有効期限が切れた後に接続の問題が発生した場合は、コンソールで [すぐに適用] を指定するか、`--apply-immediately` を使用して AWS CLI オプションを指定します。デフォルトで、このオペレーションは次のメンテナンスウィンドウの間に実行するようスケジュールされています。
> デフォルト RDS CA と異なる cluster CA のオーバーライドを設定するには、modify-certificates CLI コマンドを使用します。

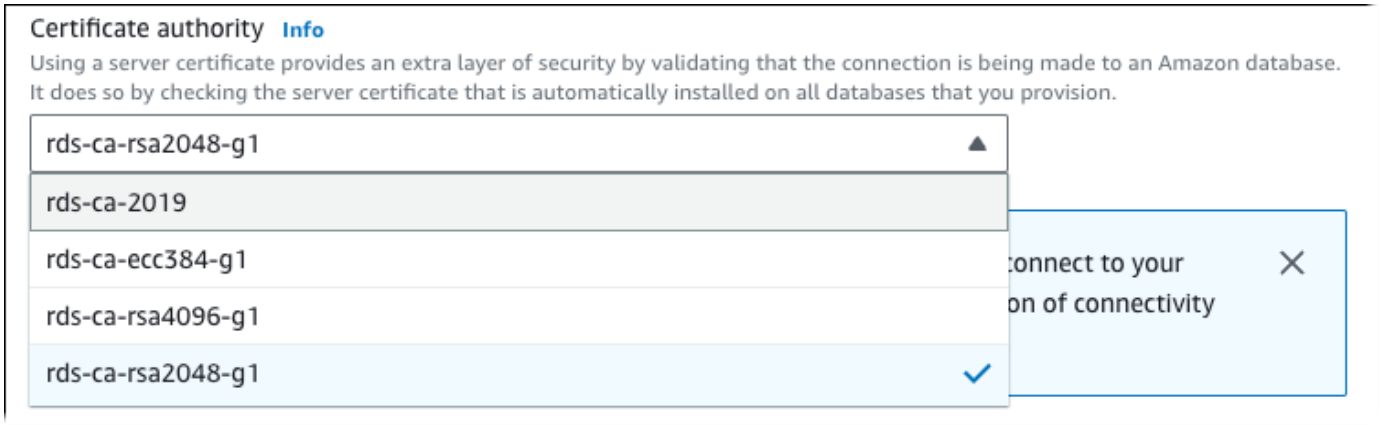
AWS Management Console または AWS CLI を使用して、DB インスタンスの CA 証明書を rds-ca-2019 から rds-ca-rsa2048-g1 に変更できます。

コンソール

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで、[データベース] を選択し、変更する DB インスタンスを選択します。
3. Modify を選択します。



4. [接続] セクションで、[rds-ca-rsa2048-g1] を選択します。



5. [Continue] を選択して、変更の概要を確認します。
6. 変更をすぐに反映させるには、[Apply immediately] を選択します。
7. 確認ページで、変更内容を確認します。変更内容が正しい場合は、[DB インスタンスを変更] を選択して変更を保存します。

> **Important**
>
> このオペレーションをスケジュールする場合は、必ずクライアント側の信頼ストアを事前に更新します。

または、[戻る] を選択して変更を編集するか、[キャンセル] を選択して変更をキャンセルします。

AWS CLI

AWS CLI を使用して DB インスタンスの CA を rds-ca-2019 から rds-ca-rsa2048-g1 に変更するには、modify-db-instance または modify-db-cluster コマンドを呼び出します。DB インスタンス識別子および `--ca-certificate-identifier` オプションを指定します。

`--apply-immediately` パラメータを使用して更新を直ちに適用します。デフォルトで、このオペレーションは次のメンテナンスウィンドウの間に実行するようスケジュールされています。

> **Important**
>
> このオペレーションをスケジュールする場合は、必ずクライアント側の信頼ストアを事前に更新します。

Example

次の例では、CA 証明書を rds-ca-rsa2048-g1 に設定することにより、mydbinstance を変更しています。

Linux、macOS、Unix の場合:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --ca-certificate-identifier rds-ca-rsa2048-g1
```

Windows の場合:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --ca-certificate-identifier rds-ca-rsa2048-g1
```

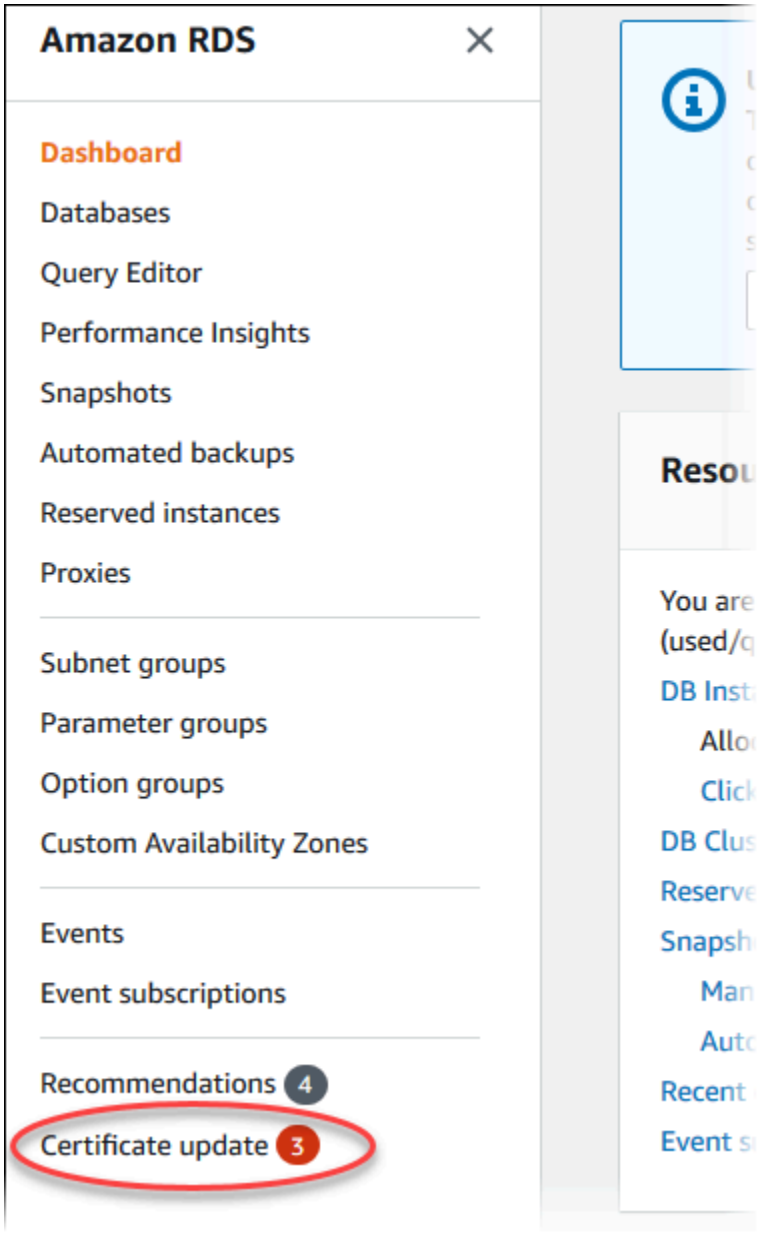
> **Note**
>
> インスタンスを再起動する必要がある場合、modify-db-instance CLI コマンドを使用して、--no-certificate-rotation-restart オプションを指定できます。

メンテナンスの適用による CA 証明書の更新

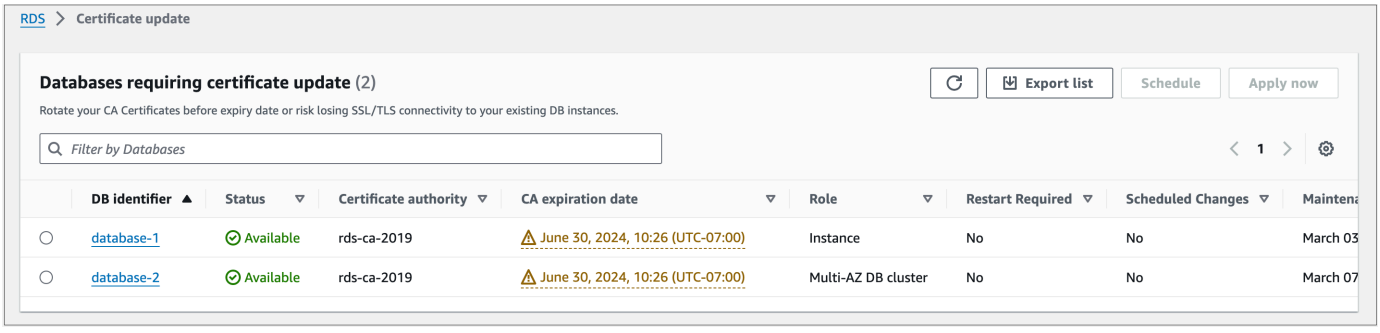
メンテナンスを適用して CA 証明書を更新するには、次のステップを実行します。

メンテナンスを適用して CA 証明書を更新するには

1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
2. ナビゲーションペインで [証明書の更新] を選択します。



[証明書の更新が必要なデータベース] ページが表示されます。



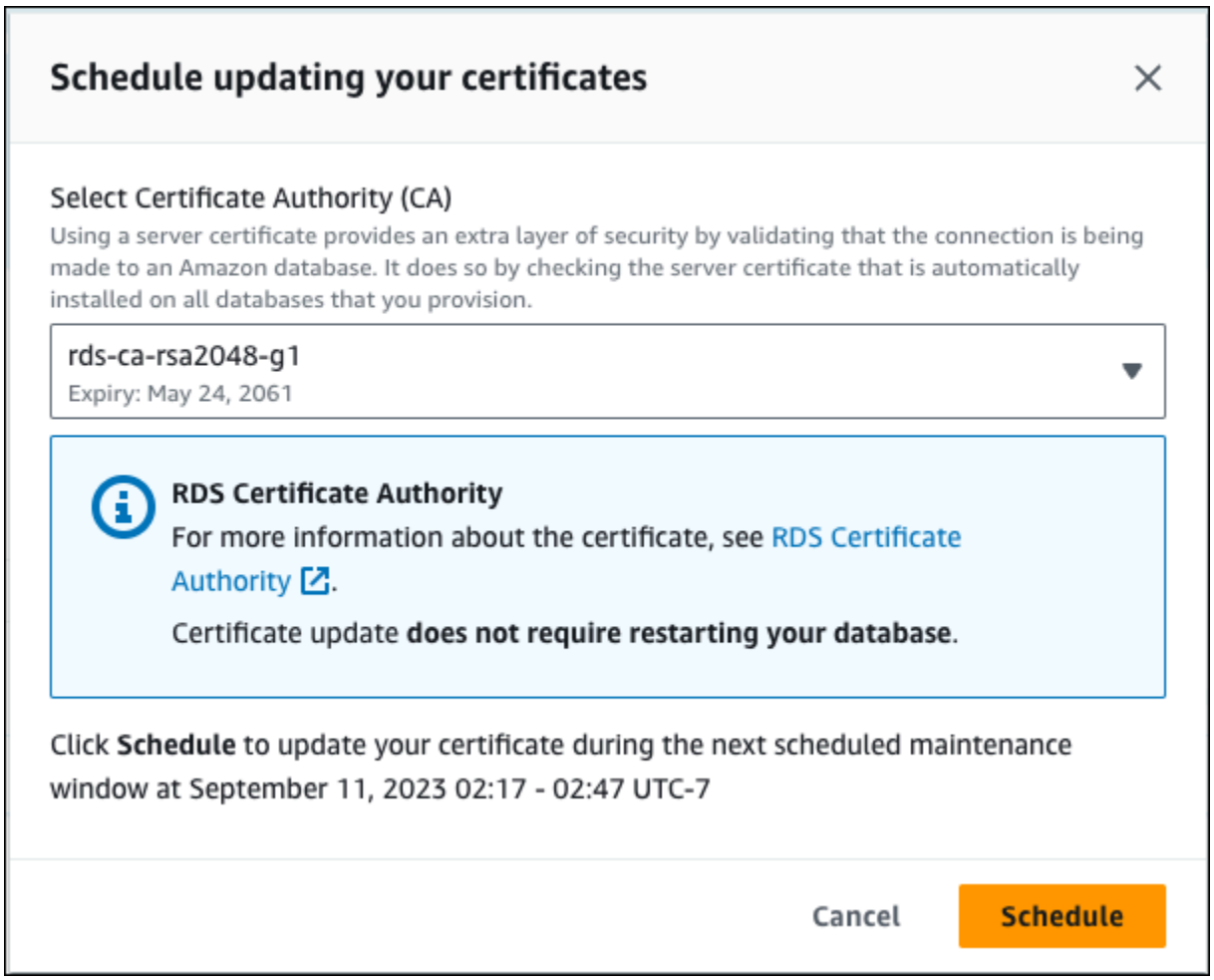
> **Note**
>
> このページには、現在の AWS リージョン リージョンの DB インスタンスのみが表示されます。複数の AWS リージョン にデータベースがある場合は、このページを AWS リージョン ごとにチェックし、古い SSL/TLS 証明書を使用しているすべての DB インスタンスを確認します。

3. 更新する DB インスタンスを選択します。

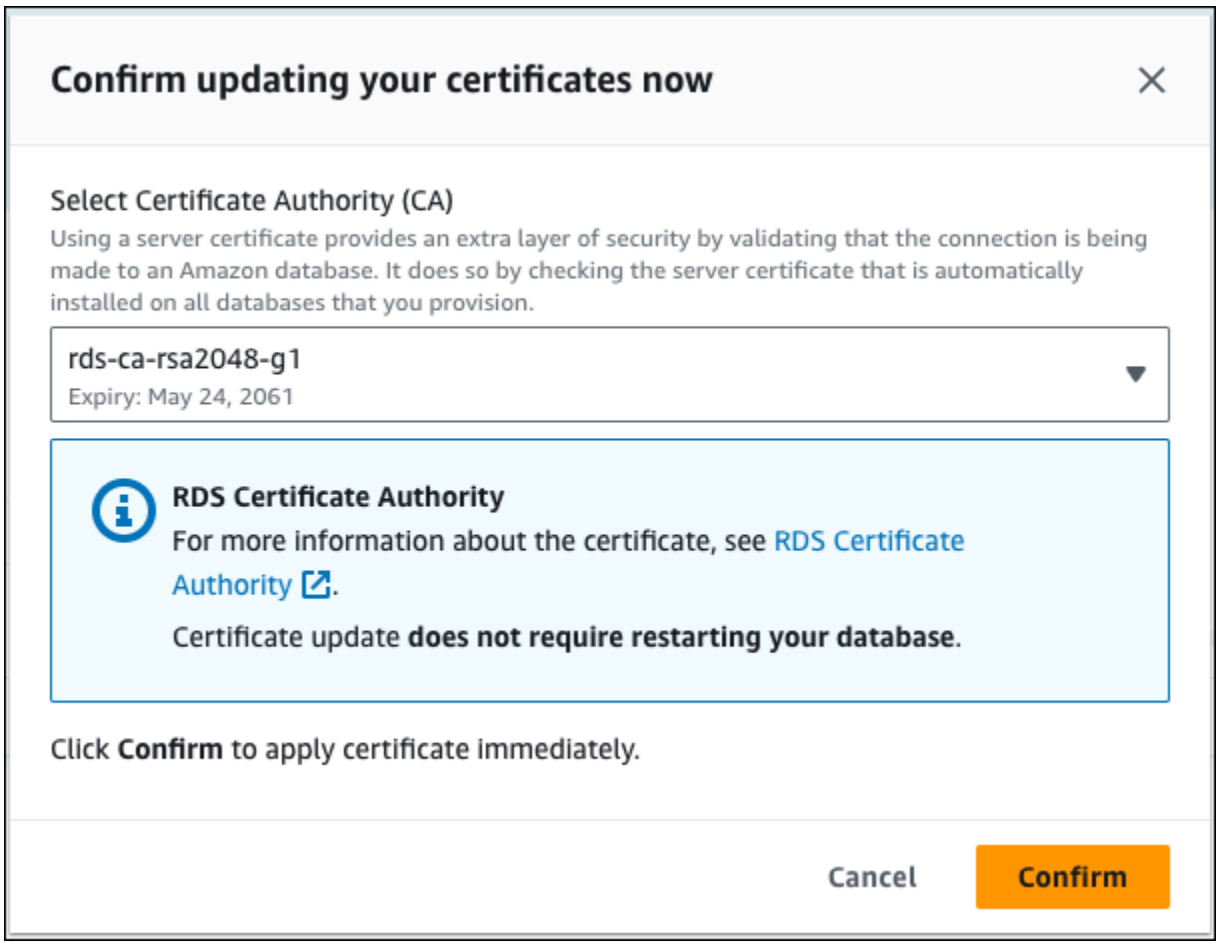
   [スケジュール] を選択すると、次のメンテナンスウィンドウでの証明書の更新をスケジュールできます。[今すぐ適用] を選択して、直ちに更新を適用します。

> **Important**
>
> 証明書の有効期限が切れた後に接続の問題が発生した場合は、[今すぐ適用] オプションを使用します。

4. a. [スケジュール] を選択すると、CA 証明書のローテーションの確認を求められます。このプロンプトには、アップデートのスケジュール期間も記載されています。



b. [今すぐ適用] を選択すると、CA 証明書のローテーションの確認を求められます。



> **Important**
>
> データベースでの CA 証明書の更新をスケジュールする前に、SSL/TLS とサーバー証明書を接続に使用するクライアントアプリケーションを更新します。これらの更新は、DB エンジンに固有です。これらのクライアントアプリケーションを更新したら、CA 証明書の更新を確認できます。

続行するには、チェックボックスをオンにし、[確認] を選択します。

5. 更新する DB インスタンスごとに、ステップ 3 と 4 を繰り返します。

サーバー証明書の自動ローテーション

CA がサーバー証明書の自動ローテーションをサポートしている場合、RDS は DB サーバー証明書のローテーションを自動的に処理します。RDS はこの自動ローテーションに同じルート CA を使用

するため、新しい CA バンドルをダウンロードする必要はありません。「認証局」を参照してください。

DB サーバー証明書のローテーションと有効期間は DB エンジンによって異なります。

- DB エンジンが再起動なしのローテーションをサポートしている場合、ユーザーによるアクションがなくても、RDS は DB サーバー証明書を自動的にローテーションします。RDS は、DB サーバー証明書の半減期になった時点で、希望するメンテナンス期間に DB サーバー証明書のローテーションを試みます。新しい DB サーバー証明書は、12 か月間有効です。
- DB エンジンが再起動なしのローテーションをサポートしていない場合、RDS は DB サーバー証明書の有効期限が切れる少なくとも 6 か月前にメンテナンスイベントについて通知します。新しい DB サーバー証明書は、36 か月間有効です。

describe-db-engine-versions コマンドを使用して、`SupportsCertificateRotationWithoutRestart` フラグを点検することで、再起動なしで DB エンジンバージョンが証明書のローテーションをサポートするかどうかを特定します。詳細については、「データベースに CA を設定する」を参照してください。

証明書を信頼ストアにインポートするためのサンプルスクリプト

次のサンプルシェルスクリプトでは、証明書バンドルを信頼ストア内にインポートします。

各サンプルシェルスクリプトでは、Java 開発キット (JDK) の一部である keytool が使用されます。JDK のインストールの詳細については、「JDK インストールガイド」を参照してください。

トピック

- Linux で証明書をインポートするためのサンプルスクリプト
- macOS で証明書をインポートするためのサンプルスクリプト

Linux で証明書をインポートするためのサンプルスクリプト

Linux オペレーティングシステムで、証明書バンドルを信頼ストアにインポートするサンプルシェルスクリプトを次に示します。

```
mydir=tmp/certs
if [ ! -e "${mydir}" ]
then
mkdir -p "${mydir}"
```

```
fi

truststore=${mydir}/rds-truststore.jks
storepassword=changeit

curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem" >
 ${mydir}/global-bundle.pem
awk 'split_after == 1 {n++;split_after=0} /-----END CERTIFICATE-----/ {split_after=1}
{print > "rds-ca-" n+1 ".pem"}' < ${mydir}/global-bundle.pem

for CERT in rds-ca-*; do
  alias=$(openssl x509 -noout -text -in $CERT | perl -ne 'next unless /Subject:/;
 s/.*(CN=|CN = )//; print')
  echo "Importing $alias"
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -keystore
 ${truststore} -noprompt
  rm $CERT
done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep Alias | cut
 -d " " -f3- | while read alias
do
   expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword} -alias
 "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print "$1\n"; }'`
   echo " Certificate ${alias} expires in '$expiry'"
done
```

macOS で証明書をインポートするためのサンプルスクリプト

macOS で証明書バンドルを信頼ストアにインポートするサンプルシェルスクリプトを次に示します。

```
mydir=tmp/certs
if [ ! -e "${mydir}" ]
then
mkdir -p "${mydir}"
fi
```

```
truststore=${mydir}/rds-truststore.jks
storepassword=changeit

curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem" >
 ${mydir}/global-bundle.pem
split -p "-----BEGIN CERTIFICATE-----" ${mydir}/global-bundle.pem rds-ca-

for CERT in rds-ca-*; do
  alias=$(openssl x509 -noout -text -in $CERT | perl -ne 'next unless /Subject:/;
 s/.*(CN=|CN = )//; print')
  echo "Importing $alias"
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -keystore
 ${truststore} -noprompt
  rm $CERT
done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep Alias | cut
 -d " " -f3- | while read alias
do
   expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword} -alias
 "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print "$1\n"; }'`
   echo " Certificate ${alias} expires in '$expiry'"
done
```

インターネットトラフィックのプライバシー

Amazon Aurora とオンプレミスのアプリケーション間、および同じ リージョン内の Amazon Aurora と他の リソース間では、接続が保護されています。

サービスとオンプレミスのクライアントおよびアプリケーションとの間のトラフィック

プライベートネットワークと AWS との間には 2 つの接続オプションがあります

- AWS Site-to-Site VPN 接続。詳細については、「AWS Site-to-Site VPN とは」を参照してください。

- AWS Direct Connect 接続。詳細については、「AWS Direct Connect とは」を参照してください。

AWS-published API オペレーションを使用することにより、ネットワークを通じて、Amazon Aurora へのアクセスを取得できます。クライアントは以下をサポートする必要があります。

- Transport Layer Security (TLS) TLS 1.2 および TLS 1.3 をお勧めします。
- DHE (Ephemeral Diffie-Hellman) や ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートです。これらのモードは、Java 7 以降など、最近のほとんどのシステムでサポートされています。

また、リクエストは、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

Amazon Aurora での Identity and Access Management

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御するために役立つ AWS のサービスです。IAM 管理者は、誰 (のサインイン) を認証して、Amazon RDS リソースの使用を承認 (許可) するかを管理します。IAM は、追加費用なしで使用できる AWS のサービスです。

トピック

対象者

AWS Identity and Access Management (IAM) の使用方法は、Amazon Aurora で行う作業によって異なります。

サービスユーザー - ジョブを実行するために Aurora サービスを使用する場合は、管理者が必要なアクセス許可と認証情報を用意します。作業を実行するためにさらに多くの Aurora 機能を使用するとき、追加のアクセス許可が必要になる場合があります。アクセスの管理方法を理解すると、管理者から適切なアクセス許可をリクエストするのに役に立ちます。Aurora の機能にアクセスできない場合は、「Amazon Aurora のアイデンティティおよびアクセスのトラブルシューティング」を参照してください。

サービス管理者 - 社内の Aurora リソースを担当している場合は、おそらく Aurora へのフルアクセスがあります。従業員がどの Aurora 機能とリソースアクセスする必要があるかを決定するのは管理者の仕事です。その後で、サービスユーザーのアクセス許可を変更するために、 管理者にリクエ

ストを送信する必要があります。このページの情報を点検して、IAM の基本概念を理解してください。お客様の会社で Aurora で IAM を利用する方法の詳細については、「Amazon Aurora と IAM の連携」を参照して ください。

管理者 - 管理者は、Aurora へのアクセスを管理するポリシーの作成方法の詳細について確認したい場合があります。IAM で使用できる Aurora アイデンティティベースのポリシーの例を表示するには、「Amazon Aurora のアイデンティティベースのポリシーの例」を参照して ください。

アイデンティティを使用した認証

認証とは、アイデンティティ認証情報を使用して AWSにサインインする方法です。ユーザーは、AWS アカウントのルートユーザー として、または IAM ロールを引き受けることによって、認証済み (AWSにサインイン済み) である必要があります。

ID ソースから提供された認証情報を使用して、フェデレーティッドアイデンティティとして AWSにサインインできます。 AWS IAM Identity Centerフェデレーションアイデンティティの例としては、(IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook の認証情報などがあります。フェデレーションアイデンティティとしてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーションが設定されています。フェデレーションを使用して AWSにアクセスする場合、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、AWS Management Console または AWSアクセスポータルにサインインできます。AWS へのサインインの詳細については、「AWS サインイン ユーザーガイド」の「AWS アカウント にサインインする方法」を参照してください。

プログラムを使用して AWSにアクセスする場合、AWS は Software Development Kit (SDK) とコマンドラインインターフェイス (CLI) を提供し、認証情報を使用してリクエストに暗号で署名します。AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。リクエストに署名する推奨方法の使用については、「IAM ユーザーガイド」の「AWS API リクエストの署名」を参照してください。

使用する認証方法を問わず、セキュリティ情報の提供を追加でリクエストされる場合もあります。例えば、AWS は、アカウントのセキュリティを強化するために多要素認証 (MFA) を使用することをお勧めします。詳細については、「AWS IAM Identity Center ユーザーガイド」の「多要素認証」および「IAM ユーザーガイド」の「AWS での多要素認証 (MFA) の使用」を参照してください。

AWS アカウントのルートユーザー

AWS アカウント を作成する場合は、このアカウントのすべての AWS のサービスとリソースに対して完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカ

ウント ルートユーザー と呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることによってアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報を保護し、それらを使用してルートユーザーのみが実行できるタスクを実行してください。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

フェデレーション ID

ベストプラクティスとして、管理者アクセスを必要とするユーザーを含む人間のユーザーに対し、ID プロバイダーとのフェデレーションを使用して、一時的な認証情報の使用により、AWS のサービスにアクセスすることを要求します。

フェデレーティッド ID は、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダー、AWS Directory Service、Identity Center ディレクトリのユーザーか、または ID ソースから提供された認証情報を使用して AWS のサービスにアクセスするユーザーです。フェデレーティッド ID が AWS アカウントにアクセスすると、ロールが継承され、ロールは一時的な認証情報を提供します。

アクセスを一元管理する場合は、AWS IAM Identity Center を使用することをお勧めします。IAM Identity Center でユーザーとグループを作成するか、すべての AWS アカウントとアプリケーションで使用するために、独自の ID ソースで一連のユーザーとグループに接続して同期することもできます。IAM Identity Center の詳細については、「AWS IAM Identity Center ユーザーガイド」の「IAM Identity Center とは？」を参照してください。

IAM ユーザーとグループ

IAM ユーザー は、1 人のユーザーまたは 1 つのアプリケーションに対して特定の許可を持つ AWS アカウント内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、IAM ユーザーガイドの「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループ は、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。

例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する許可を与えることができます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時的な認証情報が提供されます。詳細については、IAM ユーザーガイドの IAM ユーザーの作成が適している場合 (ロールではなく) を参照してください。

IAM データベース認証を使用して、DB クラスターを認証できます。

IAM データベース認証は、Aurora で使用できます。IAM を使用した DB クラスターの認証の詳細については、「 の IAM データベース認証」を参照してください。

IAM ロール

IAM ロール は、特定の許可を持つ、AWS アカウント 内のアイデンティティです。これはユーザーに似ていますが、特定のユーザーに関連付けられていません。ロールを切り替えることによって、AWS Management Console で IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、AWS CLI または AWSAPI オペレーションを呼び出すか、カスタム URL を使用します。ロールを使用する方法の詳細については、「IAM ユーザーガイド」の「IAM ロールの使用」を参照してください。

一時的な認証情報を持った IAM ロールは、以下の状況で役立ちます。

- 一時的なユーザーアクセス許可 - ユーザーは、特定のタスクのための複数の異なるアクセス許可を一時的に受け取るために、IAM ロールを引き受けることができます。
- フェデレーションユーザーアクセス – フェデレーションアイデンティティに権限を割り当てるには、ロールを作成してそのロールの権限を定義します。フェデレーションアイデンティティが認証されると、そのアイデンティティはロールに関連付けられ、ロールで定義されている権限が付与されます。フェデレーションの詳細については、「IAM ユーザーガイド」の「サードパーティーアイデンティティプロバイダー向けロールの作成」を参照してください。IAM アイデンティティセンターを使用する場合、権限セットを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。アクセス許可セットの詳細については、「AWS IAM Identity Center ユーザーガイド」の「アクセス許可セット」を参照してください。
- クロスアカウントアクセス - IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の AWS のサービ

スでは、(ロールをプロキシとして使用する代わりに) リソースにポリシーを直接アタッチできます。クロスアカウントアクセスにおけるロールとリソースベースのポリシーの違いについては、「IAM ユーザーガイド」の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

- クロスサービスアクセス - 一部の AWS のサービスでは、他の AWS のサービスの機能を使用します。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプリケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスでは、呼び出し元プリンシパルの許可、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。
  - 転送アクセスセッション – IAM ユーザーまたはロールを使用して AWS でアクションを実行するユーザーは、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、AWS のサービスを呼び出すプリンシパルの権限を、AWS のサービスのリクエストと合わせて使用し、ダウンストリームのサービスに対してリクエストを行います。FAS リクエストは、サービスが、完了するために他の AWS のサービス またはリソースとのやりとりを必要とするリクエストを受け取ったときにのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。
  - サービスロール - サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細については、「IAM ユーザーガイド」の「AWS のサービス にアクセス許可を委任するロールの作成」を参照してください。
  - サービスリンクロール - サービスリンクロールは、AWS のサービス にリンクされたサービスロールの一種です。サービスがロールを引き受け、ユーザーに代わってアクションを実行できるようになります。サービスリンクロールは、AWS アカウント に表示され、サービスによって所有されます。IAM 管理者は、サービスリンクロールの許可を表示できますが、編集することはできません。
- Amazon EC2 で実行されているアプリケーション - EC2 インスタンスで実行され、AWS CLI または AWSAPI 要求を行っているアプリケーションの一時的な認証情報を管理するには、IAM ロールを使用できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。AWS ロールを EC2 インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスに添付されたインスタンスプロファイルを作成します。インスタンスプロファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を取得できます。詳細については、IAM ユーザーガイドの「IAM ロールを使用して、Amazon EC2 インスタンスで実行されるアプリケーションにアクセス許可を付与する」を参照してください。

IAM ロールを使用するべきかどうかについては、IAM ユーザーガイドの「IAM ロールの作成が適している場合 (ユーザーではなく)」を参照してください。

ポリシーを使用したアクセスの管理

AWS でのアクセスは、ポリシーを作成し、それらを IAM アイデンティティまたは AWS リソースに添付することで制御できます。ポリシーは AWS のオブジェクトであり、ID やリソースに関連付けられて、これらのアクセス許可を定義します。AWS は、エンティティ (ルートユーザー、ユーザー、または IAM ロール) によってリクエストが行われると、それらのポリシーを評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。大半のポリシーは JSON ドキュメントとして AWSに保存されます。JSON ポリシードキュメントの構造と内容の詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は、ポリシーを使用して、AWS リソースへのアクセス権を持つユーザーと、これらのリソースに対して実行できるアクションを指定できます。すべての IAM エンティティ (アクセス許可セットまたはロール) は、アクセス許可のない状態からスタートします。言い換えると、デフォルト設定では、ユーザーは何もできず、自分のパスワードを変更することすらできません。何かを実行する許可をユーザーに付与するには、管理者がユーザーに許可ポリシーをアタッチする必要があります。また、管理者は、必要な許可があるグループにユーザーを追加できます。管理者がグループに許可を付与すると、そのグループ内のすべてのユーザーにこれらの許可が付与されます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの許可を定義します。例えば、`iam:GetRole` アクションを許可するポリシーがあるとします。このポリシーがあるユーザーは、AWS Management Console、AWS CLI、または AWSAPI からロールの情報を取得できます。

アイデンティティベースポリシー

ID ベースのポリシーは、 アクセス許可セットやロールなどの ID にアタッチできる JSON アクセス許可ポリシードキュメントです。これらのポリシーは、アイデンティティが実行できるアクション、リソース、および条件を制御します。アイデンティティベースのポリシーを作成する方法については、IAM ユーザーガイドの IAM ポリシーの作成を参照してください。

アイデンティティベースのポリシーは、さらに インラインポリシー または マネージドポリシー に分類できます。インラインポリシーは、単一のアクセス許可セットまたはロールに直接埋め込まれます。マネージドポリシーは、AWS アカウント内の複数のアクセス許可セットおよびロールにアタッチできるスタンドアロンポリシーです。マネージドポリシーには、AWS マネージドポリシーとカスタマー管理ポリシーがあります。マネージドポリシーまたはインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「マネージドポリシーとインラインポリシーの比較」を参照してください。

Amazon Auroraに固有の AWS マネージドポリシーの詳細については、「Amazon RDS の AWS マネージドポリシー」を参照してください。

他のポリシータイプ

AWS では、他の一般的ではないポリシータイプをサポートしています。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 - アクセス許可の境界は、ID ベースのポリシーによって IAM エンティティ (アクセス許可セットまたはロール) に付与できるアクセス許可の上限を設定する高度な機能です。エンティティに許可の境界を設定できます。結果として許可される範囲は、エンティティのアイデンティティベースポリシーとその許可の境界の共通部分になります。`Principal` フィールドでアクセス許可セットまたはロールを指定するリソースベースのポリシーは、アクセス許可の境界によって制限されません。これらのポリシーのいずれかを明示的に拒否した場合、許可は無効になります。許可の境界の詳細については、IAM ユーザーガイドの「IAM エンティティの許可の境界」を参照してください。
- サービスコントロールポリシー (SCP) – SCP は、AWS Organizations で組織や組織単位 (OU) に最大アクセス許可を指定する JSON ポリシーです。AWS Organizationsは、お客様のビジネスが所有する複数の AWS アカウントをグループ化し、一元的に管理するサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウントに適用できます。SCP はメンバーアカウントのエンティティに対するアクセス許可を制限します (各 AWS アカウントのルートユーザーなど)。Organizations と SCP の詳細については、 AWS Organizations ユーザーガイド の「SCP の仕組み」を参照してください。
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーティッドユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてのセッションのアクセス許可は、アクセス許可セットまたはロールの ID ベースのポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、許可は無効になります。詳細については、 IAM ユーザーガイド の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。複数のポリシータイプが関連するとき、リクエストを許可するかどうかを AWS が決定する方法の詳細については、IAM ユーザーガイドのポリシーの評価ロジックを参照してください。

Amazon Aurora と IAM の連携

IAM を使用して、Amazon Aurora へのアクセスを管理するには、Aurora で使用できる IAM の機能を理解しておく必要があります。

Amazon Aurora で使用できる IAM の機能

IAM 機能	Amazon Aurora のサポート
アイデンティティベースのポリシー	Yes
リソースベースのポリシー	いいえ
ポリシーアクション	Yes
ポリシーリソース	はい
ポリシー条件キー (サービス固有)	はい
ACL	No
属性ベースのアクセスコントロール (ABAC) (ポリシーのタグ)	Yes
一時的な認証情報	Yes
転送アクセスセッション	Yes
サービスロール	あり
サービスリンクロール	Yes

Amazon Aurora や AWS の他のサービスと IAM との連携の概要については、IAM ユーザーガイドの「IAM と連携する AWS のサービス」を参照してください。

トピック

- Aurora アイデンティティベースのポリシー
- Aurora 内のリソースベースのポリシー
- Aurora のポリシーアクション
- Aurora のポリシーリソース

- Aurora のポリシー条件キー
- Aurora のアクセスコントロールリスト (ACL)
- Aurora タグを使ったポリシーにおける属性ベースのアクセスコントロール (ABAC)
- Aurora での一時的な認証情報の使用
- Aurora のフォワードアクセスセッション
- Aurora のサービスロール
- Aurora のサービスリンクロール

Aurora アイデンティティベースのポリシー

アイデンティティベースポリシーをサポートする	Yes

アイデンティティベースポリシーは、IAM ユーザー、ユーザーのグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、「IAM ユーザーガイド」の「IAM ポリシーの作成」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。プリンシパルは、それが添付されているユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシーで使用できるすべての要素について学ぶには、IAM ユーザーガイドの「IAM JSON ポリシーの要素のリファレンス」を参照してください。

Aurora アイデンティティベースのポリシーの例

Aurora アイデンティティベースのポリシーの例を表示するには、「Amazon Aurora のアイデンティティベースのポリシーの例」を参照してください。

Aurora 内のリソースベースのポリシー

リソースベースのポリシーのサポート	なし

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。ポリシーが添付されているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または AWS のサービスを含めることができます。

クロスアカウントアクセスを有効にするには、全体のアカウント、または別のアカウントの IAM エンティティを、リソースベースのポリシーのプリンシパルとして指定します。リソースベースのポリシーにクロスアカウントのプリンシパルを追加しても、信頼関係は半分しか確立されない点に注意してください。プリンシパルとリソースが異なる AWS アカウントにある場合、信頼できるアカウントの IAM 管理者は、リソースにアクセスするための許可をプリンシパルエンティティ (ユーザーまたはロール) に付与する必要もあります。IAM 管理者は、アイデンティティベースのポリシーをエンティティにアタッチすることで権限を付与します。ただし、リソースベースのポリシーで、同じアカウントのプリンシパルへのアクセス権が付与されている場合は、アイデンティティベースのポリシーを追加する必要はありません。詳細については、IAM ユーザーガイドの「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

Aurora のポリシーアクション

ポリシーアクションに対するサポート	Yes

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

JSON ポリシーの `Action`要素には、ポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションの名前は通常、関連する AWSAPI オペレーションと同じです。一致する API オペレーションのない 許可のみのアクション など、いくつかの例外があります。また、ポリシーに複数アクションが必要なオペレーションもあります。これらの追加アクションは、 依存アクション と呼ばれます。

このアクションは、関連付けられたオペレーションを実行するためのアクセス許可を付与するポリシーで使用されます。

Aurora のポリシーアクションは、アクションの前にプレフィックス rds: を使用します。例えば、Amazon RDS DescribeDBInstances API オペレーションを使用して DB インスタンスを指定するアクセス許可を付与するには、ポリシーに rds:DescribeDBInstances アクションを含めます。ポリシーステートメントには、Action または NotAction 要素を含める必要があります。Aurora は、このサービスで実行できるタスクを記述する独自のアクションのセットを定義します。

単一のステートメントに複数の アクションを指定するには、次のようにコンマで区切ります。

```
"Action": [
      "rds:action1",
      "rds:action2"
```

ワイルドカード *を使用して複数のアクションを指定することができます。例えば、Describe という単語で始まるすべてのアクションを指定するには、次のアクションを含めます。

```
"Action": "rds:Describe*"
```

Aurora アクションのリストを確認するには、サービス認証リファレンスの「Amazon RDS で定義されるアクション」を参照してください。

Aurora のポリシーリソース

ポリシーリソースに対するサポート	あり

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Resource JSON ポリシーの要素は、オブジェクトあるいはアクションが適用されるオブジェクトを指定します。ステートメントには、Resource または NotResource要素を含める必要があります。ベストプラクティスとしては、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、 リソースレベルの許可 と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルのアクセス許可をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

DB インスタンスリソースには、次の Amazon リソースネーム (ARN) があります。

```
arn:${Partition}:rds:${Region}:${Account}:{ResourceType}/${Resource}
```

ARN の形式の詳細については、「Amazon リソースネーム ARNと AWS のサービスの名前空間」を参照してください。

例えば、ステートメントで dbtest DB インスタンスを指定するには、次の ARN を使用します。

```
"Resource": "arn:aws:rds:us-west-2:123456789012:db:dbtest"
```

特定のアカウントに属するすべての DB インスタンスを指定するには、ワイルドカード (*) を使用します。

```
"Resource": "arn:aws:rds:us-east-1:123456789012:db:*"
```

リソースの作成など、一部の RDS API オペレーションは、特定のリソースで実行できません。このような場合は、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

Amazon RDS API オペレーションの多くが複数のリソースと関連します。例えば、CreateDBInstance では、DB インスタンスが作成されます。DB インスタンス作成時に特定のセキュリティグループおよびパラメータグループを使用するように ユーザーに義務付けることができます。複数リソースを単一ステートメントで指定するには、ARN をカンマで区切ります。

```
"Resource": [
      "resource1",
      "resource2"
```

Aurora リソースのタイプとその ARN のリストを確認するには、サービス認証リファレンス の「Amazon RDS で定義されるリソース」を参照してください。どのアクションで各リソースの ARN を指定できるかについては、「Amazon RDS で定義されるアクション」を参照してください。

Aurora のポリシー条件キー

サービス固有のポリシー条件キーのサポート	はい

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Condition 要素 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。イコールや未満などの 条件演算子 を使用して条件式を作成することで、ポリシーの条件とリクエスト内の値を一致させることができます。

1 つのステートメントに複数の Condition要素を指定する場合、または 1 つの Condition要素に複数のキーを指定する場合、AWS では AND論理演算子を使用してそれらを評価します。単一の条件キーに複数の値を指定する場合、AWS では OR論理演算子を使用して条件を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。詳細については、 IAM ユーザーガイド の「IAM ポリシーの要素: 変数およびタグ」を参照してください。

AWS はグローバル条件キーとサービス固有の条件キーをサポートしています。すべての AWS グローバル条件キーを確認するには、IAM ユーザーガイドの「AWS グローバル条件コンテキストキー」を参照してください。

Aurora は独自の条件キーを定義し、一部のグローバル条件キーの使用をサポートしています。すべての AWS グローバル条件キーを確認するには、IAM ユーザーガイドの「AWS グローバル条件コンテキストキー」を参照してください。

すべての RDS API オペレーションは、aws:RequestedRegion 条件キーをサポートします。

Aurora の条件キーのリストを確認するには、サービス認証リファレンスの「Amazon RDS の条件キー」を参照してください。どのアクションおよびリソースと条件キーを使用できるかについては、「Amazon RDS で定義されるアクション」を参照してください。

Aurora のアクセスコントロールリスト (ACL)

アクセスコントロールリスト (ACL) をサポート	No

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするための許可を持つかをコントロールします。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

Aurora タグを使ったポリシーにおける属性ベースのアクセスコントロール (ABAC)

ポリシーにおける属性ベースのアクセスコントロール (ABAC) タグ	Yes

属性ベースのアクセス制御 (ABAC) は、属性に基づいてアクセス許可を定義するアクセス許可戦略です。AWS では、属性は タグ と呼ばれます。タグは、IAM エンティティ (ユーザーまたはロール)、および多数の AWSリソースにアタッチできます。エンティティとリソースのタグ付けは、ABAC の最初の手順です。次に、プリンシパルのタグがアクセスを試行するリソースのタグと一致したときにオペレーションを許可するよう、ABAC ポリシーを設計します。

ABAC は、急成長する環境やポリシー管理が煩雑になる状況で役立ちます。

タグに基づいてアクセスを管理するには、`aws:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys`の条件キーを使用して、ポリシーの条件要素 でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値は Yes です。サービスが一部のリソースタイプに対してのみ 3 つの条件キーすべてをサポートする場合、値は Partial です。

ABAC の詳細については、IAM ユーザーガイドの「ABAC とは? 」を参照してください。ABAC をセットアップするステップを説明するチュートリアルについては、IAM ユーザーガイドの「属性に基づくアクセスコントロール (ABAC) を使用する」を参照してください。

Aurora リソースのタグ付けの詳細については、「条件の指定: カスタムタグの使用」を参照してください。リソースのタグに基づいてリソースへのアクセスを制限するためのアイデンティティベースのポリシーの例を表示するには、「2 つの異なる値を持つタグが付いたリソースに対するアクションにアクセス許可を付与する」を参照してください。

Aurora での一時的な認証情報の使用

一時的な認証情報のサポート	Yes

AWS のサービス には、一時的な認証情報を使用してサインインしても機能しないものがあります。一時的な認証情報を利用できる AWS のサービスを含めた詳細情報については、「IAM ユーザーガイド」の「IAM と連携する AWS のサービス」を参照してください。

ユーザー名とパスワード以外の方法で AWS Management Consoleにサインインする場合は、一時認証情報を使用していることになります。例えば、会社の Single Sign-On (SSO) リンクを使用して AWS にアクセスすると、そのプロセスは自動的に一時認証情報を作成します。また、ユーザーとしてコンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えに関する詳細については、 IAM ユーザーガイド の「ロールへの切り替え (コンソール)」を参照してください。

一時認証情報は、AWS CLI または AWSAPI を使用して手動で作成できます。作成後、一時認証情報を使用して AWSにアクセスできるようになります。AWS は、長期的なアクセスキーを使用する代わりに、一時認証情報を動的に生成することをお勧めします。詳細については、「IAM の一時的セキュリティ認証情報」を参照してください。

Aurora のフォワードアクセスセッション

転送アクセスセッションをサポート	Yes

IAM ユーザーまたはロールを使用して AWS でアクションを実行するユーザーは、プリンシパルとみなされます。一部のサービスを使用する際に、アクションを実行してから、別のサービスの別のアクションを開始することがあります。FAS は、AWS のサービスを呼び出すプリンシパルの権限を、AWS のサービスのリクエストと合わせて使用し、ダウンストリームのサービスに対してリクエストを行います。FAS リクエストは、サービスが、完了するために他の AWS のサービス またはリソースとのやりとりを必要とするリクエストを受け取ったときにのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

Aurora のサービスロール

サービスロールに対するサポート	あり

サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。詳細につい

ては、「IAM ユーザーガイド」の「AWS のサービス にアクセス許可を委任するロールの作成」を参照してください。

> **Warning**
>
> サービスロールの許可を変更すると、Aurora のサービスロール の機能が破損する可能性があります。Aurora が指示する場合以外は、サービスロールを編集しないでください。

Aurora のサービスリンクロール

サービスリンクロールのサポート	Yes

サービスリンクロールは、AWS のサービス にリンクされているサービスロールの一種です。サービスがロールを引き受け、ユーザーに代わってアクションを実行できるようになります。サービスリンクロールは、AWS アカウント に表示され、サービスによって所有されます。IAM 管理者は、サービスリンクロールの許可を表示できますが、編集することはできません。

Aurora サービスにリンクされたロールの使用の詳細については、「Amazon Aurora のサービスにリンクされたロールの使用」を参照してください。

Amazon Aurora のアイデンティティベースのポリシーの例

デフォルトでは、アクセス許可セットとロールには、Aurora リソースを作成または変更するアクセス許可はありません。AWS Management Console、AWS CLI、または AWS API を使用してタスクを実行することもできません。管理者は、指定されたリソースに対して特定の API オペレーションを実行するために必要なアクセス許可をアクセス許可セットとロールに付与する IAM ポリシーを作成する必要があります。続いて、管理者は、それらのアクセス許可を必要とするアクセス許可セットまたはロールに、そのポリシーをアタッチします。

これらの JSON ポリシードキュメント例を使用して IAM のアイデンティティベースのポリシーを作成する方法については、『IAM ユーザーガイド』の「JSON タブでのポリシーの作成」を参照してください。

トピック

- ポリシーのベストプラクティス
- Aurora コンソールの使用

- 自分の権限の表示をユーザーに許可する
- AWS アカウントでの DB インスタンスの作成をユーザーに許可する
- コンソールの使用に必要なアクセス許可
- RDS リソースに対する Describe アクションの実行をユーザーに許可する
- 指定した DB パラメータグループとサブネットグループを使用する DB インスタンスの作成をユーザーに許可する
- 2 つの異なる値を持つタグが付いたリソースに対するアクションにアクセス許可を付与する
- ユーザーによる DB インスタンスの削除を禁止する
- リソースへのすべてのアクセスを拒否する
- ポリシー例: 条件キーの使用
- 条件の指定: カスタムタグの使用

ポリシーのベストプラクティス

ID ベースのポリシーは、ユーザーのアカウント内で誰かが Amazon RDS リソースを作成、アクセス、または削除できるどうかを決定します。これらのアクションを実行すると、AWS アカウント に料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS マネージドポリシーを使用して開始し、最小特権の権限に移行する – ユーザーとワークロードへの権限の付与を開始するには、多くの一般的なユースケースのために権限を付与する AWS マネージドポリシーを使用します。これらは AWS アカウントで使用できます。ユースケースに応じた AWS カスタマーマネージドポリシーを定義することで、権限をさらに減らすことをお勧めします。詳細については、『IAM ユーザーガイド』の「AWS マネージドポリシー」または「AWS ジョブ機能の管理ポリシー」を参照してください。
- 最小特権を適用する – IAM ポリシーで権限を設定するときは、タスクの実行に必要な権限のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権権限とも呼ばれています。IAM を使用して権限を適用する方法の詳細については、『IAM ユーザーガイド』の「IAM でのポリシーと権限」を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。また、AWS CloudFormation などの特定の AWS のサービスを介して使用する場合、条件を使ってサービスアクションへのアクセス権を付与することもできます。詳細については、『IAM ユーザーガイド』の [IAM JSON policy elements: Condition] (IAM JSON ポリシー要素：条件) を参照してください。

- IAM Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM Access Analyzer は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの作成をサポートします。詳細については、『IAM ユーザーガイド』の「IAM Access Analyzer ポリシーの検証」を参照してください。
- 多要素認証 (MFA) を要求する – AWS アカウント で IAM ユーザーまたはルートユーザーを要求するシナリオがある場合は、セキュリティを強化するために MFA をオンにします。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、『IAM ユーザーガイド』の「MFA 保護 API アクセスの設定」を参照してください。

IAM でのベストプラクティスの詳細については、『IAM ユーザーガイド』の「IAM でのセキュリティのベストプラクティス」を参照してください。

Aurora コンソールの使用

Amazon Aurora コンソールにアクセスするには、一連の最小限のアクセス許可が必要です。これらのアクセス許可により、AWS アカウント の Amazon Aurora リソースの詳細をリストおよび表示できるようにする必要があります。最小限必要なアクセス許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) ではコンソールが意図したとおりに機能しません。

AWS CLI または AWS API のみを呼び出すユーザーには、最小限のコンソール権限を付与する必要はありません。代わりに、実行しようとしている API オペレーションに一致するアクションのみへのアクセスが許可されます。

これらのエンティティが Aurora コンソールを引き続き使用できるように、エンティティに次の AWS 管理ポリシーもアタッチします。

```
AmazonRDSReadOnlyAccess
```

詳細については、「IAM ユーザーガイド」の「ユーザーへのアクセス許可の追加」を参照してください。

自分の権限の表示をユーザーに許可する

この例では、ユーザーアイデンティティにアタッチされたインラインおよびマネージドポリシーの表示を IAM ユーザーに許可するポリシーの作成方法を示します。このポリシーには、コンソールで、

または AWS CLI か AWS API を使用してプログラム的に、このアクションを完了するアクセス許可が含まれています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS アカウントでの DB インスタンスの作成をユーザーに許可する

以下は、123456789012 アカウントで ID が AWS のユーザーが DB インスタンスを作成できるようにするポリシーの例です。ポリシーは、test で始める新しい DB インスタンスの名前である必要があります。また、新しい DB インスタンスは、MySQL データベースエンジンと DB インスタンスの db.t2.micro クラスを使用する必要があります。さらに、新しい DB インスタンスでは、オプショ

ングループと default で始まる DB パラメータグループ、および default サブネットグループを使用する必要があります。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowCreateDBInstanceOnly",
         "Effect": "Allow",
         "Action": [
            "rds:CreateDBInstance"
         ],
         "Resource": [
            "arn:aws:rds:*:123456789012:db:test*",
            "arn:aws:rds:*:123456789012:og:default*",
            "arn:aws:rds:*:123456789012:pg:default*",
            "arn:aws:rds:*:123456789012:subgrp:default"
         ],
         "Condition": {
            "StringEquals": {
               "rds:DatabaseEngine": "mysql",
               "rds:DatabaseClass": "db.t2.micro"
            }
         }
      }
   ]
}
```

ポリシーには、 ユーザー用の以下のアクセス許可を指定する単一のステートメントが含まれます。

- ポリシーを使用すると、ユーザーは CreateDBInstance API オペレーションを使用して DB インスタンスを作成できます (これは create-db-instance AWS CLI コマンドと AWS Management Console にも適用されます)。
- Resource 要素では、ユーザーがリソースでアクションを実行できることを指定できます。Amazon Resources Name (ARN) を使用してリソースを指定します。この ARN には、リソースが属しているサービスの名前 (rds)、AWS リージョン (* はこの例のリージョンを示します)、AWS アカウント番号 (123456789012 はこの例のアカウント番号です)、およびリソースのタイプが含まれます。ARN の作成の詳細については、「Amazon RDS の Amazon リソースネーム (ARN) の使用」を参照してください。

  例の Resource 要素は、ユーザーのリソースで、以下のポリシーの制約を指定します。

  - 新しい DB インスタンスの DB インスタンス識別子は、test で始まる必要があります (例: testCustomerData1、test-region2-data)。
  - 新しい DB インスタンスのオプショングループは、default で始まる必要があります。
  - 新しい DB インスタンスの DB パラメータグループは、default で始まる必要があります。
  - 新しい DB インスタンスのサブネットグループは、default サブネットグループである必要があります。
- Condition 要素は、DB エンジンが MySQL で、DB インスタンスクラスが db.t2.micro である必要があることを指定します。Condition 要素は、ポリシーが有効になる条件を指定します。Condition 要素を使用して、アクセス許可または制約を追加できます。条件を指定する方法については、「Aurora のポリシー条件キー」を参照してください。この例では、rds:DatabaseEngine および rds:DatabaseClass を条件として指定します。rds:DatabaseEngine の有効な条件値については、CreateDBInstance の Engine パラメータのリストを参照してください。rds:DatabaseClass の有効な条件値については、「DB インスタンスクラスでサポートされている DB エンジン」を参照してください。

アイデンティティベースのポリシーでアクセス権限を得るプリンシパルを指定していないため、ポリシーでは Principal 要素を指定していません。ユーザーにポリシーをアタッチすると、そのユーザーが暗黙のプリンシパルになります。IAM ロールにアクセス権限ポリシーをアタッチすると、ロールの信頼ポリシーで識別されたプリンシパルがアクセス権限を得ることになります。

Aurora アクションのリストを確認するには、サービス認証リファレンスの「Amazon RDS で定義されるアクション」を参照してください。

コンソールの使用に必要なアクセス許可

コンソールを使用するユーザーには、最小限のアクセス許可のセットが必要です。これらのアクセス許可により、ユーザーは AWS アカウントの Amazon Aurora リソースを記述し、Amazon EC2 セキュリティやネットワーク情報など、その他の関連情報を提供できます。

これらの最小限必要なアクセス権限よりも制限された IAM ポリシーを作成している場合、その IAM ポリシーを使用するユーザーに対してコンソールは意図したとおりには機能しません。「AmazonRDSReadOnlyAccess」で説明されているとおり、ユーザーがコンソールを使用できること、および ポリシーを使用したアクセスの管理 管理ポリシーがユーザーにアタッチされていることを確認してください。

AWS CLI または Amazon RDS API のみを呼び出すユーザーには、最小限のコンソールアクセス許可を付与する必要はありません。

以下のポリシーでは、ルート AWS アカウントの Amazon Aurora リソースへのフルアクセスが付与されます。

```
AmazonRDSFullAccess
```

RDS リソースに対する Describe アクションの実行をユーザーに許可する

以下のアクセス権限ポリシーは、Describe で始まるすべてのアクションを実行するためのアクセス権限をユーザーに付与します。これらのアクションは、DB インスタンスなど RDS リソースに関する情報を表示します。Resource 要素内のワイルドカード文字 (*) は、アカウントによって所有されるすべての Amazon Aurora リソースに対してそれらのアクションが許可されることを示します。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowRDSDescribe",
         "Effect": "Allow",
         "Action": "rds:Describe*",
         "Resource": "*"
      }
   ]
}
```

指定した DB パラメータグループとサブネットグループを使用する DB インスタンスの作成をユーザーに許可する

以下の許可ポリシーは、mydbpg DB パラメータグループと mydbsubnetgroup DB サブネットグループを使用する必要のある DB インスタンスを作成することのみをユーザーに許可するための許可を付与します。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "VisualEditor0",
         "Effect": "Allow",
         "Action": "rds:CreateDBInstance",
         "Resource": [
```

```
            "arn:aws:rds:*:*:pg:mydbpg",
            "arn:aws:rds:*:*:subgrp:mydbsubnetgroup"
         ]
      }
   ]
}
```

2 つの異なる値を持つタグが付いたリソースに対するアクションにアクセス許可を付与する

アイデンティティベースのポリシーの条件を使用して、タグに基づいて Aurora リソースへのアクセスを制御できます。次のポリシーでは、stage タグが development または test に設定された DB インスタンスに対して CreateDBSnapshot API オペレーションを実行するためのアクセス許可が付与されます。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAnySnapshotName",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:snapshot:*"
      },
      {
         "Sid":"AllowDevTestToCreateSnapshot",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
```

```
   ]
}
```

次のポリシーでは、stage タグが development または test に設定された DB インスタンスに対して ModifyDBInstance API オペレーションを実行するためのアクセス許可が付与されます。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowChangingParameterOptionSecurityGroups",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource": [
            "arn:aws:rds:*:123456789012:pg:*",
            "arn:aws:rds:*:123456789012:secgrp:*",
            "arn:aws:rds:*:123456789012:og:*"
         ]
      },
      {
         "Sid":"AllowDevTestToModifyInstance",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
   ]
}
```

ユーザーによる DB インスタンスの削除を禁止する

以下のアクセス権限ポリシーは、特定の DB インスタンスを削除することをユーザーに禁止するためのアクセス権限を付与します。例えば、管理者以外のすべてのユーザーに対して、本稼働 DB インスタンスの削除を拒否することができます。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "DenyDelete1",
         "Effect": "Deny",
         "Action": "rds:DeleteDBInstance",
         "Resource": "arn:aws:rds:us-west-2:123456789012:db:my-mysql-instance"
      }
   ]
}
```

リソースへのすべてのアクセスを拒否する

リソースへのアクセスを明示的に拒否できます。拒否ポリシーは許可ポリシーよりも優先されます。以下のポリシーは、リソースを管理する機能をユーザーに明示的に拒否します。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Deny",
         "Action": "rds:*",
         "Resource": "arn:aws:rds:us-east-1:123456789012:db:mydb"
      }
   ]
}
```

ポリシー例: 条件キーの使用

以下に示しているのは、Amazon Aurora IAM アクセス許可ポリシーでの条件キーの使用例です。

例 1: 特定の DB エンジンを使用し、マルチ AZ ではない DB インスタンスを作成するためのアクセス許可を付与する

以下のポリシーでは、RDS 条件キーを使用して、MySQL データベースエンジンを使用するがマルチ AZ でない DB インスタンスのみをユーザーが作成できるようにします。Condition 要素では、データベースエンジンが MySQL であることが要件になることを示しています。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowMySQLCreate",
         "Effect": "Allow",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "rds:DatabaseEngine": "mysql"
            },
            "Bool": {
               "rds:MultiAz": false
            }
         }
      }
   ]
}
```

例 2: 特定の DB インスタンスクラスの DB インスタンスを作成するためのアクセス許可と、プロビジョンド IOPS を使用する DB インスタンスを作成するためのアクセス許可を明示的に拒否する

以下のポリシーでは、最もサイズが大きくてコストの高いインスタンスである DB インスタンスクラス r3.8xlarge と m4.10xlarge を使用する DB インスタンスの作成のためのアクセス許可を明示的に拒否しています。このポリシーでは、追加のコストが発生するプロビジョンド IOPS を使用する DB インスタンスの作成もユーザーに禁止しています。

明示的に拒否するアクセス権限は、付与する他のいずれのアクセス権限よりも優先されます。これにより、決して付与されることのないアクセス権限を ID が誤って取得することがなくなります。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
```

```
         "Sid": "DenyLargeCreate",
         "Effect": "Deny",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "rds:DatabaseClass": [
                  "db.r3.8xlarge",
                  "db.m4.10xlarge"
               ]
            }
         }
      },
      {
         "Sid": "DenyPIOPSCreate",
         "Effect": "Deny",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "NumericNotEquals": {
               "rds:Piops": "0"
            }
         }
      }
   ]
}
```

例 3: リソースにタグを付けるために使用できるタグキーと値のセットを制限する

次のポリシーは、RDS 条件キーを使用し、キー stage を持つタグの追加を値 test、qa、および production を持つリソースに追加することができます。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "rds:AddTagsToResource",
            "rds:RemoveTagsFromResource"
         ],
         "Resource": "*",
         "Condition": {
```

```
            "streq": {
               "rds:req-tag/stage": [
                  "test",
                  "qa",
                  "production"
               ]
            }
         }
      }
   ]
}
```

条件の指定: カスタムタグの使用

Amazon Aurora では、カスタムタグを使用して IAM ポリシーで条件を指定することがサポートされています。

例えば、environment という名前のタグを、beta、staging、production などの値で DB インスタンスに追加するとします。追加する場合、特定のユーザーを environment タグ値に基づく DB インスタンスに制限するポリシーを作成することができます。

> **Note**
>
> カスタムタグ識別子は大文字と小文字が区別されます。

以下の表では、Condition 要素で使用できる RDS タグ識別子を示しています。

RDS タグ識別子	適用先
db-tag	リードレプリカを含む DB インスタンス
snapshot-tag	DB スナップショット
ri-tag	リザーブド DB インスタンス
og-tag	DB オプショングループ
pg-tag	DB パラメータグループ
subgrp-tag	DB サブネットグループ

RDS タグ識別子	適用先
es-tag	イベントサブスクリプション
cluster-tag	DB クラスター
cluster-pg-tag	DB クラスターのパラメータグループ
cluster-snapshot-tag	DB クラススナップショット

カスタムタグの条件の構文は次のとおりです。

"Condition":{"StringEquals":{"rds:*rds-tag-identifier*/*tag-name*": ["*value*"]} }

例えば、次の Condition 要素は、environment という名前のタグを持ち、タグの値が production である DB インスタンスに適用されます。

"Condition":{"StringEquals":{"rds:db-tag/**environment**": ["**production**"]} }

タグの作成の詳細については、「Amazon RDS リソースのタグ付け」を参照してください。

> **Important**
>
> タグを使用して RDS リソースへのアクセスを管理する場合は、RDS リソースのタグへのアクセスを保護することをお勧めします。AddTagsToResource および RemoveTagsFromResource アクションのポリシーを作成することによって、タグへのアクセスを管理できます。例えば、次のポリシーは、ユーザーがすべてのリソースのタグを追加または削除することを拒否します。次に、特定のユーザーがタグを追加または削除することを許可するポリシーを作成できます。
>
> ```
> {
>    "Version":"2012-10-17",
>    "Statement":[
>       {
>          "Sid":"DenyTagUpdates",
>          "Effect":"Deny",
>          "Action":[
>             "rds:AddTagsToResource",
>             "rds:RemoveTagsFromResource"
> ```

```
            ],
            "Resource":"*"
        }
    ]
}
```

Aurora アクションのリストを確認するには、サービス認証リファレンスの「Amazon RDS で定義されるアクション」を参照してください。

ポリシー例: カスタムタグの使用

以下に示しているのは、Amazon Aurora IAM アクセス許可ポリシーでのカスタムタグの使用例です。Amazon Aurora リソースへのタグの追加の詳細については、「Amazon RDS の Amazon リソースネーム (ARN) の使用」を参照してください。

> **Note**
>
> すべての例で、us-west-2 リージョンを使用し、架空のアカウント ID を含めています。

例 1: 2 つの異なる値を持つタグが付いたリソースに対するアクションにアクセス許可を付与する

次のポリシーでは、`stage` タグが `development` または `test` に設定された DB インスタンスに対して `CreateDBSnapshot` API オペレーションを実行するためのアクセス許可が付与されます。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAnySnapshotName",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:snapshot:*"
      },
      {
         "Sid":"AllowDevTestToCreateSnapshot",
         "Effect":"Allow",
         "Action":[
```

```
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
   ]
}
```

次のポリシーでは、stage タグが development または test に設定された DB インスタンスに対して ModifyDBInstance API オペレーションを実行するためのアクセス許可が付与されます。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowChangingParameterOptionSecurityGroups",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource":" [
            "arn:aws:rds:*:123456789012:pg:*",
            "arn:aws:rds:*:123456789012:secgrp:*",
            "arn:aws:rds:*:123456789012:og:*"
         ]
      },
      {
         "Sid":"AllowDevTestToModifyInstance",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
```

```
                "development",
                "test"
            ]
         }
      }
   ]
}
```

例 2: 指定した DB パラメータグループを使用する DB インスタンスを作成するためのアクセス許可を明示的に拒否する

以下のポリシーでは、特定のタグ値が設定された DB パラメータグループを使用する DB インスタンスの作成のためのアクセス権限を明示的に拒否しています。DB インスタンスを作成するときに特定のユーザー定義の DB パラメータグループの使用を必須とする場合にも、このポリシーを適用できます。Deny を使用するポリシーは、ほとんどの場合、適用範囲のより広いポリシーによって付与されるアクセス許可を制限するために使用します。

明示的に拒否するアクセス権限は、付与する他のいずれのアクセス権限よりも優先されます。これにより、決して付与されることのないアクセス権限を ID が誤って取得することがなくなります。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"DenyProductionCreate",
         "Effect":"Deny",
         "Action":"rds:CreateDBInstance",
         "Resource":"arn:aws:rds:*:123456789012:pg:*",
         "Condition":{
            "StringEquals":{
               "rds:pg-tag/usage":"prod"
            }
         }
      }
   ]
}
```

例 3: インスタンス名にユーザー名がプレフィックスとして付加されている DB インスタンスに対するアクションにアクセス許可を付与する

以下のポリシーでは、DB インスタンス名の前にユーザー名が付いている DB インスタンスのうち、AddTagsToResource と同等の RemoveTagsFromResource というタグが付いているか、または stage というタグが付いていない DB インスタンスに対する、API (devo または stage を除く) の呼び出しのためのアクセス権限を付与しています。

ポリシーの Resource 行では、リソースをその Amazon Resource Name (ARN) により識別しています。ARN と Amazon Aurora リソースの使用の詳細については、「Amazon RDS の Amazon リソースネーム (ARN) の使用」を参照してください。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowFullDevAccessNoTags",
         "Effect":"Allow",
         "NotAction":[
            "rds:AddTagsToResource",
            "rds:RemoveTagsFromResource"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:${aws:username}*",
         "Condition":{
            "StringEqualsIfExists":{
               "rds:db-tag/stage":"devo"
            }
         }
      }
   ]
}
```

Amazon RDS の AWS マネージドポリシー

アクセス許可セットとロールにアクセス許可を追加するには、自分でポリシーを作成するよりも、AWS マネージドポリシーを使用する方が簡単です。チームに必要な権限のみを提供する IAM カスタマーマネージドポリシーを作成するには、時間と専門知識が必要です。すぐに使用を開始するために、AWS マネージドポリシーを使用できます。これらのポリシーは、一般的なユースケースをターゲット範囲に含めており、AWS アカウント で利用できます。AWS マネージドポリシーの詳細については、IAM ユーザーガイドの「AWS マネージドポリシー」を参照してください。

AWS のサービス は、AWS マネージドポリシーを維持し、更新します。AWS マネージドポリシーの権限を変更することはできません。サービスでは、新しい機能を利用できるようにするために、AWS マネージドポリシーに権限が追加されることがあります。この種類の更新は、ポリシーがアタッチされている、すべてのアイデンティティ (アクセス許可セットとロール) に影響を与えます。新しい機能が立ち上げられた場合や、新しいオペレーションが使用可能になった場合に、各サービスが AWS マネージドポリシーを更新する可能性が最も高くなります。サービスは、AWS マネージドポリシーから許可を削除しないため、ポリシーの更新によって既存の許可が破棄されることはありません。

さらに、AWS では、複数のサービスにまたがるジョブ機能のためのマネージドポリシーもサポートしています。例えば、ReadOnlyAccess AWS マネージドポリシーでは、すべての AWS のサービス およびリソースへの読み取り専用アクセスを許可します。あるサービスで新しい機能を立ち上げる場合は、AWS は、追加された演算とリソースに対し、読み込み専用の権限を追加します。職務機能ポリシーのリストと説明については、IAM ユーザーガイドの「ジョブ機能の AWS マネージドポリシー」を参照してください。

トピック

- AWS マネージドポリシー: AmazonRDSReadOnlyAccess
- AWS マネージドポリシー: AmazonRDSFullAccess
- AWS マネージドポリシー: AmazonRDSDataFullAccess
- AWS マネージドポリシー: AmazonRDSEnhancedMonitoringRole
- AWS マネージドポリシー: AmazonRDSPerformanceInsightsReadOnly
- AWS 管理ポリシー: AmazonRDSPerformanceInsightsFullAccess
- AWS マネージドポリシー: AmazonRDSDirectoryServiceAccess
- AWS マネージドポリシー: AmazonRDSServiceRolePolicy

AWS マネージドポリシー: AmazonRDSReadOnlyAccess

このポリシーは、AWS Management Console を通じた Amazon RDS への読み取り専用アクセスを許可します。

許可の詳細

このポリシーには、以下の許可が含まれています。

- `rds` — プリンシパルが Amazon RDS リソースを記述し、Amazon RDS リソースのタグを一覧表示することを許可します。
- `cloudwatch` — プリンシパルが Amazon CloudWatch メトリクスの統計情報を取得することを許可します。
- `ec2` — プリンシパルがアベイラビリティーゾーンとネットワークリソースを記述することを許可します。
- `logs` — プリンシパルがロググループの CloudWatch Logs ログストリームを記述し、CloudWatch Logs ログイベントを取得することを許可します。
- `devops-guru` - プリンシパルが Amazon DevOps Guru の対象となるリソースを記述できるようにします。リソースは、CloudFormation スタック名またはリソースタグで指定されます。

JSON ポリシードキュメントを含むこのポリシーの詳細については、AWS マネージドポリシーリファレンスガイドの「AmazonRDSReadOnlyAccess」を参照してください。

AWS マネージドポリシー: AmazonRDSFullAccess

このポリシーは、AWS Management Console を通じて Amazon RDS へのフルアクセスを提供します。

許可の詳細

このポリシーには、以下の許可が含まれています。

- `rds` - プリンシパルに Amazon RDS へのフルアクセスを許可します。
- `application-autoscaling` — プリンシパルがアプリケーションオートスケーリングのターゲットとポリシーを記述し、管理することを許可します。
- `cloudwatch` — プリンシパルが CloudWatch メトリクスの統計を取得し、CloudWatch アラームを管理することを許可します。

- ec2 — プリンシパルがアベイラビリティーゾーンとネットワークリソースを記述することを許可します。
- logs — プリンシパルがロググループの CloudWatch Logs ログストリームを記述し、CloudWatch Logs ログイベントを取得することを許可します。
- outposts — プリンシパルが AWS Outposts インスタンスタイプを取得することを許可します。
- pi — プリンシパルが Performance Insights メトリクスを取得することを許可します。
- sns — プリンシパルが Amazon Simple Notification Service (Amazon SNS) のサブスクリプションとトピックにアクセスして、Amazon SNS メッセージを発行することを許可します。
- devops-guru - プリンシパルが Amazon DevOps Guru の対象となるリソースを記述できるようにします。リソースは、CloudFormation スタック名またはリソースタグで指定されます。

JSON ポリシードキュメントを含むこのポリシーの詳細については、AWS マネージドポリシーリファレンスガイドの「AmazonRDSFullAccess」を参照してください。

AWS マネージドポリシー: AmazonRDSDataFullAccess

このポリシーは、特定の AWS アカウント 内の Aurora Serverless クラスターに対して Data API とクエリエディタを使用するためのフルアクセスを許可します。このポリシーは、AWS アカウント が AWS Secrets Manager からシークレットの値を取得することを許可します。

AmazonRDSDataFullAccess ポリシーは IAM ID にアタッチできます。

許可の詳細

このポリシーには、以下の許可が含まれています。

- dbqms — プリンシパルにクエリへのアクセス、作成、削除、記述、および更新を許可します。データベースクエリメタデータサービス (dbqms) は、内部専用のサービスです。このサービスでは、Amazon RDS を含む複数の AWS のサービス について、最新および保存済みのクエリを、AWS Management Console のクエリエディタ用に提供します。
- rds-data — プリンシパルが Aurora Serverless データベースに対して SQL ステートメントを実行するのを許可します。
- secretsmanager — プリンシパルが AWS Secrets Manager からシークレットの値を取得するのを許可します。

JSON ポリシードキュメントを含むこのポリシーの詳細については、AWS マネージドポリシーリファレンスガイドの「AmazonRDSDataFullAccess」を参照してください。

AWS マネージドポリシー: AmazonRDSEnhancedMonitoringRole

このポリシーは、Amazon RDS 拡張モニタリング用の Amazon CloudWatch Logs へのアクセスを提供します。

許可の詳細

このポリシーには、以下の許可が含まれています。

- `logs` — プリンシパルが CloudWatch Logs ロググループと保持ポリシーを作成し、ロググループの CloudWatch Logs ログストリームを作成および記述することを許可します。また、プリンシパルが CloudWatch Logs ログイベントを設定および取得することも許可します。

JSON ポリシードキュメントを含むこのポリシーの詳細については、AWS マネージドポリシーリファレンスガイドの「AmazonRDSEnhancedMonitoringRole」を参照してください。

AWS マネージドポリシー: AmazonRDSPerformanceInsightsReadOnly

このポリシーは、Amazon RDS DB インスタンスと Amazon Aurora DB クラスター用の Amazon RDS Performance Insights への読み取り専用アクセスを提供します。

このポリシーには、ポリシードキュメントの識別子として `Sid` (ステートメント ID) が含まれるようになりました。

許可の詳細

このポリシーには、以下の許可が含まれています。

- `rds` — プリンシパルが Amazon RDS DB インスタンスと Amazon Aurora DB クラスターを記述することを許可します。
- `pi` — プリンシパルが Amazon RDS Performance Insights API を呼び出し、Performance Insights メトリクスにアクセスすることを許可します。

JSON ポリシードキュメントを含むこのポリシーの詳細については、AWS マネージドポリシーリファレンスガイドの「AmazonRDSPerformanceInsightsReadOnly」を参照してください。

AWS 管理ポリシー: AmazonRDSPerformanceInsightsFullAccess

このポリシーは、Amazon RDS DB インスタンスと Amazon Aurora DB クラスター用の Amazon RDS Performance Insights へのフルアクセスを提供します。

このポリシーには、ポリシードキュメントの識別子として Sid (ステートメント ID) が含まれるようになりました。

許可の詳細

このポリシーには、以下の許可が含まれています。

- rds — プリンシパルが Amazon RDS DB インスタンスと Amazon Aurora DB クラスターを記述することを許可します。
- pi — プリンシパルが Amazon RDS Performance Insights API を呼び出したり、パフォーマンス分析レポートを作成、表示、削除したりすることを許可します。
- cloudwatch — プリンシパルが Amazon CloudWatch メトリクスを一覧表示し、メトリクスデータと統計を取得するのを許可します。

JSON ポリシードキュメントを含め、このポリシーの詳細については、「AWS マネージドポリシーリファレンスガイド」の「AmazonRDSPerformanceInsightsFullAccess」を参照してください。

AWS マネージドポリシー: AmazonRDSDirectoryServiceAccess

このポリシーは、Amazon RDS が AWS Directory Service を呼び出すことを許可します。

アクセス許可の詳細

このポリシーには、以下の許可が含まれています。

- ds — プリンシパルが AWS Directory Service ディレクトリを記述し、AWS Directory Service ディレクトリへの認可を制御することを許可します。

JSON ポリシードキュメントを含むこのポリシーの詳細については、AWS マネージドポリシーリファレンスガイドの「AmazonRDSDirectoryServiceAccess」を参照してください。

AWS マネージドポリシー: AmazonRDSServiceRolePolicy

IAM エンティティに AmazonRDSServiceRolePolicy をアタッチすることはできません。このポリシーは、Amazon RDS がユーザーに代わってアクションを実行することを許可するサービスリンクロールにアタッチされます。詳細については、「Amazon Aurora のサービスにリンクされたロールのアクセス許可」を参照してください。

Amazon RDS の AWS 管理ポリシーに関する更新

Amazon RDS の AWS マネージドポリシーに対する更新の詳細について、このサービスがこれらの変更の追跡を開始した以降のものを示します。このページへの変更に関する自動アラートを受け取るには、Amazon RDS のドキュメント履歴ページで RSS フィードにサブスクライブしてください。

変更	説明	日付
Amazon RDS の AWS マネージドポリシー – 既存のポリシーの更新	Amazon RDS は、`AWSServiceRoleForRDSCustom` サービスリンクロールの `AmazonRDSCustomServiceRolePolicy` に新しいアクセス許可を追加し、RDS Custom for SQL Server で基盤となるデータベースのホストインスタンスタイプを変更できるようにしました。また、RDS は、データベースホストに関するインスタンスタイプ情報を取得する `ec2:DescribeInstanceTypes` アクセス許可も追加しました。詳細については、「Amazon RDS の AWS マネージドポリシー」を参照してください。	2024 年 4 月 8 日
Amazon RDS の AWS マネージドポリシー - 新しいポリシー	Amazon RDS は、RDS Custom が EC2 インスタンスプロファイルを介して自動化アクションとデータベース管理タスクを実行できるように `AmazonRDSCustomInstanceProfileRolePolicy` という名前の新し	2024 年 2 月 27 日

変更	説明	日付
	いマネージドポリシーを追加しました。詳細については、「Amazon RDS の AWS マネージドポリシー」を参照してください。	
Amazon Aurora のサービスにリンクされたロールのアクセス許可 – 既存ポリシーへの更新	Amazon RDS は、AWSServiceRoleForRDS サービスリンクロールの AmazonRDSServiceRolePolicy に新しいステートメント ID を追加しました。 詳細については、「Amazon Aurora のサービスにリンクされたロールのアクセス許可」を参照してください。	2024 年 1 月 19 日
Amazon RDS の AWS マネージドポリシー - 既存のポリシーの更新	AmazonRDSPerformanceInsightsReadOnly および AmazonRDSPerformanceInsightsFullAccess マネージドポリシーには、Sid (ステートメント ID) がポリシーステートメントの識別子として含まれます。 詳細については、「AWS マネージドポリシー: AmazonRDSPerformanceInsightsReadOnly」および「AWS 管理ポリシー: AmazonRDSPerformanceInsightsFullAccess」を参照してください。	2023 年 10 月 23 日

変更	説明	日付
Amazon RDS の AWS マネージドポリシー – 既存のポリシーの更新	Amazon ECR では、新しいアクセス許可が AmazonRDSFullAccess 管理ポリシーに追加されました。これらのアクセス許可により、一定期間のパフォーマンス分析レポートを生成、表示、削除できます。 Performance Insights のアクセスポリシーの設定の詳細については、「Performance Insights 用のアクセスポリシーの設定」を参照してください。	2023 年 8 月 17 日

変更	説明	日付
Amazon RDS の AWS マネージドポリシー - 新しいポリシーと、既存のポリシーの更新	Amazon RDS では、AmazonRDSPerformanceInsightsReadOnly 管理ポリシーと AmazonRDSPerformanceInsightsFullAccess という名前の新しい管理ポリシーに新しいアクセス許可が追加されました。これらのアクセス許可により、一定期間の Performance Insights を分析したり、分析結果を推奨事項と共に表示したり、レポートを削除したりできます。 Performance Insights のアクセスポリシーの設定の詳細については、「Performance Insights 用のアクセスポリシーの設定」を参照してください。	2023 年 8 月 16 日

変更	説明	日付
Amazon RDS の AWS マネージドポリシー – 既存ポリシーへの更新	Amazon RDS は、AmazonRDSFullAccess と AmazonRDSReadOnlyAccess に新しい Amazon CloudWatch 名前空間 ListMetrics を追加しました。 この名前空間は、Amazon RDS が特定のリソース使用メトリクスを一覧表示するために必要です。 詳細については、Amazon CloudWatch ユーザーガイドの「CloudWatch リソースへの許可の管理の概要」を参照してください。	2023 年 4 月 4 日

変更	説明	日付
Amazon Aurora のサービスにリンクされたロールのアクセス許可 – 既存ポリシーへの更新	Amazon RDS は、AWS Secrets Manager との統合のために AWSServiceRoleForRDS サービスにリンクされたロールの AmazonRDSServiceRolePolicy に新しいアクセス許可を追加しました。RDS では、Secrets Manager でマスターユーザーのパスワードを管理するために、Secrets Manager との統合が必要です。シークレットでは予約された命名規則を使用しており、顧客からの更新を制限します。 詳細については、「Amazon Aurora および AWS Secrets Manager によるパスワード管理」を参照してください。	2022 年 12 月 22 日

変更	説明	日付
Amazon RDS の AWS マネージドポリシー - 既存のポリシーの更新	Amazon RDS は `AmazonRDSFullAccess` および `AmazonRDSReadOnlyAccess` マネージドポリシーに新しいアクセス許可を追加して、RDS コンソールで Amazon DevOps Guru を有効にできるようにしました。このアクセス許可は、DevOps Guru が有効になっているかどうかを確認するために必要です。 詳細については、「DevOps Guru for RDS の IAM アクセスポリシーの設定」を参照してください。	2022 年 12 月 19 日

変更	説明	日付
Amazon Aurora のサービスにリンクされたロールのアクセス許可 – 既存ポリシーへの更新	Amazon RDS は、`PutMetricData` の `AmazonRDSPreviewServiceRolePolicy` に新しい Amazon CloudWatch 名前空間を追加しました。 この名前空間は、Amazon RDS がリソース使用メトリクスを公開するために必要です。 詳細については、Amazon CloudWatch ユーザーガイドの「条件キーを使用した CloudWatch 名前空間へのアクセスの制限」を参照してください。	2022 年 6 月 7 日

変更	説明	日付
Amazon Aurora のサービスにリンクされたロールのアクセス許可 – 既存ポリシーへの更新	Amazon RDS は、PutMetricData の AmazonRDSBetaServiceRolePolicy に新しい Amazon CloudWatch 名前空間を追加しました。 この名前空間は、Amazon RDS がリソース使用メトリクスを公開するために必要です。 詳細については、Amazon CloudWatch ユーザーガイドの「条件キーを使用した CloudWatch 名前空間へのアクセスの制限」を参照してください。	2022 年 6 月 7 日
Amazon Aurora のサービスにリンクされたロールのアクセス許可 – 既存ポリシーへの更新	Amazon RDS は、PutMetricData の AWSServiceRoleForRDS に新しい Amazon CloudWatch 名前空間を追加しました。 この名前空間は、Amazon RDS がリソース使用メトリクスを公開するために必要です。 詳細については、Amazon CloudWatch ユーザーガイドの「条件キーを使用した CloudWatch 名前空間へのアクセスの制限」を参照してください。	2022 年 4 月 22 日

変更	説明	日付
Amazon RDS の AWS マネージドポリシー - 新しいポリシー	Amazon RDS では、Amazon RDS が DB インスタンスに代わって `AmazonRDSPerformanceInsightsReadOnly` サービスを呼び出せるように、AWS という名前の新しい管理ポリシーが追加されました。 Performance Insights のアクセスポリシーの設定の詳細については、「Performance Insights 用のアクセスポリシーの設定」を参照してください。	2022 年 3 月 10 日
Amazon Aurora のサービスにリンクされたロールのアクセス許可 – 既存ポリシーへの更新	Amazon RDS は、`PutMetricData` の `AWSServiceRoleForRDS` に新しい Amazon CloudWatch 名前空間を追加しました。 これらの名前空間は、Amazon DocumentDB (MongoDB と互換) と Amazon Neptune が CloudWatch メトリクスを公開するために必要です。 詳細については、Amazon CloudWatch ユーザーガイドの「条件キーを使用した CloudWatch 名前空間へのアクセスの制限」を参照してください。	2022 年 3 月 4 日

変更	説明	日付
Amazon RDS が変更の追跡を開始しました。	Amazon RDS が AWS マネージドポリシーの変更の追跡を開始しました。	2021 年 10 月 26 日

サービス間での混乱した代理問題の防止

「混乱した代理」問題は、アクションを実行するためのアクセス許可を持たないエンティティが、より特権のあるエンティティにアクションの実行を強制できてしまう場合に生じる、セキュリティ上の問題です。AWS では、サービス間でのなりすましが、混乱した代理問題を生じさせることがあります。

サービス間でのなりすましは、1 つのサービス (呼び出し元サービス) が、別のサービス (呼び出し対象サービス) を呼び出すときに発生する可能性があります。呼び出し元サービスが操作され、それ自身のアクセス許可を使用して、本来アクセス許可が付与されるべきではない方法で別の顧客のリソースに対して働きかけることがあります。これを防ぐために AWS では、お客様のすべてのサービスのデータを保護するのに役立つツールを提供しています。これらのツールでは、アカウントのリソースへのアクセス権が付与されたサービスプリンシパルを使用します。詳細については、IAM ユーザーガイド の 混乱した代理問題 を参照してください。

特定のリソースへのアクセスについて、Amazon RDS が別のサービスに付与する許可を制限する場合は、リソースポリシー内で `aws:SourceArn` および `aws:SourceAccount` のグローバル条件コンテキストキーを使用することをお勧めします。

例えば、Amazon S3 バケットの Amazon リソースネーム (ARN) を使用する場合など、`aws:SourceArn` 値にアカウント ID が含まれていないことがあります。このような場合は、前出のグローバル条件コンテキストキーの両方を使用して、パーミッションを制限する必要があります。場合によっては、両方のグローバル条件コンテキストキーと、アカウント ID を含む `aws:SourceArn` 値を併用します。これらを同じポリシーステートメントで使用する場合は、`aws:SourceAccount` の値には、`aws:SourceArn` 内のアカウントと同じアカウント ID を使用します。クロスサービスのアクセスにリソースを 1 つだけ関連付けたい場合は、`aws:SourceArn` を使用します。クロスサービスによる使用のために、AWS アカウント内の任意のリソースを関連づけたい場合は、`aws:SourceAccount` を使用します。

`aws:SourceArn` の値には、Amazon RDS リソースタイプの ARN を指定する必要があります。詳細については、「Amazon RDS の Amazon リソースネーム (ARN) の使用」を参照してください。

混乱した代理問題から保護するための最も効果的な方法は、リソースの完全な ARN を指定しながら、`aws:SourceArn` グローバル条件コンテキストキーを使用することです。この時、リソースの完全な ARN が分からない場合や、複数のリソースを指定しているという場合があり得ます。このような場合、ARN の未知の部分については、ワイルドカード (*) を指定しながら `aws:SourceArn` グローバルコンテキスト条件キーを使用します。例は `arn:aws:rds:*:123456789012:*` です。

次の例では、Amazon RDS で aws:SourceArn および aws:SourceAccount グローバル条件コンテキストキーを使用して、「混乱した代理」問題を回避する方法を示します。

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "rds.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:rds:us-east-1:123456789012:db:mydbinstance"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

グローバル条件コンテキストキーの aws:SourceArn と aws:SourceAccount を使用する他のポリシー例については、以下の各セクションを参照してください。

- Amazon SNS トピックに通知を発行するアクセス許可を付与する
- Amazon S3 バケットへのアクセスを設定する (PostgreSQL のインポート)
- Amazon S3 バケットへのアクセスを設定する (PostgreSQL エクスポート)

の IAM データベース認証

AWS Identity and Access Management (IAM) データベース認証を使用して、DB クラスターを認証できます。IAM データベース認証には、Aurora MySQL、および Aurora PostgreSQL を使用します。この認証方法では、DB クラスターに接続するときにパスワードを使用する必要はありません。代わりに、認証トークンを使用します。

認証トークンは、 Amazon Aurora がリクエストに応じて生成する一意の文字列です。認証トークンは、AWS 署名バージョン 4 を使用して生成されます。各トークンには 15 分の有効期間があります。認証は IAM を使用して外部的に管理されるため、ユーザー認証情報をデータベースに保存する必要はありません。引き続きスタンダードのデータベース認証を使用することもできます。トークンは認証にのみ使用され、確立後のセッションには影響しません。

IAM データベース認証には次の利点があります。

- データベースとの間で送受信されるネットワークトラフィックは、Secure Socket Layer (SSL) または Transport Layer Security (TLS) を使用して暗号化されます。Amazon Aurora で SSL/TLS を使用する方法については、「SSL/TLS を使用した DB クラスターへの接続の暗号化」を参照してください。
- IAM を使用して各 DB クラスターで個別に管理するのではなく、データベースリソースへのアクセスを一元的に管理できます。
- Amazon EC2 で実行するアプリケーションの場合、セキュリティを高めるため、EC2 インスタンスに固有のプロファイル認証情報を使用して、パスワードの代わりにデータベースにアクセスできます。

一般に、アプリケーションが 1 秒あたり 200 未満の接続を作成し、アプリケーションコードでユーザー名とパスワードを直接管理したくない場合は、IAM データベース認証の使用を検討してください。

Amazon Web Services (AWS) JDBC ドライバーは IAM データベース認証をサポートしています。詳細については、「Amazon Web Services (AWS) JDBC ドライバー GitHub リポジトリ」の「AWS IAM Authentication Plugin」を参照してください。https://github.com/aws/aws-advanced-jdbc-wrapper

Amazon Web Services (AWS) Python ドライバーは IAM データベース認証をサポートしています。詳細については、「Amazon Web Services (AWS) Python ドライバー GitHub リポジトリ」の「AWS IAM Authentication Plugin」を参照してください。https://github.com/aws/aws-advanced-python-wrapper

トピック

リージョンとバージョンの可用性

利用できる機能とそのサポートは、各 Aurora データベースエンジンの特定のバージョン、および AWS リージョン によって異なります。Aurora と IAM データベース認証を使用したバージョンとリージョンの可用性の詳細については、「IAM データベース認証でサポートされているリージョンと Aurora DB エンジン」を参照してください。

Aurora MySQL では、サポートされているすべての DB インスタンスクラスは、db.t2.small および db.t3.small を除き、IAM データベース認証をサポートします。サポートされている DB インスタンスクラスの詳細については、「DB インスタンスクラスでサポートされている DB エンジン」を参照してください。

CLI および SDK のサポート

IAM データベース認証は、AWS CLI と以下の各言語固有の AWS SDK について使用できます。

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP
- AWS SDK for Python (Boto3)

- AWS SDK for Ruby

IAM データベース認証の制限

IAM データベース認証を使用する場合、以下の制限が適用されます。

- DB クラスターの 1 秒あたりの最大接続数は、DB インスタンスクラスとワークロードに応じて制限される場合があります。DB 負荷のピーク時にリソースが枯渇した場合、IAM 認証が失敗する可能性があります。
- 現在、IAM データベース認証はすべてのグローバル条件コンテキストキーをサポートしていません。

  グローバル条件コンテキストキーの詳細については、「IAM ユーザーガイド」の「AWS グローバル条件コンテキストキー」を参照してください。
- PostgreSQL の場合、IAM ロール (`rds_iam`) がマスターユーザーに追加される (マスターユーザーである RDS を含む) と、IAM 認証はパスワード認証よりも優先されるため、ユーザーは IAM ユーザーとしてログインする必要があります。
- Aurora PostgreSQL では、IAM 認証を使用してレプリケーション接続を確立することはできません。
- DB クラスター エンドポイントの代わりに、カスタム Route 53 DNS レコードを使用して認証トークンを生成することはできません。
- CloudWatch と CloudTrail は IAM 認証のログ記録を行いません。これらのサービスは、IAM ロールにデータベース接続の有効化を許可する `generate-db-auth-token` API コールを追跡しません。詳細については、「Achieve auditability with Amazon RDS IAM authentication using attribute-based access control」を参照してください。

IAM データベース認証に関する推奨事項

IAM データベース認証を使用する場合には、以下のことをお勧めします。

- アプリケーションが必要とする新しい IAM データベース認証接続が 1 秒あたり 200 未満の場合は、IAM データベース認証を使用します。

  Amazon Aurora を使用するデータベースエンジンでは、1 秒あたりの認証試行回数に制限はありません。ただし、IAM データベース認証を使用するときは、アプリケーションは認証トークンを生成する必要があります。次に、アプリケーションはそのトークンを使用して DB クラスターに接

続します。1 秒あたりの新しい接続数の上限を超えた場合、IAM データベース認証の追加オーバーヘッドによって接続のスロットリングが発生する場合があります。

接続が頻繁に作成されるのを軽減するために、アプリケーションで接続プールを使用することを検討してください。これにより、IAM DB 認証のオーバーヘッドが軽減され、アプリケーションで既存の接続を再利用できるようになります。または、これらのユースケースでは RDS Proxy の使用を検討してください。RDS Proxy には追加料金がかかります。「RDS Proxy の料金表」をご覧ください。

- IAM データベース認証トークンのサイズは、IAM タグの数、IAM サービスポリシー、ARN の長さ、その他の IAM やデータベースのプロパティなど、さまざまな要素によって異なります。このトークンの最小サイズは、通常、約 1 KB ですが、それ以上になることもあります。このトークンは IAM 認証を使用するデータベースへの接続文字列のパスワードとして使用されるため、データベースドライバー (ODBC など) やツールが、サイズを理由にこのトークンを制限したり、切り詰めたりしないようにする必要があります。トークンが切り詰められると、データベースと IAM による認証検証は失敗します。
- IAM データベース認証トークンの作成時に一時的な認証情報を使用している場合でも、IAM データベース認証トークンを使用して接続リクエストを行うときには、その一時的な認証情報が引き続き有効なものである必要があります。

サポートされていない AWS グローバル条件コンテキストキー

IAMデータベース認証は AWS グローバル条件コンテキストキーのうち次のサブセットをサポートしていません。

- `aws:Referer`
- `aws:SourceIp`
- `aws:SourceVpc`
- `aws:SourceVpce`
- `aws:UserAgent`
- `aws:VpcSourceIp`

条件キーの詳細については、IAM ユーザーガイド の「AWS グローバル条件コンテキストキー」 を参照してください。

IAM データベース認証の有効化と無効化

デフォルトでは、IAM データベース認証は DB クラスターで無効になります。AWS Management Console、AWS CLI、API のいずれかを使用して、IAM データベース認証を有効または無効にすることができます。

次のいずれかのアクションを実行する際に、IAM データベース認証を有効にすることができます。

- IAM データベース認証を有効にして新しい DB クラスターを作成するには、「Amazon Aurora DB クラスターの作成」を参照してください。
- DB クラスターを変更して IAM データベース認証を有効にするには、「Amazon Aurora DB クラスターの変更」を参照してください。
- IAM データベース認証を有効にしてスナップショットから DB クラスターを復元するには、「DB クラスターのスナップショットからの復元」を参照してください。
- IAM データベース認証を有効化しながら DB クラスター を特定の時点に復元するには、「DB クラスターを指定の時点の状態に復元する」を参照してください。

コンソール

作成または変更の各ワークフローには、[データベース認証] セクションがあり、IAM データベース認証を有効または無効にすることができます。そのセクションで、[パスワードと IAM データベース認証] を選択して、IAM データベース認証を有効にします。

既存の DB クラスターに対して IAM データベース認証を有効または無効にするには

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. ナビゲーションペインで、[データベース] を選択します。
3. 変更する DB クラスターを選択します。

> **Note**
>
> DB クラスター内のすべての DB インスタンスが IAM と互換性がある場合にのみ、IAM 認証を有効にすることができます。リージョンとバージョンの可用性 の互換性要件を確認する。

4. [Modify] を選択します。

5. [データベース認証] セクションで、[パスワードと IAM データベース認証] を選択して、IAM データベース認証を有効にします。IAM 認証を無効にするには、[パスワード認証] または [パスワードと Kerberos 認証] を選択します。
6. [続行] を選択します。
7. 変更をすぐに適用するには、[変更のスケジューリング] セクションで [今すぐ] を選択します。
8. [クラスターを変更] を選択します。

AWS CLI

AWS CLI を使用して、IAM 認証により新しい DB インスタンスクラスターを作成するには、`create-db-cluster` コマンドを使用します。`--enable-iam-database-authentication` オプションを指定します。

既存の DB クラスターを更新して、IAM 認証を持つ、または持たないようにするには、AWS CLI コマンド `modify-db-cluster` を使用します。必要に応じて `--enable-iam-database-authentication` または `--no-enable-iam-database-authentication` オプションを指定します。

> **Note**
>
> DB クラスター内のすべての DB インスタンスが IAM と互換性がある場合にのみ、IAM 認証を有効にすることができます。リージョンとバージョンの可用性 の互換性要件を確認する。

デフォルトでは、Aurora は次のメンテナンスウィンドウ中に変更を実行します。これを上書きし、IAM DB 認証をできるだけ早く有効にする場合は、`--apply-immediately` パラメータを使用します。

DB クラスターを復元する場合は、次のいずれかの AWS CLI コマンドを使用します。

- `restore-db-cluster-to-point-in-time`
- `restore-db-cluster-from-db-snapshot`

IAM データベース認証設定は、デフォルトで元のスナップショットの設定になります。この設定を変更するには、必要に応じて `--enable-iam-database-authentication` または `--no-enable-iam-database-authentication` オプションを設定します。

RDS API

API を使用して、IAM 認証で新しい DB インスタンスを作成するには、API オペレーション `CreateDBCluster` を使用します。`EnableIAMDatabaseAuthentication` パラメータを `true` に設定します。

既存の DB クラスターを更新して、IAM 認証を持つ、または持たないようにするには、API オペレーション `ModifyDBCluster` を使用します。`EnableIAMDatabaseAuthentication` パラメータを `true` に設定して IAM 認証を有効にするか、`false` に設定して無効にします。

> **Note**
>
> DB クラスター内のすべての DB インスタンスが IAM と互換性がある場合にのみ、IAM 認証を有効にすることができます。リージョンとバージョンの可用性 の互換性要件を確認する。

DB クラスターを復元する場合は、次のいずれかの API オペレーションを使用します。

- `RestoreDBClusterFromSnapshot`
- `RestoreDBClusterToPointInTime`

IAM データベース認証設定は、デフォルトで元のスナップショットの設定になります。この設定を変更するには、`EnableIAMDatabaseAuthentication` パラメータを `true` に設定して IAM 認証を有効にするか、`false` に設定して無効にします。

IAM データベースアクセス用の IAM ポリシーの作成と使用

ユーザーまたはロールに DB クラスターへの接続を許可するには、IAM ポリシーを作成する必要があります。その後、ポリシーをアクセス許可セットまたはロールにアタッチします。

> **Note**
>
> IAM キーポリシーの詳細については、「Amazon Aurora での Identity and Access Management」を参照してください。

次のポリシー例では、ユーザーは IAM データベース認証を使用して、DB クラスターに接続できます。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:1234567890:dbuser:cluster-ABCDEFGHIJKL01234/
db_user"
         ]
      }
   ]
}
```

Important

管理者権限を持つユーザーは、IAM ポリシーで明示的なアクセス許可が設定されていない場合でも、DB のクラスターにアクセスできます。管理者アクセスを DB のクラスターに制限するには、最低限のアクセス許可が適切に設定された IAM ロールを作成し、それを管理者に設定します。

Note

`rds-db:` プレフィックスと、`rds:` で始まる他の RDS API オペレーションのプレフィックスを混同しないでください。IAM データベース認証に対してのみ、`rds-db:` プレフィックスと `rds-db:connect` アクションを使用します。これらは、その他のコンテキストでは有効ではありません。

このポリシーには、次の要素を持つ 1 つのステートメントが含まれています。

- `Effect` - DB クラスターへのアクセスを許可するには、`Allow` を指定します。アクセスを明示的に許可しない場合、デフォルトでアクセスは拒否されます。
- `Action` - DB クラスターへの接続を許可するには、`rds-db:connect` を指定します。

- `Resource` - 1 つの DB クラスターで 1 つのデータベースアカウントを示す Amazon リソースネーム (ARN) を指定します。ARN 形式は次のとおりです。

```
arn:aws:rds-db:region:account-id:dbuser:DbClusterResourceId/db-user-name
```

この形式では、以下のように置き換えます。

- *region* は、DB クラスターの AWS リージョンです。このポリシー例での AWS リージョンは `us-east-2` です。
- *account-id* は DB クラスターの AWS アカウント番号です。このポリシー例でのアカウント番号は `1234567890` です。ユーザーは DB クラスターのアカウントと同じアカウントでなければなりません。

  クロスアカウントアクセスを実行するには、DB クラスターのアカウントに上記のポリシーで IAM ロールを作成し、他のアカウントがそのロールを引き継ぐことを許可します。
- *DbClusterResourceId* は、DB クラスターの識別子です。この識別子は AWS リージョンに固有であり、変更されることはありません。このポリシー例での識別子は `cluster-ABCDEFGHIJKL01234` です。

  Amazon Aurora 用の DB クラスターのリソース ID を AWS Management Console で検索するには、DB クラスターを選択して、その詳細を表示します。そして、[Configuration (設定)] タブを選択します。[設定] セクションに [リソース ID] が表示されます。

  または、次のように AWS CLI コマンドを使用して、以下に示されているように、現在の AWS リージョンのすべての DB クラスターの識別子とリソース ID をリストできます。

```
aws rds describe-db-clusters --query "DBClusters[*].
[DBClusterIdentifier,DbClusterResourceId]"
```

> **Note**
>
> RDS Proxy 経由でデータベースに接続する場合は、`prx-ABCDEFGHIJKL01234` などのプロキシリソース ID を指定します。RDS Proxy で IAM データベース認証を使用する方法については、「IAM 認証を使用したプロキシへの接続」を参照してください。

- *db-user-name* は、IAM 認証に関連付けるデータベースアカウントの名前です。このポリシー例で、データベースアカウントは db_user です。

多様なアクセスパターンをサポートするため、他の ARN を構築できます。次のポリシーでは、DB クラスターで 2 つの異なるデータベースアカウントにアクセスできます。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-ABCDEFGHIJKL01234/
jane_doe",
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-ABCDEFGHIJKL01234/
mary_roe"
         ]
      }
   ]
}
```

次のポリシーでは、特定の AWS アカウントと AWS リージョンのすべての DB クラスターとデータベースアカウントに一致させるために「*」文字を使用します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rds-db:connect"
            ],
            "Resource": [
                "arn:aws:rds-db:us-east-2:1234567890:dbuser:*/*"
            ]
```

```
        }
    ]
}
```

次のポリシーは、特定の AWS アカウントと AWS リージョンの DB クラスターすべてに一致します。ただし、jane_doe データベースアカウントを持つ DB クラスターまたは DB クラスターにのみにアクセスが許可されます。

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:*/jane_doe"
         ]
      }
   ]
}
```

ユーザーまたはロールは、データベースユーザーがアクセスするデータベースにのみアクセスできます。例えば、DB クラスターに dev という名前のデータベースと、test という名前の別のデータベースがあるとします。データベースユーザー jane_doe が dev のみにアクセスできる場合、jane_doe ユーザーでその DB クラスターにアクセスできるユーザーまたはロールも、dev にのみアクセスできます。このアクセス制限は、テーブル、ビューなどその他のデータベースオブジェクトにも当てはまります。

管理者は、エンティティに必要な、指定されたリソースに対して特定の API オペレーションを実行するアクセス許可を付与する IAM ポリシーを作成する必要があります。続いて、管理者は、それらのアクセス許可を必要とするアクセス許可セットまたはロールに、そのポリシーをアタッチします。ポリシーの例については、「Amazon Aurora のアイデンティティベースのポリシーの例」を参照してください。

IAM ポリシーをアクセス許可セットまたはロールにアタッチする

データベース認証を許可する IAM ポリシーを作成した後、そのポリシーをアクセス許可セットまたはロールにアタッチする必要があります。このトピックに関するチュートリアルについては、IAM ユーザーガイドの「はじめてのカスタマー管理ポリシーの作成とアタッチ」を参照してください。

チュートリアルを進める際に、このセクションに記載されているいずれかのポリシー例をスタート点として使用し、ニーズに合わせて調整することができます。チュートリアルを完了すると、`rds-db:connect` アクションを利用できる、ポリシーがアタッチされたアクセス許可セットが作成されます。

> **Note**
>
> 複数のアクセス許可セットまたはロールを同じデータベースユーザーアカウントにマップできます。例えば、IAM ポリシーで以下のリソース ARN を指定したとします。
>
> ```
> arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-12ABC34DEFG5HIJ6KLMNOP78QR/jane_doe
> ```
>
> ポリシーを Jane、Bob、Diego にアタッチした場合、これらの各ユーザーは、`jane_doe` データベースアカウントを使用して、指定された DB クラスターに接続できます。

IAM 認証を使用したデータベースアカウントの作成

IAM データベース認証では、作成するユーザーアカウントにデータベースのパスワードを割り当てる必要はありません。データベースアカウントにマッピングされているユーザーを削除した場合は、`DROP USER` ステートメントでデータベースアカウントも削除する必要があります。

> **Note**
>
> IAM 認証に使用されるユーザー名は、データベース内のユーザー名の大文字および小文字と一致する必要があります。

トピック

- Aurora MySQL での IAM 認証の使用

- Aurora PostgreSQL での IAM 認証の使用

Aurora MySQL での IAM 認証の使用

Aurora MySQL では、認証は、AWSAuthenticationPlugin (IAM とシームレスに連携してユーザーを認証する AWS 提供のプラグイン) によって処理されます。DB クラスターに、マスターユーザーまたはユーザーを作成して権限を付与できる別のユーザーとして接続します。接続後、次の例に示すように、CREATE USER ステートメントを発行します。

```
CREATE USER jane_doe IDENTIFIED WITH AWSAuthenticationPlugin AS 'RDS';
```

IDENTIFIED WITH 句により、Aurora MySQL は AWSAuthenticationPlugin を使用して、データベースアカウント (jane_doe) を認証できます。AS 'RDS' 句は、認証方式を参照します。指定したデータベースユーザー名は、IAM データベースアクセスの IAM ポリシー内のリソースと同じであることを確認します。詳細については、「IAM データベースアクセス用の IAM ポリシーの作成と使用」を参照してください。

> **Note**
>
> 次のメッセージが表示された場合、AWS が提供するプラグインが、現在の DB クラスターに使用できないことを意味します。
> `ERROR 1524 (HY000): Plugin 'AWSAuthenticationPlugin' is not loaded`
> このエラーをトラブルシューティングするには、サポートされている設定を使用していること、および DB クラスターで IAM データベース認証を有効にしていることを確認します。詳細については、「リージョンとバージョンの可用性」および「IAM データベース認証の有効化と無効化」を参照してください。

AWSAuthenticationPlugin を使用してアカウントを作成したら、他のデータベースのアカウントと同様に管理します。例えば、GRANT および REVOKE ステートメントでアカウント特権を変更したり、ALTER USER ステートメントでさまざまなアカウント属性を変更したりできます。

IAM を使用する場合、データベースネットワークトラフィックは SSL/TLS を使用して暗号化されます。SSL 接続を許可するには、以下のコマンドでユーザーアカウントを変更します。

```
ALTER USER 'jane_doe'@'%' REQUIRE SSL;
```

Aurora PostgreSQL での IAM 認証の使用

Aurora PostgreSQL で IAM 認証を使用するには、マスターユーザーまたはユーザーを作成して権限を付与できる別のユーザーとして DB クラスターに接続します。接続後、データベースユーザーを作成して、次の例に示すように、ユーザーに `rds_iam` ロールを付与します。

```
CREATE USER db_userx;
GRANT rds_iam TO db_userx;
```

指定したデータベースユーザー名は、IAM データベースアクセスの IAM ポリシー内のリソースと同じであることを確認します。詳細については、「IAM データベースアクセス用の IAM ポリシーの作成と使用」を参照してください。

PostgreSQL データベースユーザーは IAM または Kerberos 認証のいずれかを使用できますが、両方を使用することはできないため、このユーザーも `rds_ad` ロールを持つことはできません。これは、ネストされたメンバーシップにも適用されます。詳細については、「 ステップ 7: Kerberos プリンシパル用の PostgreSQL ユーザーを作成する 」を参照してください。

IAM 認証を使用した DB クラスターへの接続

IAM データベース認証では、DB クラスターに接続するときに認証トークンを使用します。認証トークンは、パスワードの代わりに使用する文字列です。認証トークンを生成した後、期限切れになるまで 15 分間有効です。期限切れのトークンを使用して接続を試みると、接続リクエストは拒否されます。

すべての認証トークンは、AWS 署名バージョン 4 を使用した有効な署名が添付されている必要があります (詳細については、AWS 全般のリファレンスの「Signature Version 4 の署名プロセス」を参照してください)。AWS CLI や AWS など、AWS SDK for Java とAWS SDK for Python (Boto3) SDK は、作成した各トークンに自動的に署名できます。

別の AWS のサービス (AWS Lambda など) から Amazon Aurora に接続するときに、認証トークンを使用できます。トークンを使用することで、コードにパスワードを含めないで済みます。あるいは、AWS SDK を使用して、認証トークンをプログラムで作成して、プログラムで署名することもできます。

IAM 認証トークンに署名した後、 Aurora DB クラスターに接続できます。以下では、コマンドラインツールまたは AWS や AWS SDK for Java などの AWS SDK for Python (Boto3) SDK を使用して、これを行う方法を示しています。

詳細については、以下のブログ投稿を参照してください。

- IAM 認証を使用して SQL Workbench/J により Aurora MySQL または Amazon RDS for MySQL に接続する
- pgAdmin Amazon Aurora PostgreSQL または Amazon RDS for PostgreSQL と接続するための IAM 認証の使用

前提条件

IAM 認証を使用して DB クラスターに接続するための前提条件は以下のとおりです。

- IAM データベース認証の有効化と無効化
- IAM データベースアクセス用の IAM ポリシーの作成と使用
- IAM 認証を使用したデータベースアカウントの作成

トピック

- IAM 認証と AWS ドライバーを使用した DB クラスターへの接続
- コマンドラインから IAM 認証を使用して、DB クラスターに接続する: AWS CLI および mysql クライアント
- コマンドラインから IAM 認証を使用して DB クラスターに接続する: AWS CLI および psql クライアント
- IAM 認証および AWS SDK for .NET を使用した DB クラスターへの接続
- IAM 認証および AWS SDK for Go を使用した DB クラスターへの接続
- IAM 認証および AWS SDK for Java を使用した DB クラスターへの接続
- IAM 認証および AWS SDK for Python (Boto3) を使用した DB クラスターへの接続

IAM 認証と AWS ドライバーを使用した DB クラスターへの接続

AWS のドライバースイートは、スイッチオーバーとフェイルオーバーの時間の短縮、AWS Secrets Manager、AWS Identity and Access Management (IAM)、フェデレーティッド ID での認証をサポートするように設計されています。AWS ドライバーは、DB クラスターステータスをモニタリングし、クラスタートポロジを認識して新しいライターを決定することを前提としています。このアプローチにより、スイッチオーバーとフェイルオーバーの時間が 1 桁秒に短縮されます (オープンソースドライバーの場合は数十秒)。

AWS ドライバーの詳細については、お使いの Aurora MySQL または Aurora PostgreSQL DB クラスターに対応する言語ドライバーを参照してください。

コマンドラインから IAM 認証を使用して、DB クラスターに接続する: AWS CLI および mysql クライアント

以下に示すように、AWS CLI および `mysql` コマンドラインツールを使用して、コマンドラインから Aurora DB クラスターに接続できます。

前提条件

IAM 認証を使用して DB クラスターに接続するための前提条件は以下のとおりです。

- IAM データベース認証の有効化と無効化
- IAM データベースアクセス用の IAM ポリシーの作成と使用
- IAM 認証を使用したデータベースアカウントの作成

> **Note**
>
> IAM 認証を使用して SQLWorkbench/J を使用してデータベースに接続する方法については、ブログ記事「IAM 認証を使用して SQL Workbench/J で Aurora MySQL または Amazon RDS for MySQL に接続する」を参照してください。

トピック

- IAM 認証トークンの生成
- DB クラスターへの接続

IAM 認証トークンの生成

次の例では、AWS CLI を使用して署名された認証トークンを取得する方法を示します。

```
aws rds generate-db-auth-token \
   --hostname rdsmysql.123456789012.us-west-2.rds.amazonaws.com \
   --port 3306 \
   --region us-west-2 \
   --username jane_doe
```

この例で、パラメータは次のとおりです。

- `--hostname` - アクセス先の DB クラスターのホスト名。
- `--port` - DB クラスターへの接続に使用するポート番号
- `--region` - DB クラスターが実行中の AWS リージョン
- `--username` - アクセス先のデータベースアカウント

トークンの初期の複数の文字は次のようになります。

```
rdsmysql.123456789012.us-west-2.rds.amazonaws.com:3306/?
Action=connect&DBUser=jane_doe&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Expires=900...
```

> **Note**
>
> DB クラスターエンドポイントの代わりに、カスタム Route 53 DNS レコードまたは Aurora カスタムエンドポイントを使用して認証トークンを生成することはできません。

DB クラスターへの接続

接続の全般的な形式を次に示します。

```
mysql --host=hostName --port=portNumber --ssl-ca=full_path_to_ssl_certificate --enable-
cleartext-plugin --user=userName --password=authToken
```

パラメータは次のとおりです。

- `--host` - アクセス先の DB クラスターのホスト名。
- `--port` - DB クラスターへの接続に使用するポート番号
- `--ssl-ca` - 公開キーを含む SSL 証明書ファイルへのフルパス

  詳細については、「Aurora MySQL DB クラスターでの TLS の使用」を参照してください。

  SSL 証明書をダウンロードするには SSL/TLS を使用した DB クラスターへの接続の暗号化 を参照ください。
- `--enable-cleartext-plugin` - この接続で `AWSAuthenticationPlugin` を使用する必要があることを示す値

MariaDB クライアントを使用している場合、--enable-cleartext-plugin オプションは必須ではありません。

- --user - アクセス先のデータベースアカウント
- --password - 署名済みの IAM 認証トークン

認証トークンは数百の文字で構成されます。これは、コマンドラインでは手に負えなくなる可能性があります。この問題を回避する 1 つの方法は、環境可変にトークンを保存し、接続時にその可変を使用することです。次の例は、この回避策を実行する 1 つの方法を示しています。この例では、*/sample_dir/* が公開キーを含む SSL 証明書ファイルへのフルパスです。

```
RDSHOST="mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
TOKEN="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 3306 --region us-
west-2 --username jane_doe )"

mysql --host=$RDSHOST --port=3306 --ssl-ca=/sample_dir/global-bundle.pem --enable-
cleartext-plugin --user=jane_doe --password=$TOKEN
```

AWSAuthenticationPlugin を使って接続した場合、接続は SSL を使用して保護されます。これを確認するには、mysql> コマンドプロンプトで以下を入力します。

```
show status like 'Ssl%';
```

出力の次の行に詳細情報が示されます。

```
+---------------+-------------+
| Variable_name | Value

                                                                    |

+---------------+-------------+
| ...           | ...
| Ssl_cipher    | AES256-SHA

                                                                    |

| ...           | ...
| Ssl_version   | TLSv1.1

                                                                    |
```

```
| ...            | ...
+----------------------------+
```

プロキシ経由で DB クラスターに接続する場合は、「IAM 認証を使用したプロキシへの接続」を参照してください。

コマンドラインから IAM 認証を使用して DB クラスターに接続する: AWS CLI および psql クライアント

以下に示すように、AWS CLI および psql コマンドラインツールを使用して、コマンドラインから Aurora PostgreSQL DB クラスターに接続できます。

前提条件

IAM 認証を使用して DB クラスターに接続するための前提条件は以下のとおりです。

- IAM データベース認証の有効化と無効化
- IAM データベースアクセス用の IAM ポリシーの作成と使用
- IAM 認証を使用したデータベースアカウントの作成

> **Note**
>
> pgAdminを使用してIAM認証でデータベースに接続する方法については、ブログ記事IAM認証を使用してpgAdmin Amazon Aurora PostgreSQLまたはAmazon RDS for PostgreSQLで接続するを参照してください。

トピック

- IAM 認証トークンの生成
- Aurora PostgreSQL クラスターへの接続

IAM 認証トークンの生成

認証トークンは数百の文字で構成されるため、コマンドラインでは手に負えなくなる可能性があります。この問題を回避する 1 つの方法は、環境可変にトークンを保存し、接続時にその可変を使用することです。次の例では、AWS CLI コマンドを使用して署名された認証トークンを取得するために `generate-db-auth-token` を使用し、`PGPASSWORD` 環境可変に格納する方法を示しています。

```
export RDSHOST="mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com"
export PGPASSWORD="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 5432 --
region us-west-2 --username jane_doe )"
```

例では、generate-db-auth-token コマンドへのパラメータは次のとおりです。

- --hostname - アクセス先の DB クラスター (クラスターエンドポイント) のホスト名
- --port - DB クラスターへの接続に使用するポート番号
- --region - DB クラスターが実行中の AWS リージョン
- --username - アクセス先のデータベースアカウント

生成されたトークンの初期の複数の文字は次のようになります。

```
mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com:5432/?
Action=connect&DBUser=jane_doe&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Expires=900...
```

> **Note**
>
> DB クラスターエンドポイントの代わりに、カスタムRoute 53 DNS レコードまたは Aurora カスタムエンドポイントを使用して認証トークンを生成することはできません。

Aurora PostgreSQL クラスターへの接続

psql を使用して接続する全般的な形式を次に示します。

```
psql "host=hostName port=portNumber sslmode=verify-full
 sslrootcert=full_path_to_ssl_certificate dbname=DBName user=userName
 password=authToken"
```

パラメータは次のとおりです。

- host - アクセス先の DB クラスター (クラスターエンドポイント) のホスト名
- port - DB クラスターへの接続に使用するポート番号
- sslmode - 使用する SSL モード

`sslmode=verify-full` を使用すると、SSL 接続で DB クラスターのエンドポイントを SSL 証明書のエンドポイントと照合します。

- `sslrootcert` - 公開キーを含む SSL 証明書ファイルへのフルパス

  詳細については、「SSL/TLS での Aurora PostgreSQL データの保護」を参照してください。

  SSL 証明書をダウンロードするには SSL/TLS を使用した DB クラスターへの接続の暗号化 を参照ください。
- dbname - アクセス先のデータベース
- user - アクセス先のデータベースアカウント
- password - 署名済みの IAM 認証トークン

> **Note**
>
> DB クラスターエンドポイントの代わりに、カスタムRoute 53 DNS レコードまたは Aurora カスタムエンドポイントを使用して認証トークンを生成することはできません。

次の例は、psql を使用して接続する方法を示しています。この例の psql では、環境変数 RDSHOST をホスト用に、また、環境変数 PGPASSWORD を生成されたトークン用に使用しています。また、*/sample_dir/* は公開キーを含む SSL 証明書ファイルへの完全なパスを示します。

```
export RDSHOST="mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com"
export PGPASSWORD="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 5432 --
region us-west-2 --username jane_doe )"

psql "host=$RDSHOST port=5432 sslmode=verify-full sslrootcert=/sample_dir/global-
bundle.pem dbname=DBName user=jane_doe password=$PGPASSWORD"
```

プロキシ経由で DB クラスターに接続する場合は、「IAM 認証を使用したプロキシへの接続」を参照してください。

IAM 認証および AWS SDK for .NET を使用した DB クラスターへの接続

次に説明するように、AWS SDK for .NET を使用して、Aurora MySQL もしくは Aurora PostgreSQL DB クラスターに接続できます。

前提条件

IAM 認証を使用して DB クラスターに接続するための前提条件は以下のとおりです。

- IAM データベース認証の有効化と無効化
- IAM データベースアクセス用の IAM ポリシーの作成と使用
- IAM 認証を使用したデータベースアカウントの作成

例

以下のコード例で、認証トークンを生成し、それを使用して DB クラスターに接続する方法を示します。

このコードサンプルを実行するには、AWS SDK for .NET サイトにある AWS が必要です。`AWSSDK.CORE` および `AWSSDK.RDS` パッケージが必要です。DB クラスターに接続するには、MariaDB または MySQL 用の MySqlConnector や PostgreSQL 用の Npgsql など、DB エンジン用の .NET データベースコネクタを使用します。

このコードで Aurora MySQL DB クラスターに接続します。必要に応じて以下の可変の値を変更します。

- `server` - アクセス先の DB クラスターのエンドポイント
- `user` - アクセス先のデータベースアカウント
- `database` - アクセス先のデータベース
- `port` - DB クラスターへの接続に使用するポート番号
- `SslMode` - 使用する SSL モード

  `SslMode=Required` を使用すると、SSL 接続で DB クラスターのエンドポイントを SSL 証明書のエンドポイントと照合します。
- `SslCa` - Amazon Aurora のSSL 証明書へのフルパス

  証明書をダウンロードするには、「SSL/TLS を使用した DB クラスターへの接続の暗号化」を参照してください。

> **Note**
>
> DB クラスターエンドポイントの代わりにカスタム Route 53 DNS レコードまたは Aurora カスタムエンドポイントを使用して認証トークンを生成することはできません。

```
using System;
using System.Data;
using MySql.Data;
using MySql.Data.MySqlClient;
using Amazon;

namespace ubuntu
{
  class Program
  {
    static void Main(string[] args)
    {
      var pwd =
 Amazon.RDS.Util.RDSAuthTokenGenerator.GenerateAuthToken(RegionEndpoint.USEast1,
 "mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com", 3306, "jane_doe");
      // for debug only Console.Write("{0}\n", pwd);  //this verifies the token is
 generated

      MySqlConnection conn = new
 MySqlConnection($"server=mysqlcluster.cluster-123456789012.us-
east-1.rds.amazonaws.com;user=jane_doe;database=mydB;port=3306;password={pwd};SslMode=Required;
      conn.Open();

      // Define a query
      MySqlCommand sampleCommand = new MySqlCommand("SHOW DATABASES;", conn);

      // Execute a query
      MySqlDataReader mysqlDataRdr = sampleCommand.ExecuteReader();

      // Read all rows and output the first column in each row
      while (mysqlDataRdr.Read())
        Console.WriteLine(mysqlDataRdr[0]);

      mysqlDataRdr.Close();
      // Close connection
      conn.Close();
    }
  }
}
```

このコードで Aurora PostgreSQL DB クラスターに接続します。

必要に応じて以下の可変の値を変更します。

- `Server` - アクセス先の DB クラスターのエンドポイント
- `User ID` - アクセス先のデータベースアカウント
- `Database` - アクセス先のデータベース
- `Port` - DB クラスターへの接続に使用するポート番号
- `SSL Mode` - 使用する SSL モード

  `SSL Mode=Required` を使用すると、SSL 接続で DB クラスターのエンドポイントを SSL 証明書のエンドポイントと照合します。
- `Root Certificate` - Amazon Aurora のSSL 証明書へのフルパス

  証明書をダウンロードするには、「SSL/TLS を使用した DB クラスターへの接続の暗号化」を参照してください。

> **Note**
>
> DB クラスターエンドポイントの代わりにカスタム Route 53 DNS レコードまたは Aurora カスタムエンドポイントを使用して認証トークンを生成することはできません。

```
using System;
using Npgsql;
using Amazon.RDS.Util;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            var pwd =
 RDSAuthTokenGenerator.GenerateAuthToken("postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com", 5432, "jane_doe");
// for debug only Console.Write("{0}\n", pwd);  //this verifies the token is generated

            NpgsqlConnection conn = new
 NpgsqlConnection($"Server=postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com;User Id=jane_doe;Password={pwd};Database=mydb;SSL
 Mode=Require;Root Certificate=full_path_to_ssl_certificate");
            conn.Open();
```

```
            // Define a query
                   NpgsqlCommand cmd = new NpgsqlCommand("select count(*) FROM
 pg_user", conn);

            // Execute a query
            NpgsqlDataReader dr = cmd.ExecuteReader();

            // Read all rows and output the first column in each row
            while (dr.Read())
                Console.Write("{0}\n", dr[0]);

            // Close connection
            conn.Close();
        }
    }
}
```

プロキシ経由で DB クラスターに接続する場合は、「IAM 認証を使用したプロキシへの接続」を参照してください。

IAM 認証および AWS SDK for Go を使用した DB クラスターへの接続

次に説明するように、AWS SDK for Go を使用して、Aurora MySQL もしくは Aurora PostgreSQL DB クラスターに接続できます。

前提条件

IAM 認証を使用して DB クラスターに接続するための前提条件は以下のとおりです。

- IAM データベース認証の有効化と無効化
- IAM データベースアクセス用の IAM ポリシーの作成と使用
- IAM 認証を使用したデータベースアカウントの作成

例

これらのコードサンプルを実行するには、AWS SDK for Go サイトにある AWS が必要です。

必要に応じて以下の可変の値を変更します。

- dbName - アクセス先のデータベース

- `dbUser` - アクセス先のデータベースアカウント
- `dbHost` - アクセス先の DB クラスターのエンドポイント

> **Note**
>
> DB クラスターエンドポイントの代わりに、カスタムRoute 53 DNS レコードまたは Aurora カスタムエンドポイントを使用して認証トークンを生成することはできません。

- `dbPort` - DB クラスターへの接続に使用するポート番号
- `region` - DB クラスターが実行中の AWS リージョン

さらに、サンプルコード内のインポートされるライブラリがシステムに存在することを確認してください。

> **Important**
>
> このセクションの例では、次のコードを使用して、ローカル環境からデータベースにアクセスする認証情報を提供します。
> `creds := credentials.NewEnvCredentials()`
> Amazon EC2 や Amazon ECS などの AWS のサービスからデータベースにアクセスする場合は、コードを次のコードに置き換えることができます。
> `sess := session.Must(session.NewSession())`
> `creds := sess.Config.Credentials`
> この変更を行う場合は、次のインポートを追加してください。
> `"github.com/aws/aws-sdk-go/aws/session"`

トピック

- IAM 認証と AWS SDK for Go V2 を使用した接続
- IAM 認証と AWS SDK for Go V1 を使用した接続。

IAM 認証と AWS SDK for Go V2 を使用した接続

IAM 認証と AWS SDK for Go V2 を使用して DB クラスターに接続できます

以下のコード例で、認証トークンを生成し、それを使用して DB クラスターに接続する方法を示します。

このコードで Aurora MySQL DB クラスターに接続します。

```
package main

import (
     "context"
     "database/sql"
     "fmt"

     "github.com/aws/aws-sdk-go-v2/config"
     "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
     _ "github.com/go-sql-driver/mysql"
)

func main() {

     var dbName string = "DatabaseName"
     var dbUser string = "DatabaseUser"
     var dbHost string = "mysqlcluster.cluster-123456789012.us-
east-1.rds.amazonaws.com"
     var dbPort int = 3306
     var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
     var region string = "us-east-1"

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
     panic("configuration error: " + err.Error())
    }

    authenticationToken, err := auth.BuildAuthToken(
     context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
    if err != nil {
     panic("failed to create authentication token: " + err.Error())
    }

    dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
        dbUser, authenticationToken, dbEndpoint, dbName,
    )

    db, err := sql.Open("mysql", dsn)
    if err != nil {
        panic(err)
    }
```

```
    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

このコードで Aurora PostgreSQL DB クラスターに接続します。

```
package main

import (
     "context"
     "database/sql"
     "fmt"

     "github.com/aws/aws-sdk-go-v2/config"
     "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
     _ "github.com/lib/pq"
)

func main() {

     var dbName string = "DatabaseName"
     var dbUser string = "DatabaseUser"
     var dbHost string = "postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com"
     var dbPort int = 5432
     var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
     var region string = "us-east-1"

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
     panic("configuration error: " + err.Error())
    }

    authenticationToken, err := auth.BuildAuthToken(
     context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
    if err != nil {
     panic("failed to create authentication token: " + err.Error())
    }

    dsn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s",
        dbHost, dbPort, dbUser, authenticationToken, dbName,
```

```
    )

    db, err := sql.Open("postgres", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

プロキシ経由で DB クラスターに接続する場合は、「IAM 認証を使用したプロキシへの接続」を参照してください。

IAM 認証と AWS SDK for Go V1 を使用した接続。

IAM 認証と AWS SDK for Go V1 を使用して DB クラスターに接続できます

以下のコード例で、認証トークンを生成し、それを使用して DB クラスターに接続する方法を示します。

このコードで Aurora MySQL DB クラスターに接続します。

```
package main

import (
    "database/sql"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go/aws/credentials"
    "github.com/aws/aws-sdk-go/service/rds/rdsutils"
    _ "github.com/go-sql-driver/mysql"
)

func main() {
    dbName := "app"
    dbUser := "jane_doe"
    dbHost := "mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
    dbPort := 3306
    dbEndpoint := fmt.Sprintf("%s:%d", dbHost, dbPort)
```

```
    region := "us-east-1"

    creds := credentials.NewEnvCredentials()
    authToken, err := rdsutils.BuildAuthToken(dbEndpoint, region, dbUser, creds)
    if err != nil {
        panic(err)
    }

    dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
        dbUser, authToken, dbEndpoint, dbName,
    )

    db, err := sql.Open("mysql", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

このコードで Aurora PostgreSQL DB クラスターに接続します。

```
package main

import (
 "database/sql"
 "fmt"

 "github.com/aws/aws-sdk-go/aws/credentials"
 "github.com/aws/aws-sdk-go/service/rds/rdsutils"
 _ "github.com/lib/pq"
)

func main() {
    dbName := "app"
    dbUser := "jane_doe"
    dbHost := "postgresmycluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
    dbPort := 5432
    dbEndpoint := fmt.Sprintf("%s:%d", dbHost, dbPort)
    region := "us-east-1"
```

```
    creds := credentials.NewEnvCredentials()
    authToken, err := rdsutils.BuildAuthToken(dbEndpoint, region, dbUser, creds)
    if err != nil {
        panic(err)
    }

    dsn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s",
        dbHost, dbPort, dbUser, authToken, dbName,
    )

    db, err := sql.Open("postgres", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

プロキシ経由で DB クラスターに接続する場合は、「IAM 認証を使用したプロキシへの接続」を参照してください。

IAM 認証および AWS SDK for Java を使用した DB クラスターへの接続

次に説明するように、AWS SDK for Java を使用して、Aurora MySQL もしくは Aurora PostgreSQL DB クラスターに接続できます。

前提条件

IAM 認証を使用して DB クラスターに接続するための前提条件は以下のとおりです。

- IAM データベース認証の有効化と無効化
- IAM データベースアクセス用の IAM ポリシーの作成と使用
- IAM 認証を使用したデータベースアカウントの作成
- AWS SDK for Java をセットアップする

トピック

- IAM 認証トークンの生成

- IAM 認証トークンを手動で構築する
- DB クラスターへの接続

IAM 認証トークンの生成

AWS SDK for Java を使用してプログラムを作成する場合、RdsIamAuthTokenGenerator クラスを使用して署名付き認証トークンを取得できます。このクラスを使用するには、AWS 認証情報を提供する必要があります。これを行うには、DefaultAWSCredentialsProviderChain クラスのインスタンスを作成します。DefaultAWSCredentialsProviderChain は、デフォルトの認証情報プロバイダチェーンで見つける初期の AWS のアクセスキーとシークレットキーを使用します。AWS アクセスキーの詳細については、「ユーザーのアクセスキーの管理」を参照してください。

> **Note**
>
> DB クラスターエンドポイントの代わりに、カスタム Route 53 DNS レコードまたは Aurora カスタムエンドポイントを使用して認証トークンを生成することはできません。

RdsIamAuthTokenGenerator のインスタンスを作成した後、getAuthToken メソッドを呼び出して、署名済みトークンを取得できます。AWS リージョン、ホスト名、ポート番号、およびユーザー名を指定します。次のコード例はこれを行う方法を示しています。

```
package com.amazonaws.codesamples;

import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.services.rds.auth.GetIamAuthTokenRequest;
import com.amazonaws.services.rds.auth.RdsIamAuthTokenGenerator;

public class GenerateRDSAuthToken {

    public static void main(String[] args) {

     String region = "us-west-2";
     String hostname = "rdsmysql.123456789012.us-west-2.rds.amazonaws.com";
     String port = "3306";
     String username = "jane_doe";

     System.out.println(generateAuthToken(region, hostname, port, username));
    }
```

```
    static String generateAuthToken(String region, String hostName, String port, String
 username) {

     RdsIamAuthTokenGenerator generator = RdsIamAuthTokenGenerator.builder()
      .credentials(new DefaultAWSCredentialsProviderChain())
      .region(region)
      .build();

     String authToken = generator.getAuthToken(
      GetIamAuthTokenRequest.builder()
      .hostname(hostName)
      .port(Integer.parseInt(port))
      .userName(username)
      .build());

     return authToken;
    }

}
```

IAM 認証トークンを手動で構築する

Java では、認証トークンを生成するための最も簡単な方法は、RdsIamAuthTokenGenerator を使用することです。このクラスは、認証トークンを作成した後、AWS 署名バージョン 4 を使用してサインインします。詳細については、AWS 全般のリファレンスの「Signature Version 4 の署名プロセス」を参照してください。

ただし、次のコード例に示すように、認証トークンを手動で構築して署名できます。

```
package com.amazonaws.codesamples;

import com.amazonaws.SdkClientException;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.auth.SigningAlgorithm;
import com.amazonaws.util.BinaryUtils;
import org.apache.commons.lang3.StringUtils;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.Charset;
import java.security.MessageDigest;
import java.text.SimpleDateFormat;
```

```
import java.util.Date;
import java.util.SortedMap;
import java.util.TreeMap;

import static com.amazonaws.auth.internal.SignerConstants.AWS4_TERMINATOR;
import static com.amazonaws.util.StringUtils.UTF8;

public class CreateRDSAuthTokenManually {
    public static String httpMethod = "GET";
    public static String action = "connect";
    public static String canonicalURIParameter = "/";
    public static SortedMap<String, String> canonicalQueryParameters = new TreeMap();
    public static String payload = StringUtils.EMPTY;
    public static String signedHeader = "host";
    public static String algorithm = "AWS4-HMAC-SHA256";
    public static String serviceName = "rds-db";
    public static String requestWithoutSignature;

    public static void main(String[] args) throws Exception {

        String region = "us-west-2";
        String instanceName = "rdsmysql.123456789012.us-west-2.rds.amazonaws.com";
        String port = "3306";
        String username = "jane_doe";

        Date now = new Date();
        String date = new SimpleDateFormat("yyyyMMdd").format(now);
        String dateTimeStamp = new
 SimpleDateFormat("yyyyMMdd'T'HHmmss'Z'").format(now);
       DefaultAWSCredentialsProviderChain creds = new
 DefaultAWSCredentialsProviderChain();
    String awsAccessKey = creds.getCredentials().getAWSAccessKeyId();
    String awsSecretKey = creds.getCredentials().getAWSSecretKey();
       String expiryMinutes = "900";

        System.out.println("Step 1:  Create a canonical request:");
        String canonicalString = createCanonicalString(username, awsAccessKey, date,
 dateTimeStamp, region, expiryMinutes, instanceName, port);
        System.out.println(canonicalString);
        System.out.println();

        System.out.println("Step 2:  Create a string to sign:");
        String stringToSign = createStringToSign(dateTimeStamp, canonicalString,
 awsAccessKey, date, region);
```

```
        System.out.println(stringToSign);
        System.out.println();

        System.out.println("Step 3:  Calculate the signature:");
        String signature = BinaryUtils.toHex(calculateSignature(stringToSign,
newSigningKey(awsSecretKey, date, region, serviceName)));
        System.out.println(signature);
        System.out.println();

        System.out.println("Step 4:  Add the signing info to the request");

        System.out.println(appendSignature(signature));
        System.out.println();

    }

    //Step 1: Create a canonical request date should be in format YYYYMMDD and dateTime
should be in format YYYYMMDDTHHMMSSZ
    public static String createCanonicalString(String user, String accessKey, String
date, String dateTime, String region, String expiryPeriod, String hostName, String
port) throws Exception {
        canonicalQueryParameters.put("Action", action);
        canonicalQueryParameters.put("DBUser", user);
        canonicalQueryParameters.put("X-Amz-Algorithm", "AWS4-HMAC-SHA256");
        canonicalQueryParameters.put("X-Amz-Credential", accessKey + "%2F" + date +
"%2F" + region + "%2F" + serviceName + "%2Faws4_request");
        canonicalQueryParameters.put("X-Amz-Date", dateTime);
        canonicalQueryParameters.put("X-Amz-Expires", expiryPeriod);
        canonicalQueryParameters.put("X-Amz-SignedHeaders", signedHeader);
        String canonicalQueryString = "";
        while(!canonicalQueryParameters.isEmpty()) {
            String currentQueryParameter = canonicalQueryParameters.firstKey();
            String currentQueryParameterValue =
canonicalQueryParameters.remove(currentQueryParameter);
            canonicalQueryString = canonicalQueryString + currentQueryParameter + "=" +
currentQueryParameterValue;
            if (!currentQueryParameter.equals("X-Amz-SignedHeaders")) {
                canonicalQueryString += "&";
            }
        }
        String canonicalHeaders = "host:" + hostName + ":" + port + '\n';
        requestWithoutSignature = hostName + ":" + port + "/?" + canonicalQueryString;

        String hashedPayload = BinaryUtils.toHex(hash(payload));
```

```
        return httpMethod + '\n' + canonicalURIParameter + '\n' + canonicalQueryString
 + '\n' + canonicalHeaders + '\n' + signedHeader + '\n' + hashedPayload;

    }

    //Step 2: Create a string to sign using sig v4
    public static String createStringToSign(String dateTime, String canonicalRequest,
 String accessKey, String date, String region) throws Exception {
        String credentialScope = date + "/" + region + "/" + serviceName + "/
aws4_request";
        return algorithm + '\n' + dateTime + '\n' + credentialScope + '\n' +
 BinaryUtils.toHex(hash(canonicalRequest));

    }

    //Step 3: Calculate signature
    /**
     * Step 3 of the &AWS; Signature version 4 calculation. It involves deriving
     * the signing key and computing the signature. Refer to
     * http://docs.aws.amazon
     * .com/general/latest/gr/sigv4-calculate-signature.html
     */
    public static byte[] calculateSignature(String stringToSign,
                                            byte[] signingKey) {
        return sign(stringToSign.getBytes(Charset.forName("UTF-8")), signingKey,
                SigningAlgorithm.HmacSHA256);
    }

    public static byte[] sign(byte[] data, byte[] key,
                          SigningAlgorithm algorithm) throws SdkClientException {
        try {
            Mac mac = algorithm.getMac();
            mac.init(new SecretKeySpec(key, algorithm.toString()));
            return mac.doFinal(data);
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to calculate a request signature: "
                            + e.getMessage(), e);
        }
    }

    public static byte[] newSigningKey(String secretKey,
                                   String dateStamp, String regionName, String
 serviceName) {
```

```
        byte[] kSecret = ("AWS4" + secretKey).getBytes(Charset.forName("UTF-8"));
        byte[] kDate = sign(dateStamp, kSecret, SigningAlgorithm.HmacSHA256);
        byte[] kRegion = sign(regionName, kDate, SigningAlgorithm.HmacSHA256);
        byte[] kService = sign(serviceName, kRegion,
                SigningAlgorithm.HmacSHA256);
        return sign(AWS4_TERMINATOR, kService, SigningAlgorithm.HmacSHA256);
    }

    public static byte[] sign(String stringData, byte[] key,
                      SigningAlgorithm algorithm) throws SdkClientException {
        try {
            byte[] data = stringData.getBytes(UTF8);
            return sign(data, key, algorithm);
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to calculate a request signature: "
                            + e.getMessage(), e);
        }
    }

    //Step 4: append the signature
    public static String appendSignature(String signature) {
        return requestWithoutSignature + "&X-Amz-Signature=" + signature;
    }

    public static byte[] hash(String s) throws Exception {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            md.update(s.getBytes(UTF8));
            return md.digest();
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to compute hash while signing request: "
                            + e.getMessage(), e);
        }
    }
}
```

DB クラスターへの接続

次のコード例では、認証トークンを生成し、それを使用して Aurora MySQL を実行しているクラスターに接続する方法を示しています。

このコードサンプルを実行するには、AWS SDK for Java サイトにある AWS が必要です。また、以下が必要になります。

- MySQL Connector/J。 このコードの例は `mysql-connector-java-5.1.33-bin.jar` でテストされています。
- AWS リージョンに固有の、Amazon Aurora の中間証明書。(詳細については、SSL/TLS を使用した DB クラスターへの接続の暗号化 を参照してください)。クラスローダーは、実行時にこの Java コード例と同じディレクトリで証明書を探し、クラスローダーがその証明書を見つけられるようにします。
- 必要に応じて以下の可変の値を変更します。
  - `RDS_INSTANCE_HOSTNAME` - アクセス先の DB クラスターのホスト名。
  - `RDS_INSTANCE_PORT` - PostgreSQL DB クラスターへの接続に使用されるポート番号。
  - `REGION_NAME` - DB クラスターが実行中の AWS リージョン
  - `DB_USER` - アクセス先のデータベースアカウント。
  - `SSL_CERTIFICATE` - AWS リージョンに固有の、Amazon Aurora の SSL 証明書。

    AWS リージョンの証明書をダウンロードするには、SSL/TLS を使用した DB クラスターへの接続の暗号化 を参照してください。この Java プログラムファイルと同じディレクトリに SSL 証明書を配置し、クラスローダーが実行時にその証明書を見つけられるようにします。

このコード例では、デフォルトの認証情報プロバイダチェーンから AWS 認証情報を取得します。

> **Note**
>
> セキュリティ上のベストプラクティスとして、ここに示されているプロンプト以外の `DEFAULT_KEY_STORE_PASSWORD` のパスワードを指定してください。

```
package com.amazonaws.samples;

import com.amazonaws.services.rds.auth.RdsIamAuthTokenGenerator;
import com.amazonaws.services.rds.auth.GetIamAuthTokenRequest;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.io.File;
```

```
import java.io.FileOutputStream;
import java.io.InputStream;
import java.security.KeyStore;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Properties;

import java.net.URL;

public class IAMDatabaseAuthenticationTester {
    //&AWS; Credentials of the IAM user with policy enabling IAM Database Authenticated
 access to the db by the db user.
    private static final DefaultAWSCredentialsProviderChain creds = new
 DefaultAWSCredentialsProviderChain();
    private static final String AWS_ACCESS_KEY =
 creds.getCredentials().getAWSAccessKeyId();
    private static final String AWS_SECRET_KEY =
 creds.getCredentials().getAWSSecretKey();

    //Configuration parameters for the generation of the IAM Database Authentication
 token
    private static final String RDS_INSTANCE_HOSTNAME = "rdsmysql.123456789012.us-
west-2.rds.amazonaws.com";
    private static final int RDS_INSTANCE_PORT = 3306;
    private static final String REGION_NAME = "us-west-2";
    private static final String DB_USER = "jane_doe";
    private static final String JDBC_URL = "jdbc:mysql://" + RDS_INSTANCE_HOSTNAME +
 ":" + RDS_INSTANCE_PORT;

    private static final String SSL_CERTIFICATE = "rds-ca-2019-us-west-2.pem";

    private static final String KEY_STORE_TYPE = "JKS";
    private static final String KEY_STORE_PROVIDER = "SUN";
    private static final String KEY_STORE_FILE_PREFIX = "sys-connect-via-ssl-test-
cacerts";
    private static final String KEY_STORE_FILE_SUFFIX = ".jks";
    private static final String DEFAULT_KEY_STORE_PASSWORD = "changeit";

    public static void main(String[] args) throws Exception {
```

```
        //get the connection
        Connection connection = getDBConnectionUsingIam();

        //verify the connection is successful
        Statement stmt= connection.createStatement();
        ResultSet rs=stmt.executeQuery("SELECT 'Success!' FROM DUAL;");
        while (rs.next()) {
             String id = rs.getString(1);
            System.out.println(id); //Should print "Success!"
        }

        //close the connection
        stmt.close();
        connection.close();

        clearSslProperties();

    }

    /**
     * This method returns a connection to the db instance authenticated using IAM
Database Authentication
     * @return
     * @throws Exception
     */
    private static Connection getDBConnectionUsingIam() throws Exception {
        setSslProperties();
        return DriverManager.getConnection(JDBC_URL, setMySqlConnectionProperties());
    }

    /**
     * This method sets the mysql connection properties which includes the IAM Database
Authentication token
     * as the password. It also specifies that SSL verification is required.
     * @return
     */
    private static Properties setMySqlConnectionProperties() {
        Properties mysqlConnectionProperties = new Properties();
        mysqlConnectionProperties.setProperty("verifyServerCertificate","true");
        mysqlConnectionProperties.setProperty("useSSL", "true");
        mysqlConnectionProperties.setProperty("user",DB_USER);
        mysqlConnectionProperties.setProperty("password",generateAuthToken());
        return mysqlConnectionProperties;
    }
```

```
    /**
     * This method generates the IAM Auth Token.
     * An example IAM Auth Token would look like follows:
     * btusi123.cmz7kenwo2ye.rds.cn-north-1.amazonaws.com.cn:3306/?
Action=connect&DBUser=iamtestuser&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Date=20171003T010726Z&X-Amz-SignedHeaders=host&X-Amz-Expires=899&X-Amz-
Credential=AKIAPFXHGVDI5RNFO4AQ%2F20171003%2Fcn-north-1%2Frds-db%2Faws4_request&X-Amz-
Signature=f9f45ef96c1f770cdad11a53e33ffa4c3730bc03fdee820cfdf1322eed15483b
     * @return
     */
    private static String generateAuthToken() {
        BasicAWSCredentials awsCredentials = new BasicAWSCredentials(AWS_ACCESS_KEY,
 AWS_SECRET_KEY);

        RdsIamAuthTokenGenerator generator = RdsIamAuthTokenGenerator.builder()
                .credentials(new
 AWSStaticCredentialsProvider(awsCredentials)).region(REGION_NAME).build();
        return generator.getAuthToken(GetIamAuthTokenRequest.builder()

 .hostname(RDS_INSTANCE_HOSTNAME).port(RDS_INSTANCE_PORT).userName(DB_USER).build());
    }

    /**
     * This method sets the SSL properties which specify the key store file, its type
 and password:
     * @throws Exception
     */
    private static void setSslProperties() throws Exception {
        System.setProperty("javax.net.ssl.trustStore", createKeyStoreFile());
        System.setProperty("javax.net.ssl.trustStoreType", KEY_STORE_TYPE);
        System.setProperty("javax.net.ssl.trustStorePassword",
 DEFAULT_KEY_STORE_PASSWORD);
    }

    /**
     * This method returns the path of the Key Store File needed for the SSL
 verification during the IAM Database Authentication to
     * the db instance.
     * @return
     * @throws Exception
     */
    private static String createKeyStoreFile() throws Exception {
        return createKeyStoreFile(createCertificate()).getPath();
```

```
    }

    /**
     *  This method generates the SSL certificate
     * @return
     * @throws Exception
     */
    private static X509Certificate createCertificate() throws Exception {
        CertificateFactory certFactory = CertificateFactory.getInstance("X.509");
        URL url = new File(SSL_CERTIFICATE).toURI().toURL();
        if (url == null) {
            throw new Exception();
        }
        try (InputStream certInputStream = url.openStream()) {
            return (X509Certificate) certFactory.generateCertificate(certInputStream);
        }
    }

    /**
     * This method creates the Key Store File
     * @param rootX509Certificate - the SSL certificate to be stored in the KeyStore
     * @return
     * @throws Exception
     */
    private static File createKeyStoreFile(X509Certificate rootX509Certificate) throws
Exception {
        File keyStoreFile = File.createTempFile(KEY_STORE_FILE_PREFIX,
KEY_STORE_FILE_SUFFIX);
        try (FileOutputStream fos = new FileOutputStream(keyStoreFile.getPath())) {
            KeyStore ks = KeyStore.getInstance(KEY_STORE_TYPE, KEY_STORE_PROVIDER);
            ks.load(null);
            ks.setCertificateEntry("rootCaCertificate", rootX509Certificate);
            ks.store(fos, DEFAULT_KEY_STORE_PASSWORD.toCharArray());
        }
        return keyStoreFile;
    }

    /**
     * This method clears the SSL properties.
     * @throws Exception
     */
    private static void clearSslProperties() throws Exception {
           System.clearProperty("javax.net.ssl.trustStore");
           System.clearProperty("javax.net.ssl.trustStoreType");
```

```
            System.clearProperty("javax.net.ssl.trustStorePassword");
    }

}
```

プロキシ経由で DB クラスターに接続する場合は、「IAM 認証を使用したプロキシへの接続」を参照してください。

IAM 認証および AWS SDK for Python (Boto3) を使用した DB クラスターへの接続

次に説明するように、AWS SDK for Python (Boto3) を使用して、Aurora MySQL もしくは Aurora PostgreSQL DB クラスターに接続できます。

前提条件

IAM 認証を使用して DB クラスターに接続するための前提条件は以下のとおりです。

- IAM データベース認証の有効化と無効化
- IAM データベースアクセス用の IAM ポリシーの作成と使用
- IAM 認証を使用したデータベースアカウントの作成

さらに、サンプルコード内のインポートされるライブラリがシステムに存在することを確認してください。

例

コード例では、共有認証情報のプロファイルを使用します。認証情報の指定については、AWS SDK for Python (Boto3) ドキュメントの「認証情報」を参照してください。

以下のコード例で、認証トークンを生成し、それを使用して DB クラスターに接続する方法を示します。

このコードサンプルを実行するには、AWS SDK for Python (Boto3) サイトにある AWS が必要です。

必要に応じて以下の可変の値を変更します。

- `ENDPOINT` - アクセス先の DB クラスターのエンドポイント
- `PORT` - DB クラスターへの接続に使用するポート番号
- `USER` - アクセス先のデータベースアカウント
- `REGION` - DB クラスターが実行中の AWS リージョン

- DBNAME - アクセス先のデータベース
- SSLCERTIFICATE - Amazon Aurora の SSL 証明書へのフルパス

  ssl_ca を使用する場合、SSL 証明書を指定します。SSL 証明書をダウンロードするには SSL/TLS を使用した DB クラスターへの接続の暗号化 を参照ください。

> **Note**
>
> DB クラスターエンドポイントの代わりにカスタム Route 53 DNS レコードまたは Aurora カスタムエンドポイントを使用して認証トークンを生成することはできません。

このコードで Aurora MySQL DB クラスターに接続します。

このコードを実行する前に、Python Package Index の手順に従って PyMySQL ドライバーをインストールしてください。

```
import pymysql
import sys
import boto3
import os

ENDPOINT="mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
PORT="3306"
USER="jane_doe"
REGION="us-east-1"
DBNAME="mydb"
os.environ['LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN'] = '1'

#gets the credentials from .aws/credentials
session = boto3.Session(profile_name='default')
client = session.client('rds')

token = client.generate_db_auth_token(DBHostname=ENDPOINT, Port=PORT, DBUsername=USER,
 Region=REGION)

try:
    conn =  pymysql.connect(host=ENDPOINT, user=USER, passwd=token, port=PORT,
 database=DBNAME, ssl_ca='SSLCERTIFICATE')
    cur = conn.cursor()
    cur.execute("""SELECT now()""")
```

```
    query_results = cur.fetchall()
    print(query_results)
except Exception as e:
    print("Database connection failed due to {}".format(e))
```

このコードで Aurora PostgreSQL DB クラスターに接続します。

このコードを実行する前に、Psycopg documentation の手順に従って psycopg2 をインストールしてください。

```
import psycopg2
import sys
import boto3
import os

ENDPOINT="postgresmycluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
PORT="5432"
USER="jane_doe"
REGION="us-east-1"
DBNAME="mydb"

#gets the credentials from .aws/credentials
session = boto3.Session(profile_name='RDSCreds')
client = session.client('rds')

token = client.generate_db_auth_token(DBHostname=ENDPOINT, Port=PORT, DBUsername=USER,
 Region=REGION)

try:
    conn = psycopg2.connect(host=ENDPOINT, port=PORT, database=DBNAME, user=USER,
 password=token, sslrootcert="SSLCERTIFICATE")
    cur = conn.cursor()
    cur.execute("""SELECT now()""")
    query_results = cur.fetchall()
    print(query_results)
except Exception as e:
    print("Database connection failed due to {}".format(e))
```

プロキシ経由で DB クラスターに接続する場合は、「IAM 認証を使用したプロキシへの接続」を参照してください。

Amazon Aurora のアイデンティティおよびアクセスのトラブルシューティング

次の情報は、Aurora と IAM の使用に伴って発生する可能性がある一般的な問題の診断や修復に役立ちます。

トピック

- Aurora でアクションを実行する権限がありません。
- iam:PassRole を実行する権限がない
- 自分の AWS アカウントの外部のユーザーに Aurora リソースへのアクセスを許可したい

Aurora でアクションを実行する権限がありません。

AWS Management Console から、アクションを実行することが認可されていないと通知された場合、管理者に問い合わせ、サポートを依頼する必要があります。管理者は、サインイン認証情報を提供した担当者です。

以下の例のエラーは、`mateojackson` ユーザーがコンソールを使用して、*######*の詳細を表示しようとしましたが、`rds:`*`GetWidget`* アクセス許可がない場合に発生します。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 rds:GetWidget on resource: my-example-widget
```

この場合、Mateo は、*`my-example-widget`* アクションを使用して `rds:`*`GetWidget`* リソースにアクセスできるように、ポリシーの更新を管理者に依頼します。

iam:PassRole を実行する権限がない

`iam:PassRole` アクションを実行する権限がないというエラーが表示された場合、管理者にお問い合わせ、サポートを依頼する必要があります。管理者は、サインイン認証情報を提供した担当者です。Aurora にロールを渡すことができるようにポリシーを更新するよう、管理者に依頼します。

一部の AWS サービスでは、新しいサービスロールまたはサービスリンクロールを作成せずに、既存のロールをサービスに渡すことができます。そのためには、サービスにロールを渡す許可が必要です。

以下の例のエラーは、`marymajor` という名前のユーザーがコンソールを使用して Aurora でアクションを実行しようとした場合に発生します。ただし、アクションには、サービスロールによって

サービスに許可が付与されている必要があります。Mary には、ロールをサービスに渡す許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary は `iam:PassRole` アクションの実行が許可されるように、担当の管理者にポリシーの更新を依頼します。

自分の AWS アカウントの外部のユーザーに Aurora リソースへのアクセスを許可したい

他のアカウントのユーザーや組織外のユーザーが、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定することができます。リソースベースのポリシーまたはアクセス制御リスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください。

- Aurora でこれらの機能がサポートされるかどうかを確認するには、「Amazon Aurora と IAM の連携」を参照してください。
- 所有している AWS アカウント全体のリソースへのアクセス権を提供する方法については、IAM ユーザーガイドの「所有している別の AWS アカウントへのアクセス権を IAM ユーザーに提供」を参照してください。
- リソースへのアクセスをサードパーティーの AWS アカウントに提供する方法については、IAM ユーザーガイドの「サードパーティーが所有する AWS アカウントへのアクセスの提供」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、 IAM ユーザーガイドの外部認証されたユーザーへのアクセスの提供 (ID フェデレーション) を参照してください。
- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いの詳細については、「IAM ユーザーガイド」の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

Amazon Aurora でのログ記録とモニタリング

モニタリングは、Amazon Aurora と AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。マルチポイント障害が発生した場合は、その障害をより簡単にデバッグ

できるように、AWS ソリューションのすべての部分からモニタリングデータを収集する必要があります。AWS には、Amazon Aurora リソースをモニタリングし、潜在的なインシデントに対応するための複数のツールが用意されています。

Amazon CloudWatch アラーム

Amazon CloudWatch アラームを使用して、指定した期間中、1 つのメトリクスをモニタリングします。メトリクスが特定の閾値を超えると、Amazon SNS トピックまたは AWS Auto Scaling ポリシーに通知が送信されます。CloudWatch アラームは、特定の状態にあるという理由ではアクションを呼び出しません。状態が変わり、それが指定した期間だけ維持される必要があります。

AWS CloudTrail ログ

CloudTrail では、Amazon Aurora のユーザー、ロール、または AWS のサービスによって実行されたアクションの記録を確認できます。CloudTrail は、コンソールからの呼び出しと Amazon RDS API オペレーションへのコード呼び出しを含む、Amazon Aurora のすべての API コールをイベントとしてキャプチャします。CloudTrail によって収集された情報を使用して、リクエストの作成元の IP アドレス、リクエストの実行者、リクエストの実行日時などの詳細を調べて、Amazon Aurora に対してどのようなリクエストが行われたかを判断できます。詳細については、「AWS CloudTrail での Amazon Aurora API コールのモニタリング」を参照してください。

拡張モニタリング

Amazon Aurora には、DB クラスターが実行されているオペレーティングシステム (OS) のリアルタイムのメトリクスが用意されています。コンソールで DB クラスターのメトリクスを表示したり、選択したモニタリングシステムで Amazon CloudWatch Logs からの拡張モニタリング JSON 出力を使用したりできます。詳細については、「拡張モニタリングを使用した OS メトリクスのモニタリング」を参照してください。

Amazon RDS Performance Insights

Performance Insights は、既存の Amazon Aurora モニタリング機能を拡張して、データベースのパフォーマンスを明確にし、これに影響を与えるあらゆる問題を分析しやすくします。Performance Insights ダッシュボードを使用してデータベースロードを視覚化したり、ロードを待機、SQL ステートメント、ホスト、ユーザー別にフィルタリングしたりできます。詳細については、「Amazon Aurora での Performance Insights を使用したDB 負荷のモニタリング」を参照してください。

データベースのログ

データベースログを表示、ダウンロード、モニタリングするには、AWS Management Console、AWS CLI、または RDS API を使用します。詳細については、「Amazon Aurora ログファイルのモニタリング」を参照してください。

Amazon Aurora の推奨事項

Amazon Aurora は、データベースリソースに対して自動化された推奨事項を示します。これらの推奨事項では、DB クラスター設定、使用状況、パフォーマンスデータを分析して、ベストプラクティスガイダンスを提供します。詳細については、「Amazon Aurora の推奨事項の表示とこれらに対する対応」を参照してください。

Amazon Aurora イベントの通知

Amazon Aurora では、Amazon Aurora のイベントが発生したときに、Amazon Simple Notification Service (Amazon SNS) を使用して通知を送信します。これらの通知は、AWS リージョンの Amazon SNS でサポートされているすべての通知の形式を使うことができます (E メール、テキストメッセージ、HTTP エンドポイントの呼び出しなど)。詳細については、「Amazon RDS イベント通知の操作」を参照してください。

AWS Trusted Advisor

Trusted Advisor は、AWS の数十万のお客様にサービスを提供することにより得られた、運用実績から学んだベストプラクティスを活用しています。Trusted Advisor はお客様の AWS 環境を検査し、システムの可用性とパフォーマンスを向上させたりセキュリティギャップを埋めたりする機会がある場合には、推奨事項を作成します。すべての AWS のお客様は、Trusted Advisor の 5 つのチェックにアクセスできます。ビジネスまたはエンタープライズサポートプランをご利用のお客様は、すべての Trusted Advisor チェックを表示できます。

Trusted Advisor には、以下を対象とした Amazon Aurora 関連のチェックがあります。

- Amazon Aurora アイドル DB インスタンス
- Amazon Aurora セキュリティグループのアクセスリスク
- Amazon Aurora バックアップ
- Amazon Aurora マルチ AZ
- Aurora DB インスタンスのアクセシビリティ

これらのチェックの詳細については、「Trusted Advisor のベストプラクティス (チェック)」を参照してください。

データベースアクティビティストリーミング

データベースアクティビティストリーミングで、データベースアクティビティストリーミングへの DBA アクセスを制御することによって、データベースを内部の脅威から保護することができます。そのため、データベースアクティビティストリーミングの収集、転送、格納、およびその後の処理は、データベースを管理する DBA のアクセスを超えています。データベースアクティビティストリーミングでは、データベースを保護し、コンプライアンスと規制の要件を満たすことができます。詳細については、「データベースアクティビティストリームを使用した Amazon Aurora のモニタリング」を参照してください。

Aurora のモニタリングの詳細については、「Amazon Aurora クラスターでのメトリクスのモニタリング」を参照してください。

Amazon Aurora のコンプライアンス検証

サードパーティーの監査者は、複数の AWS コンプライアンスプログラムの一環として Amazon Aurora のセキュリティとコンプライアンスを評価します。このプログラムには、SOC、PCI、FedRAMP、HIPAA などがあります。

特定のコンプライアンスプログラムの範囲内の AWS サービスのリストについては、「コンプライアンスプログラムによる AWS 対象範囲内のサービス」を参照してください。一般的な情報については、「AWS コンプライアンスプログラム」を参照してください。

AWS Artifact を使用して、サードパーティーの監査レポートをダウンロードできます。詳細については、「AWS Artifact のレポートのダウンロード」を参照してください。

Amazon Aurora を使用する際のお客様のコンプライアンス責任は、データの機密性、組織のコンプライアンス目的、適用法規によって決まります。AWS は、コンプライアンスに役立つ以下のリソースを提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド — これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに焦点を当てたベースライン環境を AWS にデプロイするためのステップを示します。
- 「Architecting for HIPAA Security and Compliance on Amazon Web Services」(Amazon Web Services での HIPAA のセキュリティとコンプライアンスのためのアーキテクチャ) – このホワイトペーパーは、企業が AWS を使用して HIPAA 準拠のアプリケーションを作成する方法を説明しています。
- AWS コンプライアンスのリソース - お客様の業界や地域に当てはまる可能性のあるワークブックやガイドのコレクションです。
- AWS Config - この AWS サービスでは、自社プラクティス、業界ガイドライン、および規制に対するリソースの設定の準拠状態を評価します。
- AWS Security Hub – この AWS のサービス は、AWS 内のセキュリティ状態の包括的なビューを提供します。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールのリストについては、「Security Hub のコントロールリファレンス」を参照してください。

Amazon Aurora の耐障害性

AWS のグローバルインフラストラクチャは AWS リージョンとアベイラビリティーゾーンを中心に構築されます。AWSリージョンには、低レイテンシー、高いスループット、そして高度の冗長ネットワークで接続されている複数の物理的に独立し隔離されたアベイラビリティーゾーンがあります。アベイラビリティーゾーンでは、アベイラビリティーゾーン間で中断せずに、自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、耐障害性、および拡張性が優れています。

AWS リージョンとアベイラビリティーゾーンの詳細については、「AWS グローバルインフラストラクチャ」を参照してください。

Aurora では、AWS グローバルインフラストラクチャに加えて、データの耐障害性とバックアップのニーズに対応するための機能を提供しています。

バックアップと復元

Aurora は、クラスターボリュームを自動的にバックアップし、バックアップ保持期間中、復元データを保持します。Aurora のバックアップは連続的かつ増分的であるため、バックアップ保持期間内の任意の時点にすばやく復元できます。バックアップデータが書き込まれるときに、データベースサービスのパフォーマンスに影響が出たり、中断が発生したりすることはありません。DB クラスターを作成または変更するときに、バックアップ保持期間 (1 ~ 35 日) を指定できます。

バックアップ保持期間を超えたバックアップを保持する場合は、クラスターボリュームの中にデータのスナップショットを作成できます。Aurora は、バックアップ保持期間全体にわたって増分復元データを保持します。そのため、バックアップ保持期間を超えて保持するデータのスナップショットを作成するだけで済みます。スナップショットから新しい DB クラスターを作成できます。

Aurora が保持するバックアップデータから、または保存した DB クラスターのスナップショットから、新しい Aurora DB クラスターを作成することで、データを回復できます。DB クラスターの新しいコピーは、バックアップデータから作成し、バックアップ保持期間内の任意の時点にすばやく復元できます。バックアップ保持期間中の Aurora バックアップが継続的かつ増分的であることは、復元時間を短縮するためにデータのスナップショットを頻繁に作成する必要がないことを意味します。

詳細については、「Amazon Aurora DB クラスターのバックアップと復元」を参照してください。

レプリケーション

Aurora レプリカは、Aurora DB クラスター内の独立したエンドポイントであり、読み取りオペレーションのスケーリングと可用性の向上に最適です。最大 15 個の Aurora レプリカを、1 つの AWS リージョンの中で DB クラスターが処理するアベイラビリティーゾーン全体に分散できます。DB クラスターボリュームは DB クラスターのデータの複数のコピーで構成されます。ただし、クラスターボリュームのデータは、DB クラスターのプライマリ DB インスタンスおよび Aurora レプリカに対する単一の論理的なボリュームとして表されます。プライマリインスタンスが失敗した場合、Aurora レプリカはプライマリ DB インスタンスに昇格されます。

また、Aurora は、Aurora MySQL および Aurora PostgreSQL 特有のレプリケーションオプションもサポートしています。

詳細については、「Amazon Aurora でのレプリケーション」を参照してください。

フェイルオーバー

Aurora は、単一の AWS リージョン内の複数のアベイラビリティーゾーンにまたがる DB クラスターにデータのコピーを保存します。これは、DB クラスターの DB インスタンスが複数のアベイラビリティーゾーンにまたがっているかどうかにかかわらず保存されます。アベイラビリティーゾーンにまたがって Aurora レプリカを作成すると、それらは Aurora によって、自動的にプロビジョニングおよび維持されます。プライマリ DB インスタンスは複数のアベイラビリティーゾーンにまたがって、Aurora レプリカに同期してレプリケートされます。これにより、データの冗長性を提供し、I/O のフリーズを防ぎ、システムバックアップの間のレイテンシーの急上昇を最小限に抑えることができます。高可用性を備えた DB クラスターを実行すると、計画されたシステムメンテナンス中の可用性が向上し、障害とアベイラビリティーゾーンの中断からデータベースを保護できます。

詳しくは、「Amazon Aurora の高可用性」を参照してください。

Amazon Aurora でのインフラストラクチャセキュリティ

マネージドサービスとして、Amazon Relational Database Service は AWS グローバルネットワークセキュリティによって保護されています。AWSセキュリティサービスと AWS がインフラストラクチャを保護する方法については、「AWS クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには、「セキュリティの柱 - AWS Well-Architected Framework」の「インフラストラクチャ保護」を参照してください。

ネットワーク経由で Amazon RDS にアクセスするには、AWS が発行した API コールを使用します。クライアントは以下をサポートする必要があります。

- Transport Layer Security (TLS) TLS 1.2 および TLS 1.3 をお勧めします。
- DHE (Ephemeral Diffie-Hellman) や ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートです。これらのモードは、Java 7 以降など、最近のほとんどのシステムでサポートされています。

また、リクエストは、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時セキュリティ認証情報を生成し、リクエストに署名することもできます。

また、Aurora には、インフラストラクチャのセキュリティをサポートする機能もあります。

セキュリティグループ

セキュリティグループにより DB クラスターに対する送受信トラフィックへのアクセスを制御します。デフォルトでは、ネットワークアクセスは DB クラスターに対してオフになっています。セキュリティグループで IP アドレス範囲、ポート、またはセキュリティグループからのアクセスを許可するルールを指定できます。Ingress ルールを設定したら、そのセキュリティグループに関連付けられているすべての DB クラスターに、同じルールが適用されます。

詳しくは、「セキュリティグループによるアクセス制御」を参照してください。

パブリックアクセシビリティ

Amazon VPC サービスに基づき、仮想プライベートクラウド (VPC) 内で DB インスタンスを起動する場合は、そのインスタンスのパブリックアクセシビリティをオンまたはオフにすることができます。作成した DB インスタンスにパブリック IP アドレスに解決される DNS 名を含むかど

うかを指定するには、Public accessibility パラメータを使用します。このパラメータを使用することで、DB インスタンスに対するパブリックアクセスがあるかどうかを指定することができます。Public accessibility パラメータを変更することによって、DB インスタンスのパブリックアクセス可能性をオンまたはオフにすることができます。

詳細については、「VPC 内の DB クラスターをインターネットから隠す」を参照してください。

> **Note**
>
> DB インスタンスが VPC 内にあるがパブリックアクセス可能でない場合は、AWS Site-to-Site VPN 接続や AWS Direct Connect 接続を使用してプライベートネットワークからアクセスすることもできます。詳しくは、「インターネットトラフィックのプライバシー」を参照してください。

Amazon RDS API とインターフェイス VPC エンドポイント (AWS PrivateLink)

インターフェイス VPC エンドポイントを作成することで、VPC と Amazon RDS API エンドポイント間にプライベート接続を確立できます。インターフェイスエンドポイントは を使用しますAWS PrivateLink

AWS PrivateLink を使用すると、インターネットゲートウェイ、NAT デバイス、VPN 接続、または AWS Direct Connect 接続なしで、Amazon RDS API オペレーションにプライベートにアクセスできます。VPC 内の DB インスタンスは、DB インスタンス および DB クラスターを起動、変更、または終了するための Amazon RDS API エンドポイントとの通信に、パブリック IP アドレスを必要としません。また、RDS API オペレーションの使用にも、パブリック IP アドレスを必要としません。VPC と Amazon RDS 間のトラフィックは、Amazon ネットワークを離れません。

各インターフェイスエンドポイントは、サブネット内の 1 つ以上の Elastic Network Interface によって表されます。Elastic Network Interface の詳細については、Amazon EC2 ユーザーガイドの「Elastic Network Interface」を参照してください。

VPC エンドポイントの詳細については、Amazon VPC ユーザーガイドの「インターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。RDS API オペレーションの詳細については、Amazon RDS API リファレンスを参照してください。

DB クラスターへの接続には、インターフェイス VPC エンドポイントは必要ありません。詳細については、「VPC の DB クラスターにアクセスするシナリオ」を参照してください。

VPC エンドポイントに関する考慮事項

Amazon RDS API エンドポイントのインターフェイス VPC エンドポイントを設定する前に、Amazon VPC ユーザーガイド の「インターフェイスエンドポイントのプロパティと制限」を確認してください。

Amazon Aurora リソースの管理に関連するすべての RDS API オペレーションは、AWS PrivateLink を使用して VPC から利用することができます。

VPC エンドポイントポリシーは RDS API エンドポイントでサポートされます。デフォルトでは、エンドポイント経由で RDS API オペレーションへのフルアクセスが許可されます。詳細については、Amazon VPC ユーザーガイドの「VPC エンドポイントによるサービスのアクセス制御」を参照してください。

可用性

現在、Amazon RDS API は、次の AWS リージョンで VPC エンドポイントをサポートしています。

- 米国東部 (オハイオ)
- 米国東部 (バージニア北部)
- 米国西部 (北カリフォルニア)
- 米国西部 (オレゴン)
- アフリカ (ケープタウン)
- アジアパシフィック (香港)
- アジアパシフィック (ムンバイ)
- アジアパシフィック (大阪)
- アジアパシフィック (ソウル)
- アジアパシフィック (シンガポール)
- アジアパシフィック (シドニー)
- アジアパシフィック (東京)
- カナダ (中部)
- カナダ西部 (カルガリー)
- 中国 (北京)
- 中国 (寧夏)
- 欧州 (フランクフルト)
- 欧州 (チューリッヒ)
- 欧州 (アイルランド)
- 欧州 (ロンドン)
- 欧州 (パリ)
- 欧州 (ストックホルム)
- 欧州 (ミラノ)
- イスラエル (テルアビブ)
- 中東 (バーレーン)
- 南米 (サンパウロ)
- AWS GovCloud (米国東部)
- AWS GovCloud (米国西部)

Amazon RDS API 用のインターフェイス VPC エンドポイントの作成

Amazon RDS API 用の VPC エンドポイントは、Amazon VPC コンソールまたは AWS Command Line Interface (AWS CLI) で作成できます。詳細については、Amazon VPC ユーザーガイドのインターフェイスエンドポイントの作成を参照してください。

サービス名 `com.amazonaws.region.rds` を使用して、Amazon RDS API の VPC エンドポイントを作成します。

中国の AWS リージョンを除き、エンドポイントでプライベート DNS を有効にすると、AWS リージョンのデフォルト DNS 名 (`rds.us-east-1.amazonaws.com` など) を使用して、VPC エンドポイントで Amazon RDS に API リクエストを行うことができます。中国 (北京) および 中国 (寧夏) AWS リージョンの場合、それぞれ `rds-api.cn-north-1.amazonaws.com.cn` および `rds-api.cn-northwest-1.amazonaws.com.cn` を使用して VPC エンドポイントで API リクエストを行うことができます。

詳細については、「Amazon VPC ユーザーガイド」の「インターフェイスエンドポイントを介したサービスへのアクセス」を参照してください。

Amazon RDS API 用の VPC エンドポイントポリシーの作成

VPC エンドポイントに Amazon RDS API へのアクセスを制御するエンドポイントポリシーをアタッチできます。このポリシーでは、以下の情報を指定します。

- アクションを実行できるプリンシパル。
- 実行可能なアクション。
- このアクションを実行できるリソース。

詳細については、Amazon VPC ユーザーガイドの「VPC エンドポイントによるサービスのアクセス制御」を参照してください。

例: Amazon RDS API アクションの VPC エンドポイントポリシー

Amazon RDS API のエンドポイントポリシーの例を次に示します。このポリシーは、エンドポイントにアタッチされると、すべてのリソースのすべてのプリンシパルに対して、登録されている Amazon RDS API アクションへのアクセスを許可します。

```
{
```

```
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBInstance",
            "rds:ModifyDBInstance",
            "rds:CreateDBSnapshot"
         ],
         "Resource":"*"
      }
   ]
}
```

例: 指定した AWS アカウントからのすべてのアクセスを拒否する VPC エンドポイントポリシー

以下の VPC エンドポイントポリシーは、AWS アカウント 123456789012 からリソースへのエンドポイントを使用したすべてのアクセスを拒否します。このポリシーは、他のアカウントからのすべてのアクションを許可します。

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
      "Resource": "*",
      "Principal": { "AWS": [ "123456789012" ] }
     }
   ]
}
```

Amazon Aurora のセキュリティのベストプラクティス

AWS Identity and Access Management (IAM) アカウントを使用して、Amazon RDS API オペレーション、特に Amazon Aurora リソースの作成、変更、削除を行うオペレーションへのアクセスを制

御します。そのようなリソースには、DB クラスター、セキュリティグループ、およびパラメータグループなどがあります。また、IAM を使用して、DB クラスターのバックアップや復元など、一般的な管理アクションを実行するアクションも制御します。

- Amazon Aurora リソースを管理するユーザー (本人を含む) ごとに個別のユーザーを作成します。Amazon Aurora リソースの管理には、AWS ルート認証情報を使用しないでください。
- それぞれの職務の実行に最低限必要になる一連のアクセス許可を各ユーザーに付与します。
- IAM グループを使用して、複数のユーザーのアクセス許可を効果的に管理します。
- IAM 認証情報のローテーションを定期的に行います。
- Amazon Aurora のシークレットが自動的にローテーションされるように、AWS Secrets Manager を設定します。詳細については、AWS Secrets Manager ユーザーガイドの「AWS Secrets Manager シークレットのローテーション」を参照してください。認証情報は、AWS Secrets Manager プログラムから取得することもできます。詳細については、AWS Secrets Manager ユーザーガイドの「シークレット値の取得」を参照してください。

Amazon Aurora でのセキュリティの詳細については、「Amazon Aurora でのセキュリティ」を参照してください。IAM の詳細については、「AWS Identity and Access Management」を参照してください。IAM のベストプラクティスについては、「IAM のベストプラクティス」を参照してください。

AWS Security Hub は、セキュリティコントロールを使用してリソース設定とセキュリティ標準を評価し、お客様がさまざまなコンプライアンスフレームワークに準拠できるようサポートします。Security Hub を使用して RDS リソースを評価する方法の詳細については、「AWS Security Hub ユーザーガイド」の「Amazon リレーショナルデータベースサービスコントロール」を参照してください。

Security Hub を使用して、セキュリティのベストプラクティスに関連する RDS の使用状況をモニタリングできます。詳細については、「What is AWS Security Hub?」を参照してください。

AWS Management Console、AWS CLI、RDS API を使用して、マスターユーザーのパスワードを変更します。SQL クライアントなどの別のツールを使用する場合、マスターユーザーのパスワードを変更すると、ユーザーの権限が意図せずに取り消される可能性があります。

Amazon GuardDuty は、Amazon RDS ログインアクティビティなど、さまざまなデータソースを分析および処理する継続的セキュリティモニタリングサービスです。脅威インテリジェンスフィードと機械学習を使用して、AWS 環境内の予期しない、不正の可能性がある、不審なログイン動作や、悪意のあるアクティビティを特定します。

Amazon GuardDuty RDS Protection がデータベースへの脅威となる可能性がある不審な試行や異常なログイン試行を検出すると、GuardDuty は新しい検出結果を生成し、侵害の可能性があるデータベースに関する詳細を示します。詳細については、「Amazon GuardDuty RDS Protection による脅威のモニタリング」を参照してください。

セキュリティグループによるアクセス制御

VPC セキュリティグループにより DB クラスターに対する送受信トラフィックへのアクセスを制御します。デフォルトでは、ネットワークアクセスは DB クラスターに対してオフになっています。セキュリティグループで IP アドレス範囲、ポート、またはセキュリティグループからのアクセスを許可するルールを指定できます。Ingress ルールを設定したら、そのセキュリティグループに関連付けられているすべての DB クラスターに、同じルールが適用されます。セキュリティグループでは最大 20 のルールを指定できます。

VPC セキュリティグループの概要

VPC セキュリティグループの各ルールにより、特定のソースがその VPC セキュリティグループに関連付けられている VPC 内の DB クラスターにアクセスできるようになります。ソースとしては、アドレスの範囲 (203.0.113.0/24 など) または別の VPC セキュリティグループを指定できます。VPC セキュリティグループをソースとして指定すると、ソース VPC セキュリティグループを使用するすべてのインスタンス (通常はアプリケーションサーバー) からの受信トラフィックを許可することになります。VPC セキュリティグループには、インバウンドトラフィックとアウトバウンドトラフィック両方を制御するルールを追加できます。ただし、アウトバウンドトラフィックのルールは通常、DB クラスターには適用されません。送信トラフィックのルールは、DB クラスター がクライアントである場合にのみ適用されます。VPC セキュリティグループを作成するには、「Amazon EC2 API」を使用するか、VPC コンソールの [Security Group] (セキュリティグループ) オプションを使用する必要があります。

VPC 内のクラスターへのアクセスを許可する VPC セキュリティグループのルールを作成するときは、そのルールがアクセスを許可するアドレスの範囲ごとにポートを指定する必要があります。例えば、VPC 内のインスタンスへの Secure Shell (SSH) アクセスをオンにするには、指定したアドレスの範囲の TCP ポート 22 に対するアクセスを許可するルールを作成します。

VPC 内の異なるインスタンスに対する異なるポートへのアクセスを許可する複数の VPC セキュリティグループを設定できます。例えば、VPC のウェブサーバーに TCP ポート 80 へのアクセスを許可する VPC セキュリティグループを作成できます。次に、VPC の Aurora MySQL DB インスタンスの TCP ポート 3306 へのアクセスを許可する別の VPC セキュリティグループを作成できます。

> **Note**
>
> Aurora DB クラスターでは、DB クラスターに関連付けられている VPC セキュリティグループも、DB クラスター内のすべての DB インスタンスに関連付けられています。DB クラスターまたは DB インスタンスの VPC セキュリティグループを変更した場合、その変更は DB クラスター内のすべての DB インスタンスに自動的に適用されます。

VPC セキュリティグループの詳細については、Amazon Virtual Private Cloud ユーザーガイドの「セキュリティグループ」を参照してください。

> **Note**
>
> DB クラスター が VPC 内にあるが、パブリックアクセス可能でない場合は、AWS Site-to-Site VPN 接続または AWS Direct Connect 接続を使用してプライベートネットワークからアクセスすることもできます。詳しくは、「インターネットトラフィックのプライバシー」を参照してください。

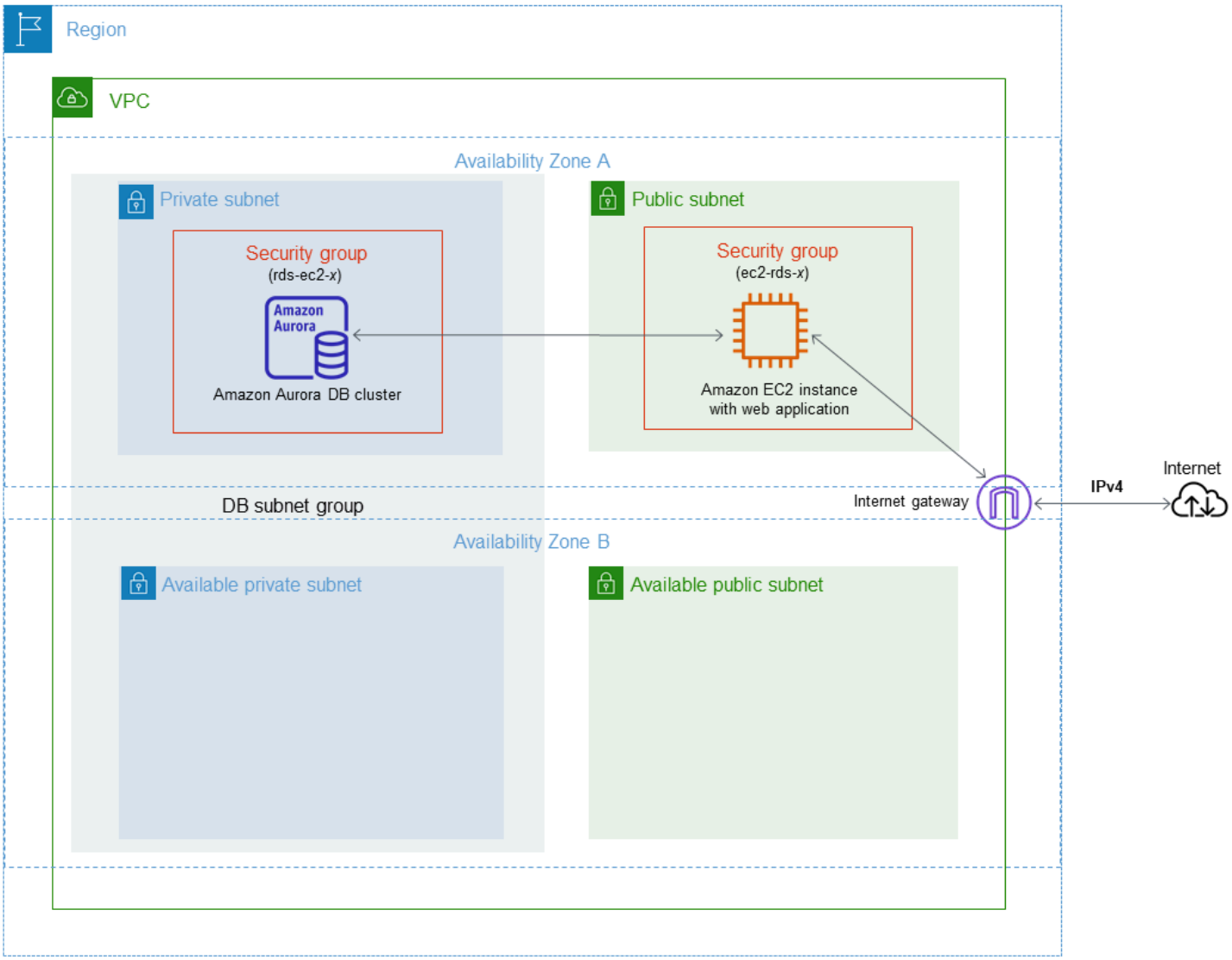
セキュリティグループのシナリオ

VPC 内の DB クラスターの一般的な用途は、同じ VPC 内の Amazon EC2 インスタンスで実行され、VPC の外にあるクライアントアプリケーションによってアクセスされるアプリケーションサーバーとデータを共有することです。このシナリオでは、AWS Management Consoleの RDS および VPC ページ、または RDS および EC2 API オペレーションを使用して、必要なインスタンスおよびセキュリティグループを作成します。

1. VPC セキュリティグループ (「sg-0123ec2example」など) を作成し、ソースとしてクライアントアプリケーションの IP アドレスを使用するという受信ルールを定義します。このセキュリティグループにより、クライアントアプリケーションは、このセキュリティグループを使用する VPC 内の EC2 インスタンスに接続できるようになります。
2. アプリケーションの EC2 インスタンスを作成し、前のステップで作成した VPC セキュリティグループ (「sg-0123ec2example」) に EC2 インスタンスを追加します。
3. 2 つ目の VPC セキュリティグループ (「sg-6789rdsexample」など) を作成し、ステップ 1 で作成した VPC セキュリティグループ (「sg-0123ec2example」) をソースとして指定して新しいルールを作成します。

4. 新しい DB クラスターを作成し、その DB クラスターを前のステップで作成した VPC セキュリティグループ (「sg-6789rdsexample」) に追加します。DB クラスターを作成するとき、ステップ 3 で作成した VPC セキュリティグループ (「sg-6789rdsexample」) のルールに指定した同じポート番号を使用します。

以下の図に、このシナリオを示しています。



このシナリオの VPC 設定の詳細については、「チュートリアル: DB クラスターで使用する VPC を作成する (IPv4 専用)」をご参照ください。VPC の使用方法の詳細については、「Amazon VPC VPC とAmazon Aurora」を参照してください。

VPC セキュリティグループを作成する

DB インスタンスの VPC セキュリティグループは、VPC コンソールを使って作成できます。セキュリティグループを作成する方法については、Amazon Virtual Private Cloud ユーザーガイドの「セキュリティグループを作成して VPC 内の DB クラスターへのアクセスを提供する」および「セキュリティグループ」を参照してください。

セキュリティグループを DB クラスターと関連付ける

RDS コンソールの [クラスターの変更]、`ModifyDBCluster` Amazon RDS API、または `modify-db-cluster` AWS CLI コマンドを使用して、セキュリティグループを DB クラスターに関連付けることができます。

次の CLI の例では、特定の VPC グループを関連付け、DB クラスターから DB セキュリティグループを削除します。

```
aws rds modify-db-cluster --db-cluster-identifier dbName --vpc-security-group-ids sg-ID
```

DB クラスターの変更については、「Amazon Aurora DB クラスターの変更」を参照してください。

マスターユーザーアカウント権限

新しい DB クラスターを作成すると、使用するデフォルトマスターユーザーがその DB クラスターの特定の権限を取得します。DB クラスターの作成後にマスターユーザー名を変更することはできません。

> **Important**
>
> アプリケーションではマスターユーザーを直接使用しないことを強くお勧めします。代わりに、アプリケーションに必要な最小の特権で作成されたデータベースユーザーを使用するというベストプラクティスに従ってください。

> **Note**
>
> マスターユーザーの権限を誤って削除した場合は、DB クラスターを変更して新しいマスターユーザーパスワードを設定することで復元できます。DB クラスターの変更の詳細については、「Amazon Aurora DB クラスターの変更」を参照してください。

次の表は、各データベースエンジンに対してマスターユーザーが取得する権限とデータベースロールを示しています。

データベースエンジン	システム権限	データベースロール
Aurora MySQL	バージョン 2: ALTER, ALTER ROUTINE, CREATE, CREATE ROUTINE, CREATE TEMPORARY TABLES, CREATE USER, CREATE VIEW, DELETE, DROP, EVENT, EXECUTE, GRANT OPTION, INDEX, INSERT, LOAD FROM S3, LOCK TABLES, PROCESS, REFERENCES , RELOAD, REPLICATION CLIENT , REPLICATION SLAVE , SELECT, SELECT INTO S3, SHOW DATABASES , SHOW VIEW, TRIGGER, UPDATE	—
	バージョン 3: ALTER、APPLICATION_PASSWORD_ADMIN 、ALTER ROUTINE、CONNECTION_ADMIN 、CREATE、CREATE ROLE、CREATE ROUTINE、CREATE TEMPORARY TABLES、CREATE USER、CREATE VIEW、DELETE、DROP、DROP ROLE、EVENT、EXECUTE、INDEX、INSERT、LOCK TABLES、PROCESS、REFERENCES 、RELOAD、REPLICATION CLIENT、REPLICATION SLAVE 、ROLE_ADMIN 、SET_USER_ID 、SELECT、SHOW DATABASES 、SHOW_ROUTINE (Aurora MySQL バージョン 3.04 以降)、SHOW VIEW、TRIGGER、UPDATE、XA_RECOVER_ADMIN	rds_superuser_role rds_superuser_role の詳細については、「ロールベースの特権モデル」を参照してください。
Aurora PostgreSQL	LOGIN, NOSUPERUSER , INHERIT, CREATEDB, CREATEROLE , NOREPLICATION , VALID UNTIL 'infinity'	RDS_SUPERUSER

データベースエンジン	システム権限	データベースロール
		RDS_SUPERUSER の詳細については、「PostgreSQL のロールとアクセス権限について」を参照してください。

Amazon Aurora のサービスにリンクされたロールの使用

Amazon Aurora は、AWS Identity and Access Management (IAM) のサービスにリンクされたロールを使用します。サービスにリンクされたロールは、Amazon Aurora に直接リンクされた一意のタイプの IAM ロールです。サービスにリンクされたロールは、Amazon Aurora による事前定義済みのロールであり、ユーザーに代わってサービスから AWS の他のサービスを呼び出すために必要なすべてのアクセス許可を備えています。

サービスにリンクされたロールを使用することで、必要なアクセス許可を手動で追加する必要がなくなるため、Amazon Aurora の使用が簡単になります。サービスにリンクされたロールのアクセス許可は、Amazon Aurora により定義されます。特に指定されている場合を除き、Amazon Aurora のみがそのロールを引き受けることができます。定義される許可は、信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の IAM エンティティにアタッチすることはできません。

ロールを削除するには、まず関連リソースを削除します。これにより、リソースへの意図しないアクセスによるアクセス許可の削除が防止され、Amazon Aurora リソースは保護されます。

サービスにリンクされたロールをサポートするその他のサービスについては、IAM と連携する AWS サービスを参照の上、[Service-Linked Role] (サービスにリンクされたロール) の欄が [Yes] (はい) になっているサービスを検索してください。そのサービスに関するサービスにリンクされたロールのドキュメントを表示するには、リンクが設定されている [はい] を選択します。

Amazon Aurora のサービスにリンクされたロールのアクセス許可

Amazon Auroraでは、AWSServiceRoleForRDS と呼ばれるサービスにリンクされたロールを使用します。これにより Amazon RDS は DB クラスターに代わって AWS サービスを呼び出せるようになります。

サービスにリンクされたロール AWSServiceRoleForRDS では、以下のサービスを信頼してロールを引き受けます。

- `rds.amazonaws.com`

このサービスにリンクされたロールには、アカウントで操作するためのアクセス許可を付与する `AmazonRDSServiceRolePolicy` というアクセス許可ポリシーがアタッチされています。ロールのアクセス許可ポリシーは、指定したリソースに対して以下のアクションを実行することを Amazon Aurora に許可します。

JSON ポリシードキュメントを含むこのポリシーの詳細については、AWS マネージドポリシーリファレンスガイドの「AmazonRDSServiceRolePolicy」を参照してください。

> **Note**
>
> サービスにリンクされたロールの作成、編集、削除を IAM エンティティ (ユーザー、グループ、ロールなど) に許可するには、許可を設定する必要があります。次のエラーメッセージが表示された場合は、以下のように対応します。
> リソースを作成できません。サービスにリンクされたロールを作成するために必要なアクセス許可があることを確認します。それ以外の場合は、時間をおいてからもう一度お試しください。
> 次のアクセス許可が有効であることを確認します。
>
> ```
> {
>     "Action": "iam:CreateServiceLinkedRole",
>     "Effect": "Allow",
>     "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
> AWSServiceRoleForRDS",
>     "Condition": {
>         "StringLike": {
>             "iam:AWSServiceName":"rds.amazonaws.com"
>         }
>     }
> }
> ```
>
> 詳細については、IAM ユーザーガイド の「サービスにリンクされたロールのアクセス許可」を参照してください。

Amazon Aurora のサービスにリンクされたロールの作成

サービスにリンクされたロールを手動で作成する必要はありません。DB クラスターを作成する際、Amazon Aurora がサービスにリンクされたロールを作成します。

> **Important**
>
> サービスにリンクされたロールのサポートが開始された 2017 年 12 月 1 日より前に Amazon Aurora サービスを使用している場合、その時点で Amazon Aurora が、ご使用のアカウント

内に AWSServiceRoleForRDS ロールを作成しています。詳細については、「AWS アカウントに新しいロールが表示される」を参照してください。

このサービスにリンクされたロールを削除した後で再度作成する必要が生じた場合は、同じ方法でアカウントにロールを再作成できます。DB クラスターを作成する際、Amazon Aurora がサービスにリンクされたロールを再度作成します。

Amazon Aurora のサービスにリンクされたロールの編集

Amazon Aurora では、サービスにリンクされたロール AWSServiceRoleForRDS を編集できません。サービスにリンクされたロールを作成すると、多くのエンティティによってロールが参照される可能性があるため、ロール名を変更することはできません。ただし、IAM を使用したロールの説明の編集はできます。詳細については、IAM ユーザーガイドの「サービスにリンクされたロールの編集」を参照してください。

Amazon Aurora のサービスにリンクされたロールの削除

サービスにリンクされたロールが必要な機能またはサービスが不要になった場合には、そのロールを削除することをお勧めします。そうすることで、使用していないエンティティがアクティブにモニタリングされたり、メンテナンスされたりすることがなくなります。ただし、サービスにリンクされたロールを削除する前に、すべての DB クラスターを削除する必要があります。

サービスにリンクされたロールのクリーンアップ

IAM を使用してサービスにリンクされたロールを削除するには、まずそのロールにアクティブなセッションがないことを確認し、そのロールで使用されているリソースをすべて削除する必要があります。

サービスにリンクされたロールにアクティブなセッションがあるかどうかを、IAM コンソールで確認するには

1. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. IAM コンソールのナビゲーションペインで [ロール] を選択します。次に、AWSServiceRoleForRDS ロールの名前 (チェックボックスではありません) を選択します。
3. 選択したロールの [概要] ページで、[アクセスアドバイザー] タブを選択します。
4. [Access Advisor] タブで、サービスにリンクされたロールの最新のアクティビティを確認します。

> **Note**
>
> Amazon Aurora が AWSServiceRoleForRDS ロールを使用しているかどうか不明な場合は、ロールを削除してみてください。サービスでロールが使用されている場合、削除は失敗し、ロールが使用されている AWS リージョンが表示されます。ロールが使用されている場合は、ロールを削除する前にセッションが終了するのを待つ必要があります。サービスにリンクされたロールのセッションを取り消すことはできません。

AWSServiceRoleForRDS ロールを削除する場合、初期にすべての DB クラスターを削除する必要があります。

すべてのクラスターの削除

以下のいずれかの手順を使用して、単一のクラスターを削除します。クラスターごとに手順を繰り返します。

クラスターを削除するには (コンソール)

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. [データベース] リストで、削除するクラスターを選択します。
3. [クラスターアクション] で、[削除] を選択します。
4. [Delete] (削除) をクリックします。

クラスターを削除するには (CLI)

AWS CLI コマンドリファレンス の「`delete-db-cluster`」を参照してください。

クラスターを削除するには (API)

Amazon RDS API Reference の「`DeleteDBCluster`」を参照してください。

IAM コンソール、IAM CLI、または IAM API を使用して、AWSServiceRoleForRDS サービスにリンクされたロールを削除します。詳細については、IAM ユーザーガイドの「サービスにリンクされたロールの削除」を参照してください。

Amazon VPC VPC とAmazon Aurora

Amazon Virtual Private Cloud (Amazon VPC) を使用すると、Aurora DB クラスターなどの AWS リソースを仮想プライベートクラウド (VPC) で起動できます。

VPC を使用する場合、仮想ネットワーキング環境を制御できます。独自の IP アドレスの範囲を選択し、サブネットを作成してルーティングおよびアクセス制御リストを設定できます。VPC で DB クラスターを実行するために、追加料金はかかりません。

アカウントにはデフォルト VPC があります。新しいすべての DB クラスターは、特に指定がない限り、デフォルトの VPC 内に作成されます。

トピック

- VPC 内の DB クラスターの使用
- VPC の DB クラスターにアクセスするシナリオ
- チュートリアル: DB クラスターで使用する VPC を作成する (IPv4 専用)
- チュートリアル: DB クラスター用の VPC を作成する (デュアルスタックモード)

以下に、 Amazon Aurora DB クラスターに関連する VPC の機能について説明します。Amazon VPC の詳細については、Amazon VPC 入門ガイドおよびAmazon VPC ユーザーガイドを参照してください。

VPC 内の DB クラスターの使用

DB クラスターは仮想プライベートクラウド (VPC) 内にあります。VPC は、AWS クラウドの他の仮想ネットワークから論理的に切り離された仮想ネットワークです。Amazon VPC では、 Amazon Aurora DB クラスターや Amazon EC2 インスタンスなど、AWS リソースを VPC で起動できます。VPC は、自分のアカウントに属するデフォルト VPC を使用するか、独自に作成することもできます。すべての VPC は、AWS アカウントに関連付けられます。

デフォルト VPC には、VPC 内でリソースを隔離するために使用できる 3 つのサブネットがあります。デフォルト VPC には、VPC 外から VPC 内のリソースへのアクセスを可能にするインターネットゲートウェイもあります。

VPC 内の Amazon RDS DB インスタンスが関係するシナリオのリストについては、「VPC の DB クラスターにアクセスするシナリオ」を参照してください。

トピック

- VPC 内の DB クラスターの使用
- DB サブネットグループの使用
- 共有サブネット
- Amazon Aurora IP アドレス指定
- VPC 内の DB クラスターをインターネットから隠す
- VPC に DB クラスターを作成する

以下のチュートリアルでは、一般的な Amazon Aurora 状況で使用できる VPC の作成方法を学ぶことができます。

- チュートリアル: DB クラスターで使用する VPC を作成する (IPv4 専用)
- チュートリアル: DB クラスター用の VPC を作成する (デュアルスタックモード)

VPC 内の DB クラスターの使用

次に、VPC の DB クラスターの使用に関するヒントを紹介します。

- VPC には少なくとも 2 つのサブネットが必要です。これらのサブネットは、DB クラスターをデプロイする AWS リージョン 内の 2 つの異なるアベイラビリティーゾーンに存在している必要があります。サブネットは、VPC の IP アドレス範囲の指定可能なセグメントで、セキュリティや運用上のニーズに基づいて DB クラスターをグループ化することができます。
- VPC の DB グループをパブリックにアクセス可能にする場合は、VPC 属性の DNS hostnames と DNS resolution を有効にしてください。
- ご利用の VPC では、DB サブネットグループを作成する必要があります。DB サブネットグループを作成するには、作成したサブネットを指定します。Amazon Aurora は、サブネットとそのサブネット内の IP アドレスを選択し、DB クラスターのプライマリ DB インスタンスに関連付けます。プライマリ DB インスタンスは、そのサブネットを含むアベイラビリティーゾーンを使用します。
- VPC には、DB クラスターへのアクセスを許可する VPC セキュリティグループが必要です。

  詳細については、「VPC の DB クラスターにアクセスするシナリオ」を参照してください。
- 各サブネットの CIDR ブロックは、フェイルオーバーやコンピュートスケーリングの見積もりなどのメンテナンス作業中に Amazon Aurora が使用する予備の IP アドレスに十分対応できる大きさが必要です。例えば、通常は 10.0.0.0/24 や 10.0.1.0/24 などの範囲で十分な大きさです。

- VPC では、インスタンスのテナント属性が default または dedicated のいずれかに設定されます。デフォルト VPC では、インスタンスのテナント属性はすべて default に設定され、DB インスタンスのすべてのクラスがサポートされます。

  インスタンステナンシーを dedicated に設定した専有 VPC に DB クラスターを保持する場合は、その DB クラスターの DB インスタンスクラスは、Amazon EC2 で承認された 専有インスタンスタイプのいずれかである必要があります。例えば、r5.large Rc2 ハードウェア専有インスタンスは、db.r5.large DB インスタンスクラスに対応します。VPC のインスタンステナンシーについては、Amazon Elastic Compute Cloud ユーザーガイドの「ハードウェア専有インスタンス」を参照してください。

  ハードウェア専有インスタンスに対応するインスタンスタイプの詳細については、EC2 の料金ページで「Amazon EC2 のハードウェア専有インスタンス」を参照してください。

  > **Note**
  >
  > インスタンスのテナンシー属性を DB クラスター専有に設定しても、DB クラスターが専有ホストで実行されることは保証されません。

DB サブネットグループの使用

サブネットは、VPC の IP アドレス範囲のセグメントで、セキュリティや運用上のニーズに基づいてリソースをグループ化するために指定します。DB サブネットグループは VPC に作成するサブネット (通常はプライベート) のコレクションで、DB クラスター用に指定します。DB サブネットグループを使用することにより、AWS CLI または RDS API を使用して DB クラスターを作成するときに、特定の VPC を指定することができます。コンソールを使用する場合は、使用する VPC とサブネットグループを選択できます。

各 DB サブネットグループには、特定の AWS リージョン 内の少なくとも 2 つのアベイラビリティーゾーンにサブネットが必要です。VPC に DB クラスターを作成するときに、DB サブネットグループを選択する必要があります。DB サブネットグループから、 Amazon Aurora は DB インスタンスのプライマリ DB インスタンスに関連付けるサブネットとそのサブネット内の IP アドレスを選択します。DB は、そのサブネットを含むアベイラビリティーゾーンを使用します。

DB サブネットグループのサブネットはパブリックまたはプライベートのいずれかです。サブネットは、ネットワークアクセス制御リスト (ネットワーク ACL) とルーティングテーブルに定義した設定に応じて、パブリックまたはプライベートになります。DB クラスターをパブリックにアクセス可能にするには、その DB サブネットグループ内のすべてのサブネットがパブリックである必要がありま

す。パブリックにアクセスできる DB クラスターに関連付けられているサブネットがパブリックからプライベートに変更された場合、DB クラスターの可用性に影響する可能性があります。

デュアルスタックモードをサポートする DB サブネットグループを作成するには、DB サブネットグループに追加する各サブネットに Internet Protocol version 6 (IPv6) CIDR ブロックが関連付けられていることを確認してください。詳細については、Amazon Aurora IP アドレス指定 と Amazon VPC ユーザーガイドの「IPv6 に移行する」を参照してください。

Amazon Aurora は、VPC に DB クラスターを作成すると、DB サブネットグループから選択した IP アドレスを使用して、DB クラスターにネットワークインターフェイスを割り当てます。ただし、DB クラスターに接続するときにはドメインネームシステム (DNS) 名を使用することを強くお勧めします。基になる IP アドレスはフェイルオーバー中に変わるため、ドメインネームシステム (DNS) 名を使用することを強くお勧めします。

> **Note**
>
> VPC で実行する DB クラスターごとに、 Amazon Aurora による復旧アクション用として、DB サブネットグループのサブネットごとに最低 1 つのアドレスを確保してください。

共有サブネット

DB クラスターは共有 VPC に作成できます。

共有 VPC を使用する際に留意すべき点がいくつかあります。

- DB クラスターを共有 VPC サブネットから非共有 VPC サブネットに、またはその逆に移動できます。
- 共有 VPC の参加者は、DB クラスターを作成できるように、VPC にセキュリティグループを作成する必要があります。
- 共有 VPC の所有者と参加者は、SQL クエリを使用してデータベースにアクセスできます。ただし、リソースに対して任意の API 呼び出しを行うことができるのは、リソースの作成者だけです。

Amazon Aurora IP アドレス指定

IP アドレスは、VPC のリソースの相互通信とインターネット上のリソースとの通信を有効にします。Amazon Aurora は、IPv4 と IPv6 の両方のアドレス指定プロトコルをサポートしています。デ

フォルトでは、Amazon Aurora と Amazon VPC は IPv4 アドレス指定プロトコルを使用します。この動作をオフにすることはできません。VPC の作成時には、IPv4 CIDR ブロック (プライベート IPv4 アドレスの範囲) を指定する必要があります。必要に応じて、IPv6 CIDR ブロックを VPC とサブネットに割り当て、そのブロックからサブネットの DB クラスターに IPv6 アドレスを割り当てることができます。

IPv6 プロトコルのサポートにより、サポートされる IP アドレスの数が増えます。IPv6 プロトコルを使用することで、インターネットの今後の成長に十分なアドレスを確保できます。新規および既存の RDS リソースは、VPC 内で IPv4 アドレスと IPv6 アドレスを使用できます。アプリケーションの異なる部分で使用される 2 つのプロトコル間のネットワークトラフィックの設定、保護、および変換を行うと、運用上のオーバーヘッドが発生する可能性があります。Amazon RDS リソースについては IPv6 プロトコルを標準化して、ネットワーク構成を簡素化できます。

トピック

- IPv4 アドレス
- IPv6 アドレス
- デュアルスタックモード

IPv4 アドレス

VPC を作成するときには、その VPC の IPv4 アドレスの範囲を CIDR ブロックの形式で指定する必要があります (`10.0.0.0/16` など)。DB サブネットグループは、DB クラスターが使用できる、この CIDR ブロック内の IP アドレスの範囲を定義します。これらの IP アドレスはプライベートまたはパブリックです。

プライベート IPv4 アドレスは、インターネットから到達できない IP アドレスです。DB クラスターと同じ VPC 内の他のリソース (Amazon EC2 インスタンスなど) との通信には、プライベート IPv4 アドレスを使用できます。各 DB クラスターには、VPC 内の通信用のプライベート IP アドレスがあります。

パブリック IP アドレスは、インターネットから到達可能な IPv4 アドレスです。DB クラスターとインターネット上のリソース (SQL クライアントなど) との通信には、パブリックアドレスを使用できます。DB クラスターにパブリック IP アドレスが割り当てられるかどうかは、ユーザーが制御します。

一般的な Amazon Aurora 状況で使用できるプライベート IPv4 アドレスのみで VPC を作成する方法のチュートリアルについては、「チュートリアル: DB クラスターで使用する VPC を作成する (IPv4 専用)」を参照してください。

IPv6 アドレス

オプションで IPv6 CIDR ブロックを VPC およびサブネットと関連付けて、そのブロックから VPC 内のリソースに IPv6 アドレスを割り当てることができます。各 IPv6 アドレスは、グローバルに一意です。

VPC の IPv6 CIDR ブロックは、Amazon の IPv6 アドレスのプールから自動的に割り当てられます。範囲を自分で選択することはできません。

IPv6 アドレスに接続するときには、以下の条件が満たされていることを確認してください。

- クライアントは、クライアントから IPv6 経由でのデータベーストラフィックが許可されるように構成されています。
- DB インスタンスによって使用される RDS セキュリティグループは、クライアントからデータベースへの IPv6 経由のトラフィックが許可されるように、正しく構成されています。
- クライアントのオペレーティングシステムスタックは IPv6 アドレス上のトラフィックを許可し、オペレーティングシステムドライバーとライブラリは、正しいデフォルトの DB インスタンスエンドポイント (IPv4 または IPv6) を選択するように構成されています。

IPv6 の詳細については、Amazon VPC ユーザーガイドの「IP アドレス指定」を参照してください。

デュアルスタックモード

DB クラスターが IPv4 と IPv6 の両方のアドレス指定プロトコルで通信できるときには、デュアルスタックモードで実行しています。したがって、リソースは DB クラスターと IPv4、IPv6、またはその両方で通信できます。RDS は、プライベートデュアルスタックモード DB インスタンスの IPv6 エンドポイントについてインターネットゲートウェイアクセスを無効にします。IPv6 エンドポイントがプライベートであり、VPC 内からのみアクセスできるようにします。

トピック

- デュアルスタックモードと DB サブネットグループ
- デュアルスタックモードの DB インスタンスの操作
- IPv4 専用 DB クラスターをデュアルスタックモードを使用するように変更する
- デュアルスタックネットワーク DB クラスターの可用性
- デュアルスタックネットワーク DB クラスターの制限

一般的な Amazon Aurora 状況で使用できる IPv4 と IPv6 の両方のアドレスを持つ VPC を作成する方法のチュートリアルについては、「チュートリアル: DB クラスター用の VPC を作成する (デュアルスタックモード)」を参照してください。

デュアルスタックモードと DB サブネットグループ

デュアルスタックモードを使用するには、DB クラスターに関連付ける DB サブネットグループ内の各サブネットに IPv6 CIDR ブロックが関連付けられていることを確認してください。新しい DB サブネットグループを作成するか、既存の DB サブネットグループを変更して、この要件を満たすことができます。DB クラスターがデュアルスタックモードになった後も、クライアントは通常どおり接続できます。クライアントセキュリティファイアウォールと RDS DB インスタンスのセキュリティグループが、IPv6 経由のトラフィックを許可するように正しく設定されていることを確認します。接続するために、クライアントは DB クラスターのプライマリインスタンスのエンドポイントを使用します。クライアントアプリケーションは、データベースへの接続時に優先するプロトコルを指定できます。デュアルスタックモードでは、DB クラスターは、クライアントの優先ネットワークプロトコル (IPv4 または IPv6) を検出し、そのプロトコルを使用して接続します。

サブネットの削除または CIDR の関連付け解除により、DB サブネットグループがデュアルスタックモードをサポートしなくなった場合、DB サブネットグループに関連付けられている DB インスタンスに対して互換性のないネットワーク状態が発生するリスクがあります。また、新しいデュアルスタックモードの DB クラスターの作成時に DB サブネットグループを使用することはできません。

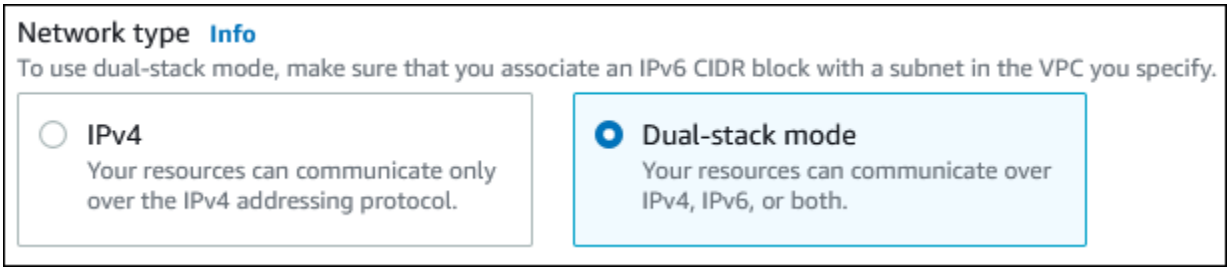
AWS Management Console を使用して DB サブネットグループがデュアルスタックモードをサポートしているかどうかを判断するには、DB サブネットグループの詳細ページで [Network type] (ネットワークタイプ) を確認します。DB サブネットグループが AWS CLI を使用してデュアルスタックモードをサポートしているかどうかを判断するには、describe-db-subnet-groups コマンドを実行して、出力の `SupportedNetworkTypes` を確認します。

リードレプリカは独立した DB インスタンスとして扱われ、プライマリ DB インスタンスとは異なるネットワークタイプを持つことができます。リードレプリカのプライマリ DB インスタンスのネットワークタイプを変更しても、リードレプリカは影響を受けません。DB インスタンスを復元するときには、サポートされている任意のネットワークタイプに復元できます。

デュアルスタックモードの DB インスタンスの操作

DB クラスターを作成または変更する場合は、デュアルスタックモードを指定して、リソースが DB クラスターと IPv4、IPv6、またはその両方で通信することを許可できます。

AWS Management Console を使用して DB インスタンスを変更するときには、[Network type] (ネットワークタイプ) セクションでデュアルスタックモードを指定できます。次の画像は、コンソールの [Network type] (ネットワークの種類) セクションを示しています。



AWS CLI を使用して DB クラスターを作成または変更するときには、`--network-type` オプションを `DUAL` に設定して、デュアルスタックモードを使用します。RDS API を使用して DB クラスターを作成または変更するときには、`NetworkType` パラメータを `DUAL` に設定して、デュアルスタックモードを使用します。DB インスタンスのネットワークタイプを変更すると、ダウンタイムが発生する可能性があります。指定された DB エンジンバージョンまたは DB サブネットグループでデュアルスタックモードがサポートされていない場合は、`NetworkTypeNotSupported` エラーが返されます。

DB クラスター作成の詳細については、「Amazon Aurora DB クラスターの作成」を参照してください。DB クラスターの変更の詳細については、「Amazon Aurora DB クラスターの変更」を参照してください。

コンソールを使用して DB クラスターがデュアルスタックモードであるかどうかを判断するには、DB クラスターの [Connectivity & security] (接続性とセキュリティ) タブの [Network type] (ネットワークの種類) を確認します。

IPv4 専用 DB クラスターをデュアルスタックモードを使用するように変更する

IPv4 専用 DB クラスターをデュアルスタックモードを使用するように変更できます。このためには、DB クラスターのネットワークの種類を変更します。変更によってダウンタイムが発生する可能性があります。

メンテナンスウィンドウ中に Amazon Aurora クラスターのネットワークタイプを変更することをお勧めします。現在、新しいインスタンスのネットワークタイプをデュアルスタックモードに設定することはサポートされていません。ネットワークタイプは、`modify-db-cluster` コマンドを使用して手動で設定できます。

DB クラスターをデュアルスタックモードを使用するように変更する前に、その DB サブネットグループがデュアルスタックモードをサポートしていることを確認してください。DB クラスターに関連付けられた DB サブネットグループがデュアルスタックモードをサポートしていない場合は、DB

クラスターを変更するときに、それをサポートする別の DB サブネットグループを指定します。DB クラスターの DB サブネットグループを変更すると、ダウンタイムが発生する可能性があります。

DB クラスターをデュアルスタックモードを使用するように変更する前に DB クラスターの DB サブネットグループを変更する場合は、変更の前後に DB サブネットグループが DB クラスターに対して有効であることを確認してください。

Amazon Aurora クラスターのネットワークをデュアルスタックモードに変更するには、`--network-type` パラメータに値 `DUAL` のみを指定して modify-db-cluster API を実行することをお勧めします。同じ API コールで `--network-type` パラメータと一緒に他のパラメータを追加すると、ダウンタイムが発生する可能性があります。

変更後に DB クラスターに接続できない場合は、選択したネットワーク (IPv4 または IPv6) でデータベースへのトラフィックを許可するように、クライアントとデータベースのセキュリティファイアウォールとルートテーブルが正確に設定されていることを確認してください。IPv6 アドレスを使用して接続するには、オペレーティングシステムのパラメータ、ライブラリ、またはドライバーを変更しなければならない場合もあります。

IPv4 専用の DB クラスターをデュアルスタックモードを使用するように変更するには

1. DB サブネットグループをデュアルスタックモードをサポートするように変更するか、デュアルスタックモードをサポートする DB サブネットグループを作成します。

   a. IPv6 CIDR ブロックと VPC の関連付け

      詳細については、「Amazon VPC ユーザーガイド」の「IPv6 CIDR ブロックを VPC に追加する」を参照してください。

   b. IPv6 CIDR ブロックを DB サブネットグループ内のすべてのサブネットにアタッチします。

      詳細については、「Amazon VPC ユーザーガイド」の「IPv6 CIDR ブロックをサブネットに追加する」を参照してください。

   c. DB サブネットグループがデュアルスタックモードをサポートしていることを確認します。

      AWS Management Console を使用している場合は、DB サブネットグループを選択し、[Supported network types] (サポートされているネットワークタイプ) の値が [Dual, IP v4] (デュアル、IPv4) であることを確認します。

      AWS CLI を使用している場合は、describe-db-subnet-groups コマンドを実行して、DB インスタンスの `SupportedNetworkType` の値が `Dual, IPv4` であることを確認します。

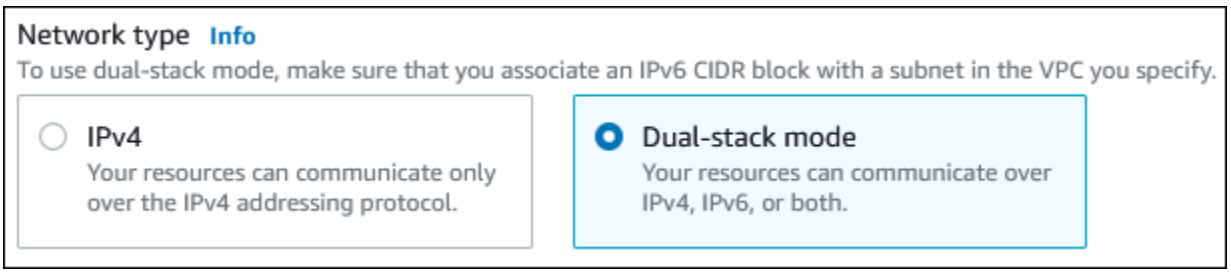
2. DB クラスターに関連付けられたセキュリティグループを、データベースへの IPv6 接続を許可するように変更するか、IPv6 接続を許可する新しいセキュリティグループを作成します。

   手順については、Amazon VPC ユーザーガイドの「セキュリティグループのルール」を参照してください。

3. DB クラスターを変更して、デュアルスタックモードをサポートします。そのためには、ネットワークの種類をデュアルスタックモードに設定します。

   コンソールを使用している場合、以下の設定が正しいことを確認します。

   - [Network type] (ネットワークタイプ) – [Dual-stack mode] (デュアルスタックモード)

   - [DB Subnet group] (DB サブネットグループ) — 前のステップで設定した DB サブネットグループ
   - [Security group] (セキュリティグループ) - 前のステップで設定したセキュリティ

   AWS CLI を使用している場合、以下の設定が正しいことを確認します。

   - `--network-type` – `dual`
   - `--db-subnet-group-name` - 前のステップで設定した DB サブネットグループ
   - `--vpc-security-group-ids` - 前のステップで設定した VPC セキュリティグループ

   例:

   ```
   aws rds modify-db-cluster --db-cluster-identifier my-cluster --network-type "DUAL"
   ```

4. DB クラスターがデュアルスタックモードをサポートしていることを確認します。

   コンソールを使用している場合、DB クラスターの (接続とセキュリティ) [Configuration] (設定) タブを選択します。そのタブで、ネットワークの種類値がデュアルスタックモードであることを確認してください。

AWS CLI を使用している場合は、describe-db-clusters コマンドを実行して、DB クラスターの NetworkType の値が dual であることを確認します。

ライター DB インスタンスエンドポイントで dig コマンドを実行して、関連付けられている IPv6 アドレスを特定します。

```
dig db-instance-endpoint AAAA
```

DB クラスターに接続するには、IPv6 アドレスではなくライター DB インスタンスエンドポイントを使用します。

デュアルスタックネットワーク DB クラスターの可用性

以下の DB エンジンのバージョンは、アジアパシフィック (ハイデラバード)、アジアパシフィック (メルボルン)、カナダ西部 (カルガリー)、欧州 (スペイン)、欧州 (チューリッヒ)、イスラエル (テルアビブ)、および中東 (アラブ首長国連邦) リージョンを除き、デュアルスタックネットワーク DB クラスターをサポートします。

- Aurora MySQL バージョン:
  - バージョン 3 の 3.02 以降
  - バージョン 2 の 2.09.1 以降

  Aurora MySQL バージョンの詳細については、「Aurora MySQL リリースノート」を参照してください。
- Aurora PostgreSQL バージョン
  - バージョン 14 の中の 14.3 以降
  - バージョン 13 の中の 13.7 以降

  Aurora PostgreSQL バージョンの詳細については、「Aurora PostgreSQL リリースノート」を参照してください。

デュアルスタックネットワーク DB クラスターの制限

デュアルスタックネットワーク DB クラスターには、次の制限が適用されます。

- DB クラスターは、IPv6 プロトコルを排他的に使用することはできません。IPv4 を排他的に使用するか、IPv4 と IPv6 プロトコル を使用することができます (デュアルスタックモード)。

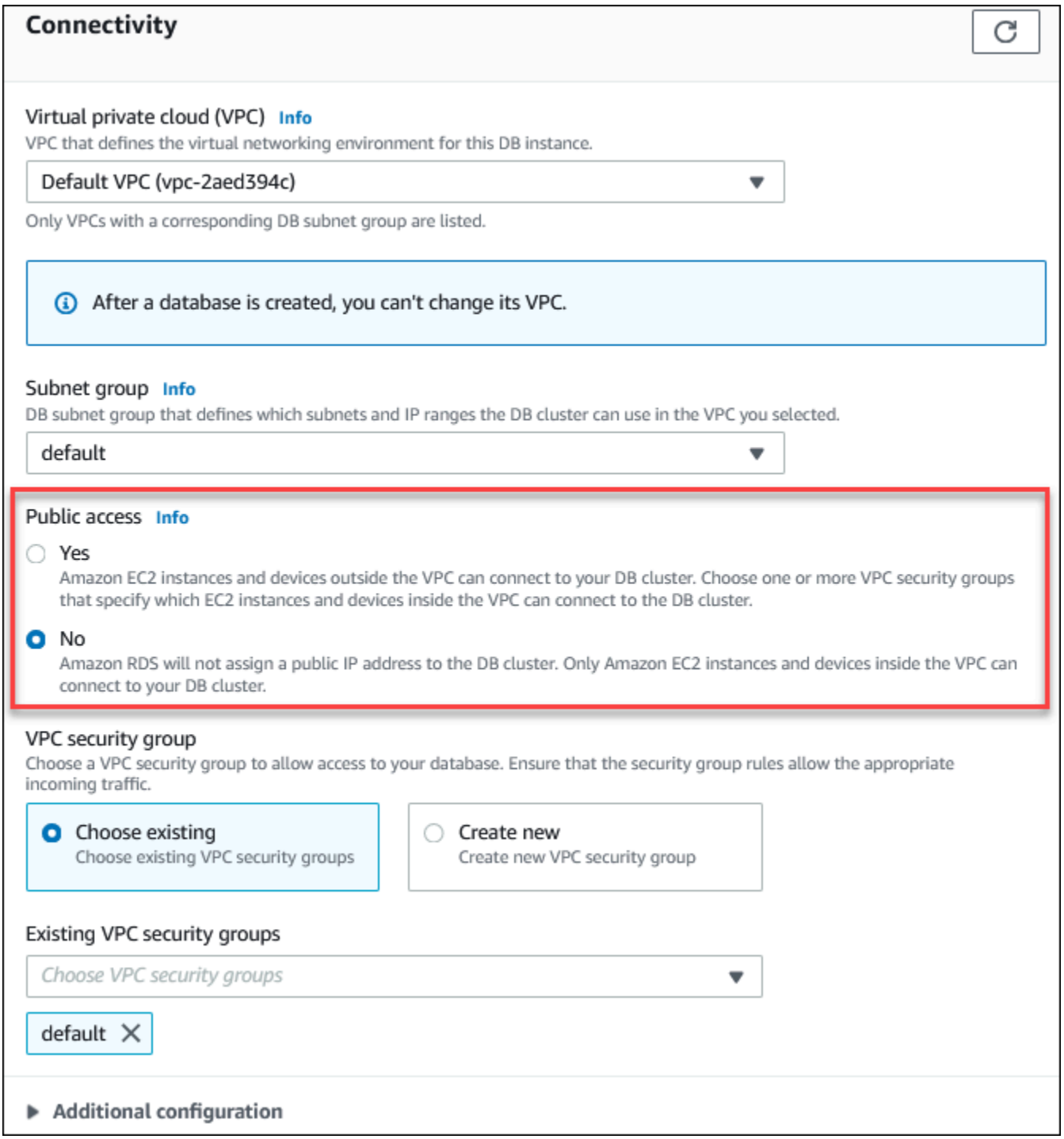
- Amazon RDS は、ネイティブ IPv6 サブネットをサポートしていません。
- デュアルスタックモードを使用する DB クラスターはプライベートでなければなりません。パブリックにアクセス可能にすることはできません。
- デュアルスタックモードは、 および db.r3 DB インスタンスクラスをサポートしません。
- デュアルスタックモード DB クラスターでは RDS Proxy を使用できません。

VPC 内の DB クラスターをインターネットから隠す

Amazon Aurora の一般的なシナリオの 1 つでは、一般向けウェブアプリケーションを使用する EC2 インスタンスと、パブリックアクセスが不可能なデータベースを使用する DB クラスターがある VPC を想定しています。例えば、パブリックサブネットとプライベートサブネットを持つ VPC を作成できます。ウェブサーバーとして機能する Amazon EC2 インスタンスをパブリックサブネットにデプロイできます。DB クラスターは、プライベートサブネットにデプロイされます。このような配置では、ウェブサーバーだけが DB クラスターにアクセスできます。このシナリオの説明については、「VPC 内の DB クラスターに同じ VPC 内の EC2 インスタンスからアクセスする」を参照してください。

VPC 内で DB クラスターを起動すると、DB クラスターには VPC 内のトラフィック用のプライベート IP アドレスが割り当てられます。このプライベート IP アドレスにはパブリックアクセスができません。パブリックアクセスオプションを使用すると、DB クラスターがプライベート IP アドレスだけでなく、パブリック IP アドレスも保持するかどうかを指定できます。DB クラスターがパブリックアクセスに指定されている場合、その DNS エンドポイントは VPC 内からプライベート IP アドレスに解決されます。VPC の外部からパブリック IP アドレスに解決されます。DB クラスターへのアクセスは、最終的に使用されるセキュリティグループによって制御されます。DB クラスターに割り当てられたセキュリティグループに、それを許可するインバウンドルールが含まれていない場合、そのパブリックアクセスは許可されません。また、内部ゲートウェイ DB クラスターをパブリックにアクセス可能にするには、その DB サブネットグループのサブネットにインターネットゲートウェイが必要です。詳細については、「Amazon RDS DB インスタンスに接続できない」を参照してください。

パブリックアクセスオプションを変更することによって、DB クラスターのパブリックアクセシビリティをオンまたはオフにすることができます。次の図は、[追加の接続設定] セクションの [パブリックアクセス] オプションを示しています。このオプションを設定するには、[接続] セクションの [追加の接続設定] セクションを開きます。



DB インスタンスを変更して [パブリックアクセス] オプションを設定する方法については、「DB クラスター内の DB インスタンスの変更」を参照してください。

VPC に DB クラスターを作成する

次の手順で VPC 内に DB クラスターを作成できます。デフォルトの VPC を使用する場合は、ステップ 2 から始めて、既に作成されている VPC と DB サブネットグループを使用することができます。VPC を追加で作成する場合は、VPC を新規に作成できます。

> **Note**
>
> VPC の DB クラスターへのパブリックアクセスを可能にするには、VPC 属性の DNS hostnames と DNS resolution を有効化して、VPC に関する DNS 情報を更新する必要があります。VPC インスタンスの DNS 情報の更新については、「VPC の DNS サポートを更新する」を参照してください。

VPC 内に DB インスタンスを作成するには、以下のステップを実行します。

- ステップ 1: VPC を作成する
- ステップ 2: DB サブネットグループを作成する
- ステップ 3: VPC セキュリティグループを作成する
- ステップ 4: VPC に DB インスタンスを作成する

ステップ 1: VPC を作成する

最低 2 つのアベイラビリティーゾーンの中にサブネットを持つ VPC を作成します。これらのサブネットは、DB サブネットグループを作成するときに使用します。デフォルト VPC がある場合、AWS リージョン 内の各アベイラビリティーゾーンに、自動的にサブネットが作成されます。

詳細については、「プライベートサブネットおよびパブリックサブネットを持つ VPC を作成する」または Amazon VPC ユーザーガイドの「VPC を作成する」を参照してください。

ステップ 2: DB サブネットグループを作成する

DB サブネットグループは VPC 用に作成するサブネット (通常はプライベート) のコレクションで、DB クラスターに指定します。DB サブネットグループを使用すると、AWS CLI または RDS API を使用して DB クラスターを作成するときに、特定の VPC を指定できます。コンソールを使用する場合は、使用する VPC とサブネットを選択できます。各 DB サブネットグループには、AWS リージョン 内の少なくとも 2 つのアベイラビリティーゾーンに少なくとも 1 つのサブネットが必要で

す。ベストプラクティスとして、各 DB サブネットグループには、AWS リージョン 内のアベイラビリティーゾーンごとに少なくとも 1 つのサブネットが必要です。

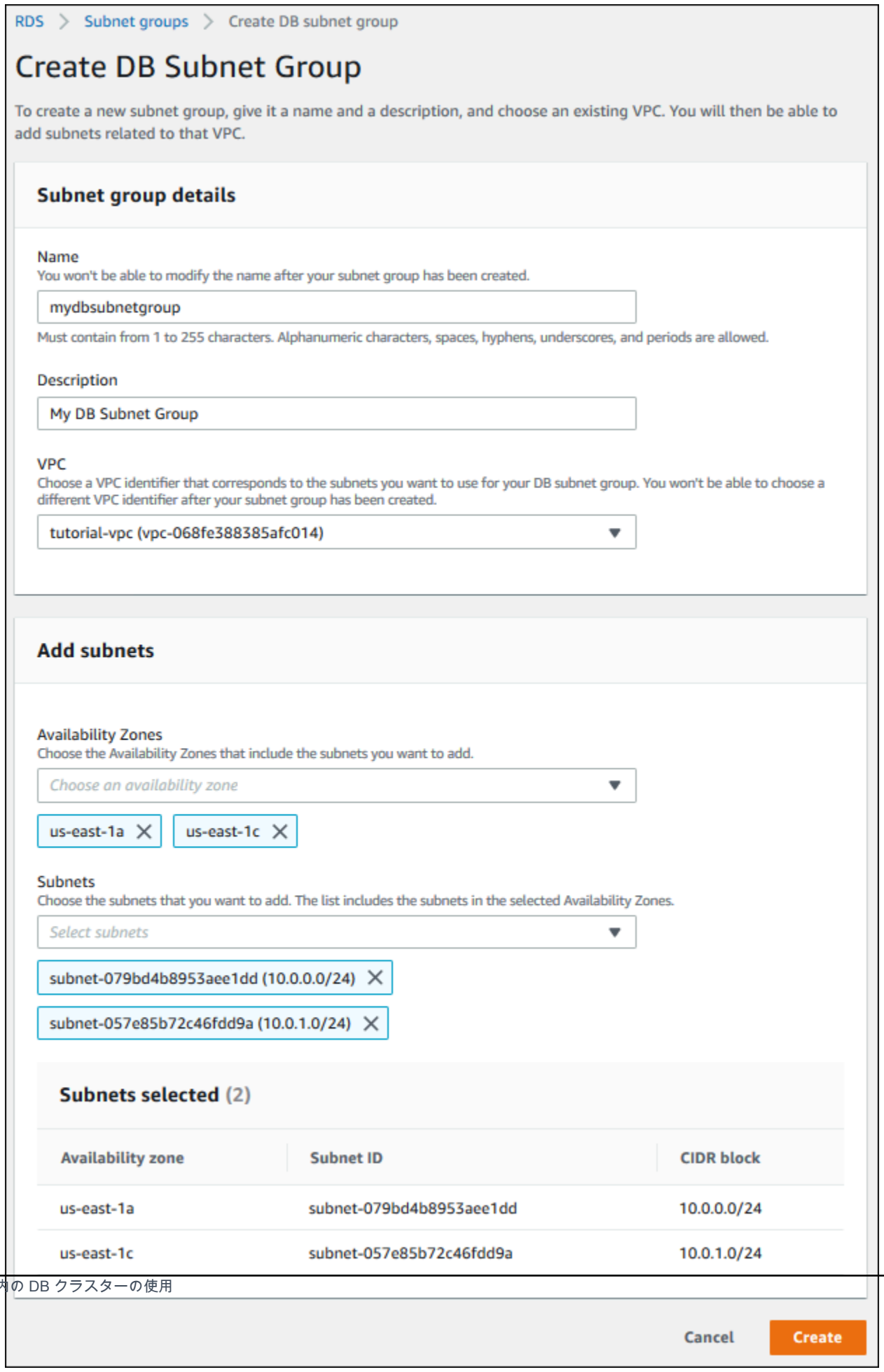
DB クラスターをパブリックにアクセス可能にするには、DB サブネットグループのサブネットにインターネットゲートウェイが必要です。サブネット用のインターネットゲートウェイの詳細については、Amazon VPC ユーザーガイドの「インターネットゲートウェイを使用してサブネットをインターネットに接続する」を参照してください。

VPC に DB クラスターを作成するときに、DB サブネットグループを選択できます。 Amazon Aurora は、サブネットとそのサブネット内の IP アドレスを選択し、DB クラスターに関連付けます。DB サブネットグループが存在しない場合、DB クラスターを作成すると、 Amazon Aurora DB によってデフォルトのサブネットグループが作成されます。 Amazon Aurora では、Elastic Network Interface が作成され、その IP アドレスで DB クラスターに関連付けられます。DB クラスターは、そのサブネットを含むアベイラビリティーゾーンを使用します。

このステップでは、DB サブネットグループを作成し、このグループに VPC 用に作成したサブネットを追加します。

DB サブネットグループを作成する方法

1. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. [Navigation] ペインで、[Subnet groups] を選択します。
3. [Create DB Subnet Group] を選択します。
4. [Name] には、DB サブネットグループの名前を入力します。
5. [Description] に、DB サブネットグループの説明を入力します。
6. [VPC] では、デフォルトの VPC または作成した VPC を選択します。
7. [サブネットの追加] セクションで、サブネットを含むアベイラビリティーゾーンを [アベイラビリティーゾーン] から選択し、サブネットを [サブネット] から選択します。


RDS > Subnet groups > Create DB subnet group
Create DB Subnet Group
To create a new subnet group, give it a name and a description, and choose an existing VPC. You will then be able to add subnets related to that VPC.
Subnet group details
Name
You won't be able to modify the name after your subnet group has been created.
mydbsubnetgroup
Must contain from 1 to 255 characters. Alphanumeric characters, spaces, hyphens, underscores, and periods are allowed.
Description
My DB Subnet Group
VPC
Choose a VPC identifier that corresponds to the subnets you want to use for your DB subnet group. You won't be able to choose a different VPC identifier after your subnet group has been created.
tutorial-vpc (vpc-068fe388385afc014)
Add subnets
Availability Zones
Choose the Availability Zones that include the subnets you want to add.
Choose an availability zone
us-east-1a
us-east-1c
Subnets
Choose the subnets that you want to add. The list includes the subnets in the selected Availability Zones.
Select subnets
subnet-079bd4b8953aee1dd (10.0.0.0/24)
subnet-057e85b72c46fdd9a (10.0.1.0/24)
Subnets selected (2)
Availability zone
Subnet ID
CIDR block
us-east-1a
subnet-079bd4b8953aee1dd
10.0.0.0/24
us-east-1c
subnet-057e85b72c46fdd9a
10.0.1.0/24
Cancel
Create

8. [作成] を選択します。

   RDS コンソールの DB サブネットグループリストに新しい DB サブネットグループが表示されます。DB サブネットグループを選択すると、ウィンドウ下部の詳細ペインに、そのグループに関連付けられたすべてのサブネットなどの詳細を表示することができます。

ステップ 3: VPC セキュリティグループを作成する

DB クラスターを作成する前に、DB クラスターに関連付ける VPC セキュリティグループを作成する必要があります。VPC セキュリティグループを作成しない場合、DB クラスターを作成するときにデフォルトのセキュリティグループを使用します。DB クラスターのセキュリティグループを作成する方法については、「プライベート DB クラスターの VPC セキュリティグループを作成する」を参照するか、Amazon VPC ユーザーガイドの「セキュリティグループを使用してリソースへのトラフィックを制御する」を参照してください。

ステップ 4: VPC に DB インスタンスを作成する

このステップでは、DB クラスターを作成し、前のステップで作成した VPC 名、DB サブネットグループ、および VPC セキュリティグループを使用します。

> **Note**
>
> VPC の DB クラスターをパブリックにアクセス可能にする場合は、VPC 属性の DNS hostnames と DNS resolution を有効にする必要があります。詳細については、Amazon VPC ユーザーガイドの「DNS attributes for your VPC」(VPC の DNS 属性) を参照してください。

DB クラスターの作成方法の詳細については、「Amazon Aurora DB クラスターの作成」を参照してください。

[Connectivity] (接続) セクションにプロンプトが表示されたら、VPC の名前、DB サブネットグループ、および VPC セキュリティグループを入力します。

> **Note**
>
> VPC の更新は Aurora DB クラスターで現在サポートされていません。

VPC の DB クラスターにアクセスするシナリオ

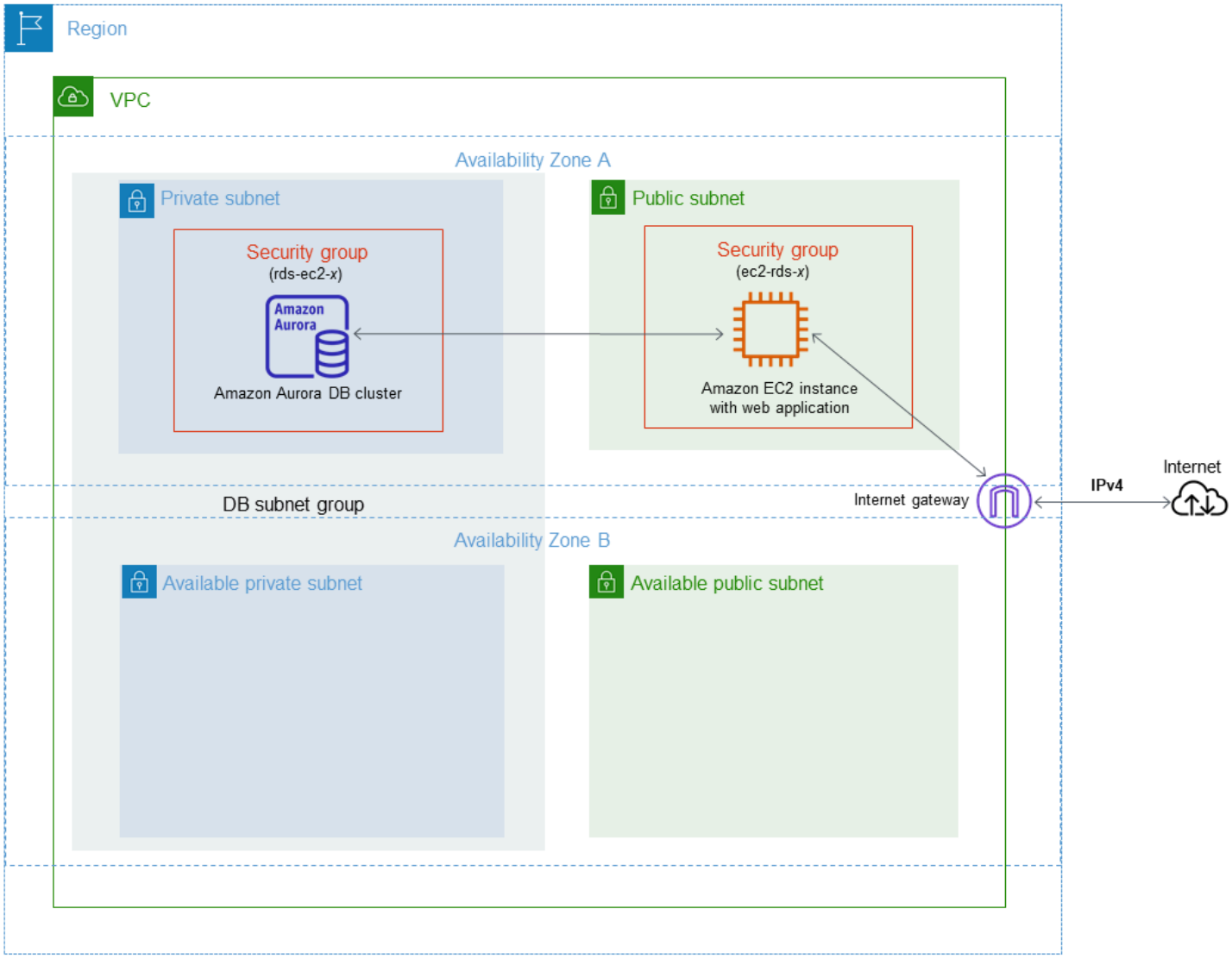
Amazon Aurora は、VPC の DB クラスターにアクセスするための以下のシナリオをサポートします。

- 同じ VPC 内の EC2 インスタンス
- 別の VPC 内の EC2 インスタンス
- インターネット経由のクライアントアプリケーション
- プライベートネットワーク

VPC 内の DB クラスターに同じ VPC 内の EC2 インスタンスからアクセスする

VPC 内の DB クラスターの一般的な用途は、同じ VPC 内の EC2 インスタンスで実行されるアプリケーションサーバーとデータを共有することです。

以下の図に、このシナリオを示しています。



同じ VPC 内の EC2 インスタンスと DB クラスター間のアクセスを管理する方法として最も簡単なのは、次の方法です。

- DB クラスターが存在する VPC セキュリティグループを作成します。このセキュリティグループは、DB クラスターへのアクセスを制限するのに使用できます。たとえば、このセキュリティグループのカスタムルールを作成できます。これにより、DB クラスターを作成したときに割り当てたポートと、開発またはそのほかの目的で DB クラスターにアクセスするのに使用する IP アドレスを使用して TCP へのアクセスを許可できます。
- EC2 インスタンス (ウェブサーバーとクライアント) が属する VPC セキュリティグループを作成します。このセキュリティグループは、必要に応じて、VPC のルーティングテーブルを介したインターネットから EC2 インスタンスへのアクセスを許可できます。例えば、ポート 22 経由で

EC2 インスタンスへの TCP アクセスを許可するルールをこのセキュリティグループに設定できます。

- EC2 インスタンス用に作成したセキュリティグループからの接続を許可する DB クラスターのセキュリティグループで、カスタムルールを作成します。このルールは、セキュリティグループのメンバーに DB クラスターへのアクセスを許可します。

別のアベイラビリティーゾーンに、追加のパブリックサブネットとプライベートサブネットがあります。RDS DB サブネットグループには、2 つ以上のアベイラビリティーゾーンにサブネットが必要です。サブネットが追加されたことで、将来的にマルチ AZ DB インスタンス配置に簡単に切り替えることができるようになります。

このシナリオのパブリックとプライベートの両方のサブネットを使用する VPC を作成する方法のチュートリアルについては、「チュートリアル: DB クラスターで使用する VPC を作成する (IPv4 専用)」を参照してください。

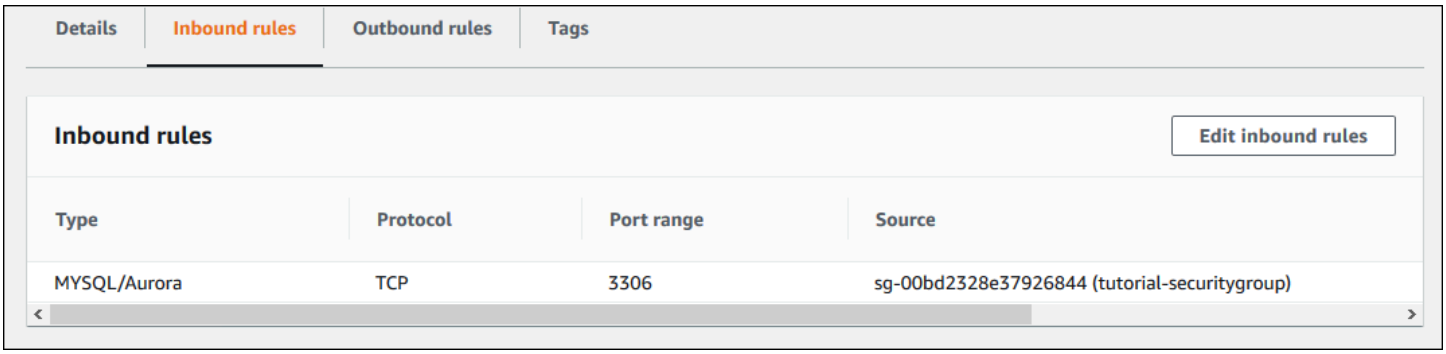
> **Tip**
>
> DB クラスターを作成すると自動的に、Amazon EC2 インスタンスと DB クラスター間のネットワーク接続を設定できるようになります。詳細については、「EC2 インスタンスとの自動ネットワーク接続を設定する」を参照してください。

他のセキュリティグループからの接続を許可する VPC セキュリティグループにルールを作成するには、以下を実行します。

1. AWS Management Console にサインインして、Amazon VPC コンソール (https://console.aws.amazon.com/vpc) を開きます。
2. ナビゲーションペインで、[Security Groups] (セキュリティグループ) を選択します。
3. 他のセキュリティグループのメンバーからのアクセスを許可するセキュリティグループを、選択または作成します。前述のシナリオで、これは DB クラスター向けに使用するセキュリティグループです。[インバウンドルール] タブを選択してから、[インバウンドルールの編集] を選択します。
4. [インバウンドルールの編集] ページで、[ルールの追加] を選択します。
5. [Type] (タイプ) から、DB クラスターの作成時に使用したポートに対応するエントリ ([MYSQL/Aurora] など) を選択します。

6. [ソース] ボックスで、セキュリティグループの ID の入力をスタートすると、一致するセキュリティグループが一覧表示されます。このセキュリティグループによって保護されているリソースへのアクセスを許可するメンバーが所属しているセキュリティグループを選択します。前述のシナリオで、これは EC2 インスタンス向けに使用するセキュリティグループです。
7. 必要に応じて、[タイプ] に [すべての TCP] を、[ソース] ボックスにお客様のセキュリティグループを指定してルールを作成することで、TCP プロトコルのステップを繰り返します。UDP プロトコルを使用する場合は、[All UDP] (すべての UDP) を [Type] (タイプ) と [Source] (送信元) のセキュリティグループとして使用してルールを作成します。
8. [Save Rules] (ルールの保存) を選択します。

次の画面には、ソース用のセキュリティグループを含むインバウンドルールが表示されます。

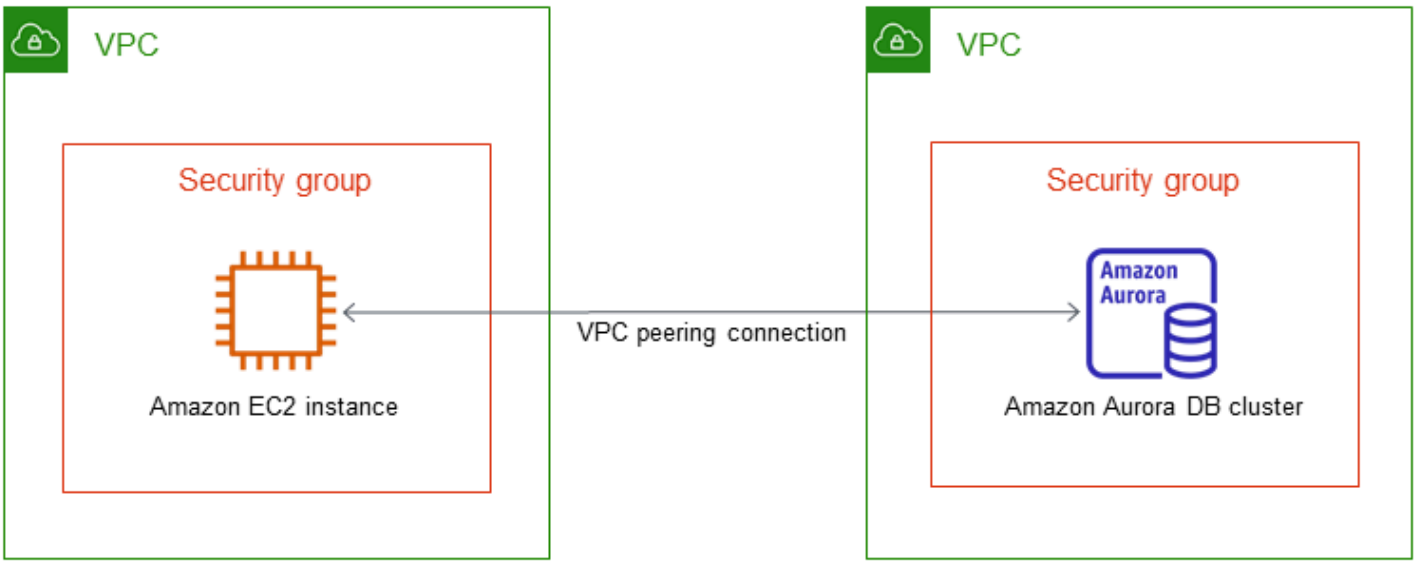


EC2 インスタンスから DB クラスターに接続する方法の詳細については、「Amazon Aurora DB クラスターへの接続」を参照してください。

VPC 内の DB クラスターに別の VPC 内の EC2 インスタンスからアクセスする

DB クラスターがアクセスに使用している EC2 インスタンスとは異なる VPC にある場合、VPC ピア接続を使用してその DB クラスターにアクセスできます。

以下の図に、このシナリオを示しています。

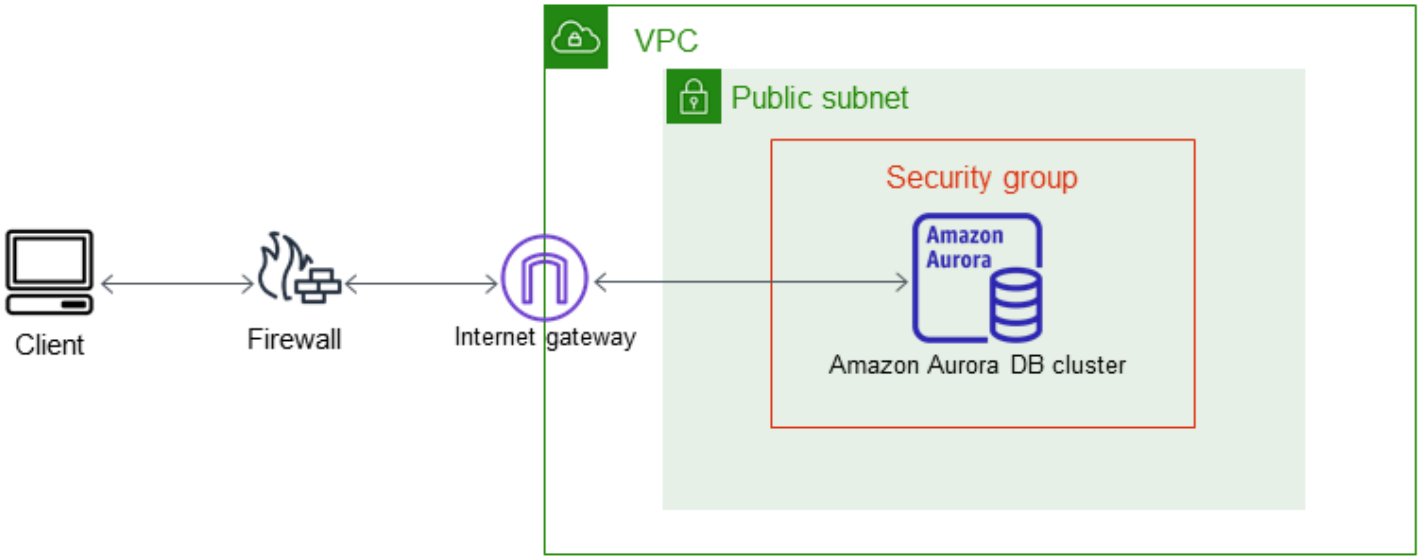


VPC ピア接続は、プライベート IP アドレスを使用して 2 つの VPC 間でトラフィックをルーティングすることを可能にするネットワーク接続です。どちらの VPC のリソースも、同じネットワーク内に存在しているかのように、相互に通信できます。VPC ピアリング接続は、自分の VPC 間、別の AWS アカウントの VPC との間、または別の AWS リージョン の VPC との間に作成できます。VPC ピア接続の詳細については、Amazon Virtual Private Cloud ユーザーガイドの「VPC ピア接続」を参照してください。

インターネット経由でクライアントアプリケーションから VPC 内の DB クラスターにアクセスする

インターネット経由でクライアントアプリケーションから VPC 内の DB クラスターにアクセスするには、1 つのパブリックサブネットを持つ VPC と、インターネットを介した通信を可能にするインターネットゲートウェイを設定します。

以下の図に、このシナリオを示しています。



次の構成をお勧めします。

- サイズ /16 (例えば CIDR: 10.0.0.0/16) の VPC。このサイズでは 65,536 個のプライベート IP アドレスが提供されます。
- サイズ /24 (例えば CIDR: 10.0.0.0/24) のサブネット。このサイズでは 256 個のプライベート IP アドレスが提供されます。
- VPC およびサブネットに関連付けられている Amazon Aurora DB クラスター。Amazon RDS は、サブネット内の IP アドレスを DB クラスターに割り当てます。
- VPC をインターネットと他の AWS 製品に接続するインターネットゲートウェイ。
- DB クラスターに関連付けられたセキュリティグループ。セキュリティグループのインバウンドルールにより、クライアントアプリケーションは DB クラスターにアクセスできます。

VPC での DB クラスターの作成方法に関する詳細は、「VPC に DB クラスターを作成する」を参照してください。

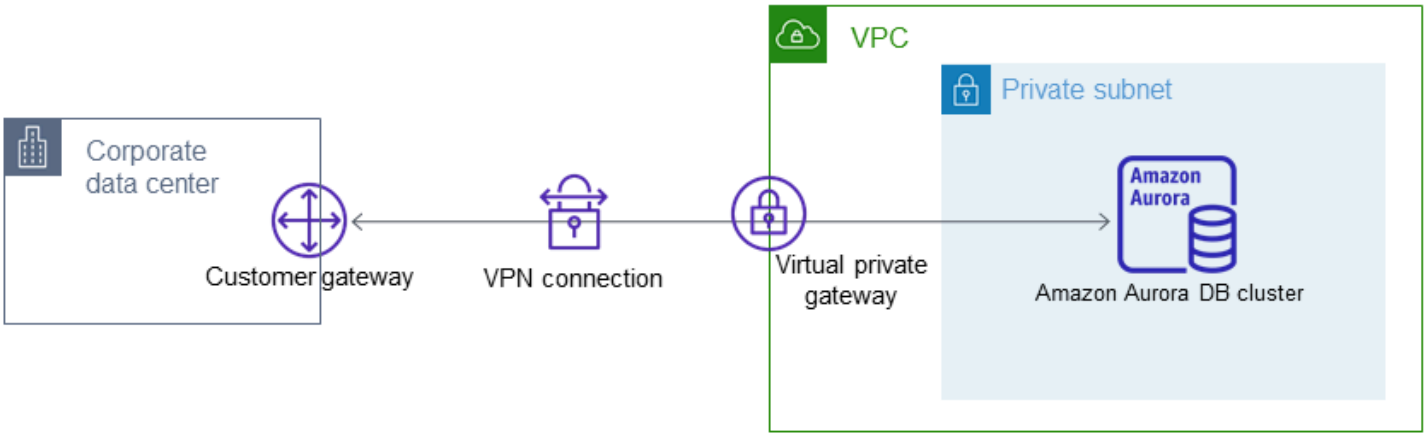
プライベートネットワークによってアクセスされる VPC 内の DB クラスター

DB クラスターがパブリックにアクセスできない場合は、プライベートネットワークからアクセスするための次のオプションがあります。

- AWS Site-to-Site VPN 接続。詳細については、「AWS Site-to-Site VPN とは」を参照してください。

- AWS Direct Connect 接続。詳細については、「AWS Direct Connect とは?」を参照してください。
- AWS Client VPN 接続。詳細については、「AWS Client VPN とは?」を参照してください。

次の図は、AWS Site-to-Site VPN 接続のシナリオを示しています。

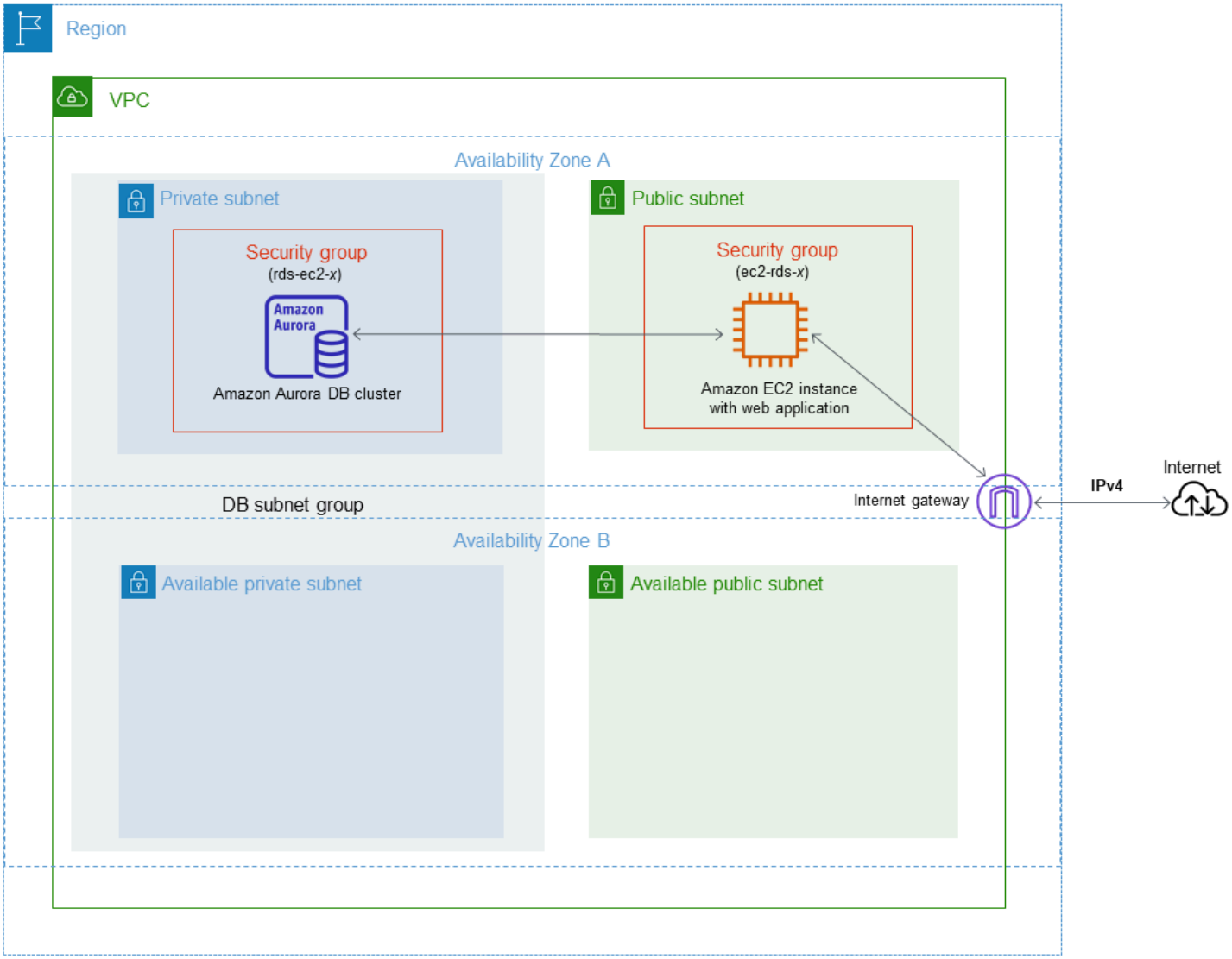


詳細については、「インターネットトラフィックのプライバシー」を参照してください。

チュートリアル: DB クラスターで使用する VPC を作成する (IPv4 専用)

一般的なシナリオには、Amazon VPC サービスに基づく仮想プライベートクラウド (VPC) 内の DB クラスターが含まれます。この VPC は、同じ VPC で実行しているウェブサーバーとデータを共有します。このチュートリアルでは、このシナリオの VPC を作成します。

以下の図に、このシナリオを示しています。その他のシナリオについては、VPC の DB クラスターにアクセスするシナリオ を参照してください。



DB クラスターは、ウェブサーバーからのみ使用可能で、パブリックインターネットからは使用できないようにする必要があります。したがって、パブリックサブネットとプライベートサブネットを持つ VPC を作成します。ウェブサーバーはパブリックサブネットでホストされることで、パブリックインターネットにアクセスできます。DB クラスターはプライベートサブネットでホストされます。ウェブサーバーは、同じ VPC 内でホストされているため、DB クラスターに接続できます。た

だし、DB クラスターはパブリックインターネットからは使用できないため、セキュリティが向上します。

このチュートリアルでは、別のアベイラビリティーゾーンに追加のパブリックサブネットとプライベートサブネットを設定します。これらのサブネットはチュートリアルでは使用されません。RDS DB サブネットグループは、少なくとも 2 つのアベイラビリティーゾーン内のサブネットを必要とします。サブネットを追加すると、複数の Aurora DB インスタンスを簡単に設定できます。

このチュートリアルでは、Amazon Aurora DB クラスター用に VPC を設定する方法について説明します。この VPC シナリオ用のウェブサーバーを作成する方法を示すチュートリアルについては、「チュートリアル: ウェブサーバーと Amazon Aurora DB クラスターを作成する」を参照してください。Amazon VPC の詳細については、Amazon VPC 入門ガイドおよびAmazon VPC ユーザーガイドを参照してください。

> **Tip**
>
> DB クラスターを作成すると自動的に、Amazon EC2 インスタンスと DB クラスター間のネットワーク接続を設定できるようになります。ネットワーク構成は、このチュートリアルで説明したものと似ています。詳細については、「EC2 インスタンスとの自動ネットワーク接続を設定する」を参照してください。

プライベートサブネットおよびパブリックサブネットを持つ VPC を作成する

以下の手順で、パブリックサブネットとプライベートサブネットを持つ VPC を作成します。

VPC とサブネットを作成するには

1. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
2. AWS Management Console の右上隅で、VPC を作成するリージョンを選択します。この例では、米国西部 (オレゴン) リージョンを使用します。
3. 左上隅の [VPC dashboard] (VPC ダッシュボード) を選択します。VPC の作成を開始するには、[Create VPC] (VPC の作成) を選択します。
4. [VPC Settings] (VPC 設定) の [Resources to create] (作成するリソース) で、[VPC and more] (VPC など) を選択します。
5. [VPC settings] (VPC 設定) で、これらの値を設定します。

    - [Name tag auto-generation] (ネームタグ自動生成) – **`tutorial`**

- [IPv4 CIDR block] (IPv4 CIDR ブロック) – **`10.0.0.0/16`**
- [IPv6 CIDR block] (IPv6 CIDR ブロック) – [No IPv6 CIDR block] (IPv6 CIDR ブロックなし)
- [Tenancy] (テナンシー) – デフォルト
- [Number of Availability Zones (AZs)] (アベイラビリティーゾーンの数 (AZ)) – 2
- [Customize AZs] (AZ をカスタマイズする) – デフォルト値を維持します。
- [Number of public subnet] (パブリックサブネット数) – 2
- [Number of private subnets] (プライベートサブネット数) – 2
- [Customize subnets CIDR blocks] (サブネット CIDR ブロックをカスタマイズ) — デフォルト値を維持します。
- [NAT gateways ($)] (NAT ゲートウェイ ($)) – なし
- [VPC endpoints] (VPC エンドポイント) – なし
- [DNS options] (DNS オプション) — デフォルト値を維持します。

6. [VPC の作成] を選択します。

パブリックウェブサーバーの VPC セキュリティグループを作成する

次に、パブリックアクセスのためのセキュリティグループを作成します。VPC 内のパブリック EC2 インスタンスに接続するには、インバウンドルールを VPC セキュリティグループに追加します。これにより、インターネットからのトラフィックを接続できるようになります。

VPC セキュリティグループを作成するには

1. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
2. [VPC ダッシュボード]、[セキュリティグループ]、[セキュリティグループの作成] の順に選択します。
3. [セキュリティグループの作成] ページで、以下の値を設定します。
   - セキュリティグループ名: **`tutorial-securitygroup`**
   - 説明: **`Tutorial Security Group`**
   - [VPC ID]: 前に作成した VPC を選択します (例: [vpc-***`identifier`*** (tutorial-vpc)])。
4. インバウンドルールをセキュリティグループに追加します。
   a. Secure Shell (SSH) を使用して VPC の EC2 インスタンスへの接続に使用する IP アドレスを決定します。パブリック IP アドレスを決定するには、別のブラウザウィンドウまたは

タブで、https://checkip.amazonaws.com のサービスを使用できます。IP アドレスの例は `203.0.113.25/32` です。

多くの場合、インターネットサービスプロバイダー (ISP) 経由、またはファイアウォールの内側から静的 IP アドレスなしで接続することがあります。この場合は、クライアントコンピュータが使用する IP アドレスの範囲を検索します。

> **Warning**
>
> SSH アクセスに `0.0.0.0/0` を使用すると、すべての IP アドレスが SSH を使ってパブリックインスタンスにアクセスできるようになります。この方法は、テスト環境で短時間なら許容できますが、実稼働環境では安全ではありません。実稼働環境では、特定の IP アドレスまたは特定のアドレス範囲にのみ、SSH を使ったインスタンスへのアクセスを限定します。

b. [インバウンドルール] セクションで、[ルールの追加] を選択します。

c. 新しいインバウンドルールに次の値を設定して、Amazon EC2 インスタンスへの SSH アクセスを許可します。こうすることで、Amazon EC2 インスタンスに接続して、ウェブサーバーなどのユーティリティをインストールできます。また、EC2 インスタンスに接続して、ウェブサーバー用のコンテンツをアップロードします。

- タイプ: **SSH**
- ソース: ステップ a で指定した IP アドレスまたはアドレス範囲 (**203.0.113.25/32** など)

d. [ルールの追加] を選択します。

e. 新しいインバウンドルールに次の値を設定して、ウェブサーバーに HTTP へのアクセスを許可します。

- [Type] (タイプ): **HTTP**
- ソース: **0.0.0.0/0**

5. セキュリティグループを作成するには、[Create security group] (セキュリティグループの作成) を選択します。

セキュリティグループ ID を書き留めます。このチュートリアルで後に必要になります。

プライベート DB クラスターの VPC セキュリティグループを作成する

DB クラスターをプライベートのままにするには、プライベートアクセス用の第 2 のセキュリティグループを作成します。VPC 内の専用 DB クラスターに接続するには、ウェブサーバーからのみトラフィックを許可するインバウンドルールを VPC セキュリティグループに追加します。

VPC セキュリティグループを作成するには

1. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
2. [VPC ダッシュボード]、[セキュリティグループ]、[セキュリティグループの作成] の順に選択します。
3. [セキュリティグループの作成] ページで、以下の値を設定します。
   - セキュリティグループ名: **tutorial-db-securitygroup**
   - 説明: **Tutorial DB Instance Security Group**
   - [VPC ID]: 前に作成した VPC を選択します (例: [vpc-***identifier*** (tutorial-vpc)])。
4. インバウンドルールをセキュリティグループに追加します。
   a. [インバウンドルール] セクションで、[ルールの追加] を選択します。
   b. 新しいインバウンドルールに次の値を設定して、Amazon EC2 インスタンスからポート 3306 への MySQL トラフィックを許可します。これを実行すると、ウェブサーバーから DB クラスターに接続できます。そうすることで、ウェブアプリケーションからのデータをデータベースに保存および取得できるようになります。
      - [Type] (タイプ): **MySQL/Aurora**
      - [Source] (ソース): このチュートリアルで以前に作成した tutorial-securitygroup セキュリティグループの ID (例: sg-9edd5cfb)。
5. セキュリティグループを作成するには、[Create security group] (セキュリティグループの作成) を選択します。

DB サブネットグループを作成する

DB サブネットグループは VPC に作成するサブネットのコレクションで、DB クラスター用に指定します。DB サブネットグループでは、DB クラスターの作成時に特定の VPC を指定することができます。

DB サブネットグループを作成するには

1. VPC 内のデータベースのプライベートサブネットを特定します。

   a. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。

   b. [VPC Dashboard] (VPC ダッシュボード) を選択してから、[Subnets] (サブネット) を選択します。

   c. tutorial-subnet-private1-us-west-2a と tutorial-subnet-private2-us-west-2b という名前のサブネット ID に注意してください。

      DB サブネットグループを作成するときに、サブネット ID が必要です。

2. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。

   Amazon VPC コンソールではなく、Amazon RDS コンソールに接続してください。

3. [Navigation] ペインで、[Subnet groups] を選択します。

4. [Create DB subnet group] (DB サブネットグループの作成) を選択します。

5. [DB サブネットグループを作成する] ページで、[サブネットグループの詳細] に値を設定します。

   - 名前: **`tutorial-db-subnet-group`**
   - 説明: **`Tutorial DB Subnet Group`**
   - VPC: tutorial-vpc (vpc-***`identifier`***)

6. [サブネットの追加] セクションで、[アベイラビリティーゾーン] と [サブネット] を選択します。

   このチュートリアルでは、[Availability Zones] (アベイラビリティーゾーン) として [us-west-2a] と [us-west-2b] を選択します。[Subnets] (サブネット) では、前のステップで特定したプライベートサブネットを選択します。

7. [Create] (作成) を選択します。

   RDS コンソールの DB サブネットグループリストに新しい DB サブネットグループが表示されます。DB サブネットグループを選択すると、ウィンドウ下部の詳細ペインに、詳細を表示することができます。これらの詳細には、グループに関連付けられているすべてのサブネットが含まれます。

> **Note**
>
> この VPC を作成して チュートリアル: ウェブサーバーと Amazon Aurora DB クラスターを作成する を完了した場合は、Amazon Aurora DB クラスターの作成 の手順に従って DB クラスターを作成します 。

VPC の削除

このチュートリアルの VPC およびその他のリソースを作成後、不要になった場合は、削除できます。

> **Note**
>
> このチュートリアルで作成した VPC にリソースを追加した場合は、VPC を削除する前にこれらを削除しなければならない場合があります。例えば、これらのリソースには Amazon EC2 インスタンスや Amazon RDS DB クラスター が含まれる場合があります。詳細については、Amazon VPC ユーザーガイドの「VPC の削除」を参照してください。

VPC と関連リソースを削除する方法

1. DB サブネットグループを削除する。
    a. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
    b. [ナビゲーション] ペインで、[サブネットグループ] を選択します。
    c. 削除する DB サブネットグループを選択します。(例: tutorial-db-subnet-group)
    d. [Delete] (削除) を選択してから、確認ウィンドウの [Delete] (削除) を選択します。
2. VPC ID を書き留める。
    a. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
    b. [VPC ダッシュボード] を選択してから、[VPC] を選択します。
    c. リストで、作成した VPC を特定します。(例: tutorial-vpc)
    d. 作成した VPC の [VPC ID] をメモします。後続のステップで VPC ID が必要になります。
3. セキュリティグループを削除する。
    a. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。

b. [VPC Dashboard] (VPC ダッシュボード) を選択してから、[Security Groups] (セキュリティグループ) を選択します。

c. Amazon RDS DB インスタンスのセキュリティグループを選択します。(例: tutorial-db-securitygroup)

d. [Actions] (アクション) で、[Delete security groups] (セキュリティグループの削除) を選択してから、確認ページで [Delete] (削除) を選択します。

e. [Security Groups] (セキュリティグループ) ページで、Amazon EC2 インスタンスのセキュリティグループを選択します。(例: tutorial-securitygroup)

f. [Actions] (アクション) で、[Delete security groups] (セキュリティグループの削除) を選択してから、確認ページで [Delete] (削除) を選択します。

4. VPC を削除する。

a. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。

b. [VPC Dashboard] (VPC ダッシュボード) を選択してから、[VPC] を選択します。

c. 削除する VPC を選択します。(例: tutorial-vpc)

d. [アクション] で、[VPC の削除] を選択します。

確認ページには、VPC に関連付けられたサブネットを含め、削除される VPC に関連付けられているその他のリソースが表示されます。

e. 確認ページで、「**delete**」を入力してから、[Delete] (削除) を選択します。

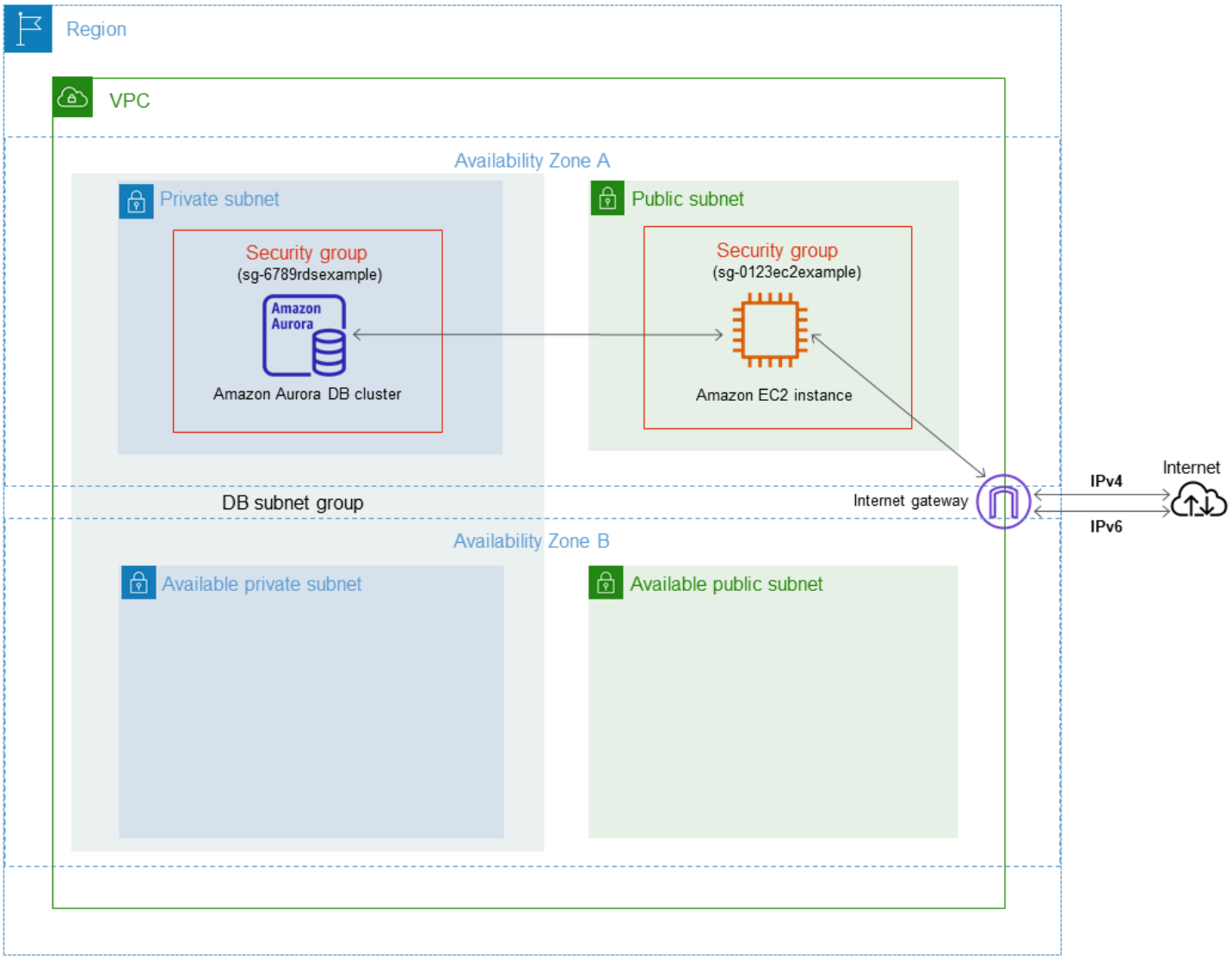
チュートリアル: DB クラスター用の VPC を作成する (デュアルスタックモード)

一般的なシナリオには、Amazon VPC サービスに基づく仮想プライベートクラウド (VPC) 内の DB クラスターが含まれます。この VPC は、同じ VPC で実行しているパブリック Amazon EC2 インスタンスとデータを共有します。

このチュートリアルでは、デュアルスタックモードで実行されているデータベースで動作する VPC を、このシナリオで作成します。IPv6 アドレッシングプロトコルを介した接続を可能にするデュアルスタックモード。IP アドレスの割り当てについては、「Amazon Aurora IP アドレス指定」を参照してください。

デュアルスタックのネットワーククラスターは、ほとんどのリージョンでサポートされています。詳細については、「デュアルスタックネットワーク DB クラスターの可用性」を参照してください。デュアルスタックモードの制限については、「デュアルスタックネットワーク DB クラスターの制限」を参照してください。

以下の図に、このシナリオを示しています。



その他のシナリオについては、VPC の DB クラスターにアクセスするシナリオ を参照してください。

DB クラスターは、Amazon EC2 インスタンスでのみ使用でき、パブリックインターネットから使用できないようにする必要があります。したがって、パブリックサブネットとプライベートサブネットを持つ VPC を作成します。Amazon EC2 インスタンスは、パブリックインターネットにアクセスできるようにパブリックサブネットでホストされます。DB クラスターはプライベートサブネットでホストされます。Amazon EC2 インスタンスは同じ VPC 内でホストされているため、DB クラスターに接続できます。ただし、DB クラスターはパブリックインターネットからは使用できないため、セキュリティが向上します。

このチュートリアルでは、別のアベイラビリティーゾーンに追加のパブリックサブネットとプライベートサブネットを設定します。これらのサブネットはチュートリアルでは使用されません。RDS

DB サブネットグループは、少なくとも 2 つのアベイラビリティーゾーン内のサブネットを必要とします。サブネットを追加すると、複数の Aurora DB インスタンスを簡単に設定できます。

デュアルスタックモードを使用する DB クラスターを作成するには、[Network type] (ネットワークタイプ) 設定として [Dual-stack mode] (デュアルスタックモード) を指定します。DB クラスターを同じ設定で変更することもできます。DB クラスター作成の詳細については、「Amazon Aurora DB クラスターの作成」を参照してください。DB クラスターの変更の詳細については、「Amazon Aurora DB クラスターの変更」を参照してください。

このチュートリアルでは、Amazon Aurora DB クラスター用に VPC を設定する方法について説明します。Amazon VPC の詳細については、「Amazon VPC ユーザーガイド」を参照してください。

プライベートサブネットおよびパブリックサブネットを持つ VPC を作成する

以下の手順で、パブリックサブネットとプライベートサブネットを持つ VPC を作成します。

VPC とサブネットを作成するには

1. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
2. AWS Management Console の右上隅で、VPC を作成するリージョンを選択します。この例では、米国東部 (オハイオ) リージョンを使用します。
3. 左上隅の [VPC dashboard] (VPC ダッシュボード) を選択します。VPC の作成を開始するには、[Create VPC] (VPC の作成) を選択します。
4. [VPC Settings] (VPC 設定) の [Resources to create] (作成するリソース) で、[VPC and more] (VPC など) を選択します。
5. 残りの [VPC settings] (VPC 設定) で、これらの値を設定します。

   - [Name tag auto-generation] (ネームタグ自動生成) – **`tutorial-dual-stack`**
   - [IPv4 CIDR block] (IPv4 CIDR ブロック) – **`10.0.0.0/16`**
   - [IPv6 CIDR block] (IPv6 CIDR ブロック) – [Amazon-provided IPv6 CIDR block] (Amazon 提供の IPv6 CIDR ブロック)
   - [Tenancy] (テナンシー) – デフォルト
   - [Number of Availability Zones (AZs)] (アベイラビリティーゾーンの数 (AZ)) – 2
   - [Customize AZs] (AZ をカスタマイズする) – デフォルト値を維持します。
   - [Number of public subnet] (パブリックサブネット数) – 2
   - [Number of private subnets] (プライベートサブネット数) – 2

- [Customize subnets CIDR blocks] (サブネット CIDR ブロックをカスタマイズ) — デフォルト値を維持します。
- [NAT gateways ($)] (NAT ゲートウェイ ($)) – なし
- [Egress only internet gateway] (Egress-only インターネットゲートウェイ): [No] (なし)
- [VPC endpoints] (VPC エンドポイント) – なし
- [DNS options] (DNS オプション) — デフォルト値を維持します。

> **Note**
>
> Amazon RDS では、マルチ AZ DB インスタンス配置をサポートするために、2 つの異なるアベイラビリティーゾーン内のサブネットを少なくとも 2 つ含んでいる必要があります。このチュートリアルではシングル AZ 配置を作成しますが、この要件により将来的に マルチ AZ DB インスタンス配備に簡単に変換できます。

6. [VPC の作成] を選択します。

パブリック Amazon EC2 インスタンスの VPC セキュリティグループを作成する

次に、パブリックアクセスのためのセキュリティグループを作成します。VPC 内のパブリック EC2 インスタンスに接続するには、インターネットから接続するトラフィックを許可するインバウンドルールを VPC セキュリティグループに追加します。

VPC セキュリティグループを作成するには

1. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
2. [VPC ダッシュボード]、[セキュリティグループ]、[セキュリティグループの作成] の順に選択します。
3. [セキュリティグループの作成] ページで、以下の値を設定します。
   - セキュリティグループ名: **`tutorial-dual-stack-securitygroup`**
   - 説明: **`Tutorial Dual-Stack Security Group`**
   - [VPC ID]: 前に作成した VPC を選択します (例: [vpc-***`identifier`*** (tutorial-dual-stack-vpc)])。
4. インバウンドルールをセキュリティグループに追加します。
   a. Secure Shell (SSH) を使用して VPC の EC2 インスタンスへの接続に使用する IP アドレスを決定します。

インターネットプロトコルバージョン 4 (IPv4) アドレスの例は `203.0.113.25/32` です。インターネットプロトコルバージョン 6 (IPv6) のアドレス範囲の例は `2001:db8:1234:1a00::/64` です。

多くの場合、インターネットサービスプロバイダー (ISP) 経由、またはファイアウォールの内側から静的 IP アドレスなしで接続することがあります。この場合は、クライアントコンピュータが使用する IP アドレスの範囲を検索します。

> **Warning**
>
> IPv4 の `0.0.0.0/0` または IPv6 の `::0` を使用している場合は、すべての IP アドレスが SSH を使ってパブリックインスタンスにアクセスできるようにします。この方法は、テスト環境で短時間なら許容できますが、実稼働環境では安全ではありません。実稼働環境では、特定の IP アドレスまたは特定のアドレス範囲にのみ、インスタンスへのアクセスを許可します。

b. [インバウンドルール] セクションで、[ルールの追加] を選択します。

c. 新しいインバウンドルールに次の値を設定して、Amazon EC2 インスタンスへの Secure Shell (SSH) アクセスを許可します。このようにした場合、EC2 インスタンスに接続して SQL クライアントやその他のアプリケーションをインストールできます。EC2 インスタンスへのアクセスできるように IP アドレスを指定します。

- [Type] (タイプ): **`SSH`**
- [Source] (ソース): ステップ a で指定した IP アドレスまたは範囲。IPv4 IP アドレスの例は **`203.0.113.25/32`** です。IPv6 IP アドレスの例は **`2001:DB8::/32`** です。

5. セキュリティグループを作成するには、[Create security group] (セキュリティグループの作成) を選択します。

   セキュリティグループ ID を書き留めます。このチュートリアルで後に必要になります。

プライベート DB クラスターの VPC セキュリティグループを作成する

DB クラスターをプライベートのままにするには、プライベートアクセス用の第 2 のセキュリティグループを作成します。VPC 内のプライベート DB クラスターに接続するには、VPC セキュリティグループにインバウンドルールを追加します。これにより、Amazon EC2 インスタンスからのトラフィックのみを許可します。

VPC セキュリティグループを作成するには

1. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
2. [VPC ダッシュボード]、[セキュリティグループ]、[セキュリティグループの作成] の順に選択します。
3. [セキュリティグループの作成] ページで、以下の値を設定します。
   - セキュリティグループ名: **tutorial-dual-stack-db-securitygroup**
   - 説明: **Tutorial Dual-Stack DB Instance Security Group**
   - [VPC ID]: 前に作成した VPC を選択します (例: [vpc-***identifier*** (tutorial-dual-stack-vpc)])。
4. インバウンドルールをセキュリティグループに追加します。
   a. [インバウンドルール] セクションで、[ルールの追加] を選択します。
   b. 新しいインバウンドルールに次の値を設定して、Amazon EC2 インスタンスからポート 3306 への MySQL トラフィックを許可します。その場合、EC2 インスタンスから DB クラスターに接続できます。これにより、EC2 インスタンスからデータベースにデータを送信できるようになります。
      - [Type] (タイプ): MySQL/Aurora
      - [Source] (ソース): このチュートリアルで以前に作成した tutorial-dual-stack-securitygroup セキュリティグループの ID (例: sg-9edd5cfb)。
5. セキュリティグループを作成するには、[セキュリティグループの作成] を選択します。

DB サブネットグループを作成する

DB サブネットグループは VPC に作成するサブネットのコレクションで、DB クラスター用に指定します。DB サブネットグループを使用することにより、DB クラスターを作成するときに、特定の VPC を指定することができます。DUAL 互換の DB サブネットグループを作成するには、すべてのサブネットが DUAL 互換である必要があります。DUAL 互換であるためには、サブネットに IPv6 CIDR が関連付けられている必要があります。

DB サブネットグループを作成するには

1. VPC 内のデータベースのプライベートサブネットを特定します。
   a. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。

b. [VPC Dashboard] (VPC ダッシュボード) を選択してから、[Subnets] (サブネット) を選択します。

c. tutorial-dual-stack-subnet-private1-us-west-2a と tutorial-dual-stack-subnet-private2-us-west-2b という名前のサブネット ID に注意してください。

サブネット ID は、DB サブネットグループを作成するときに必要になります。

2. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。

Amazon VPC コンソールではなく、Amazon RDS コンソールに接続してください。

3. [Navigation] ペインで、[Subnet groups] を選択します。
4. [Create DB subnet group] (DB サブネットグループの作成) を選択します。
5. [DB サブネットグループを作成する] ページで、[サブネットグループの詳細] に値を設定します。

- 名前: **`tutorial-dual-stack-db-subnet-group`**
- 説明: **`Tutorial Dual-Stack DB Subnet Group`**
- VPC: tutorial-dual-stack-vpc (vpc-***`identifier`***)

6. [Add subnets] (サブネットの追加) セクションで、[Availability Zones] (アベイラビリティーゾーン) オプションと [Subnets] (サブネット) オプションの値を選択します。

このチュートリアルでは、[Availability Zones] (アベイラビリティーゾーン) として [us-east-2a] と [us-east-2b] を選択します。[Subnets] (サブネット) では、前のステップで特定したプライベートサブネットを選択します。

7. [Create] (作成) を選択します。

RDS コンソールの DB サブネットグループリストに新しい DB サブネットグループが表示されます。DB サブネットグループを選択して詳細を表示できます。これには、サポートされているアドレス指定プロトコルと、そのグループに関連付けられたすべてのサブネット、DB サブネットグループによってサポートされるネットワークタイプが含まれます。

デュアルスタックモードの Amazon EC2 インスタンスを作成する

Amazon EC2 インスタンスを作成するには、Linux インスタンス向け Amazon EC2 ユーザーガイドの「新しい起動インスタンスウィザードを使用したインスタンスの起動」の指示に従います。

次に示すように、[Configure Instance Details] (インスタンスの詳細の設定) ページで次の値を設定し、他の値はデフォルトのままにします。

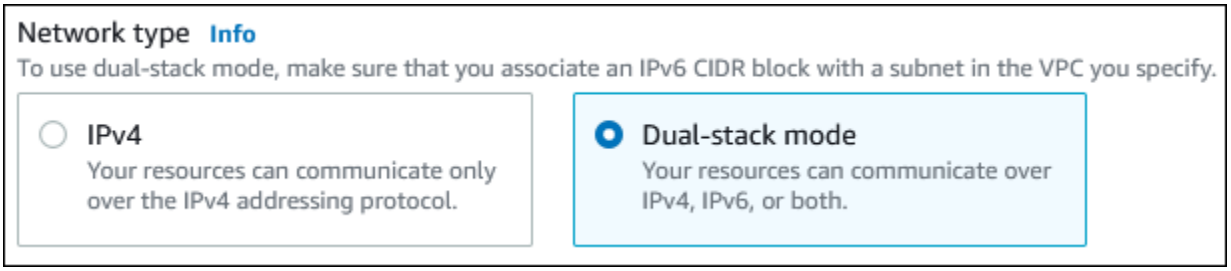
- ネットワーク – パブリックサブネットとプライベートサブネットの両方を持つ既存の VPC を選択します (プライベートサブネットおよびパブリックサブネットを持つ VPC を作成する で作成した tutorial-dual-stack-vpc (vpc-*identifier*) など)。
- [Subnet] (サブネット): 既存のパブリックサブネットを選択します (パブリック Amazon EC2 インスタンスの VPC セキュリティグループを作成する で作成した subnet-***identifier*** | tutorial-dual-stack-subnet-public1-us-east-2a | us-east-2a など)。
- [Auto-assign Public IP] (パブリック IP の自動割り当て)]: [Enable] (有効化) を選択します。
- [Auto-assign IPv6 IP]: [Enable] (有効化) を選択します。
- [Firewall (security groups)] (ファイアウォール (セキュリティグループ)) — [Select an existing security group] (既存のセキュリティグループを選択する) を選択します。
- [Common security groups] (共通セキュリティグループ) — `tutorial-securitygroup` で作成された パブリック Amazon EC2 インスタンスの VPC セキュリティグループを作成する などの既存のセキュリティグループを選択します。選択するセキュリティグループに、Secure Shell (SSH) および HTTP アクセスのインバウンドルールが含まれていることを確認します。

デュアルスタックモードの DB クラスターを作成する

このステップでは、デュアルスタックモードで実行する DB クラスターを作成します。

DB インスタンスを作成するには

1. AWS Management Console にサインインし、Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
2. コンソールの右上隅で、DB クラスターを作成する AWS リージョン を選択します。この例では、米国東部 (オハイオ) リージョンを使用します。
3. ナビゲーションペインで、[データベース] を選択します。
4. [Create database] (データベースを作成) を選択します。
5. [Create database] (データベースの作成) ページで、[Standard create] (スタンダード作成) オプションがオンになっていることを確認し、[Aurora] MySQL DB を選択します。
6. [接続] セクションで、次の値を設定します。
    - [Network type] (ネットワークタイプ): [Dual-stack mode] (デュアルスタックモード) を選択します。



- [Virtual private cloud (VPC)] (仮想プライベートクラウド (VPC)): パブリックサブネットとプライベートサブネットの両方を持つ既存の VPC を選択します (プライベートサブネットおよびパブリックサブネットを持つ VPC を作成する で作成した tutorial-dual-stack-vpc (vpc-*identifier*) など)。

  VPC の各サブネットは異なるアベイラビリティーゾーンに存在している必要があります。
- [DB Subnet group] (DB サブネットグループ): VPC の DB サブネットグループ (DB サブネットグループを作成する で作成した tutorial-dual-stack-db-subnet-group など)。
- [Public access] (公開アクセス) — [No] (いいえ) を選択します。
- [VPC security group (firewall)] (VPC セキュリティグループ (ファイアウォール)) — [Choose existing] (既存を選択) を選択します。
- [Existing VPC security groups] (既存の VPC セキュリティグループ) – プライベートアクセス用に設定されている既存の VPC セキュリティグループを選択します (プライベート DB クラスターの VPC セキュリティグループを作成する で作成した tutorial-dual-stack-db-securitygroup など)。

  他のセキュリティグループ (デフォルトのセキュリティグループなど) は、それぞれの対応する [X] を選択して削除します。
- [Availability zone] (アベイラビリティーゾーン): us-west-2a を選択します。

  AZ 間のトラフィックを回避するには、DB インスタンスと EC2 インスタンスが同じアベイラビリティーゾーンにあることを確認してください。

7. 残りのセクションで、DB クラスター設定を指定します。各設定の詳細については、「Aurora DB クラスターの設定」を参照してください。

Amazon EC2 インスタンスと DB クラスターに接続する

Amazon EC2 インスタンスと DB クラスターをデュアルスタックモードで作成した後、IPv6 プロトコルを使用して各インスタンスに接続できます。IPv6 プロトコルを使用して Amazon EC2 インスタ

ンスに接続するには、Linux インスタンス用 Amazon EC2 ユーザーガイドの「Linux インスタンスに接続する」の手順に従ってください。

Amazon EC2 インスタンスから Aurora MySQL DB クラスターに接続するには、「Aurora MySQL DB クラスターに接続する」の手順に従ってください。

VPC の削除

このチュートリアルの VPC およびその他のリソースを作成後、不要になった場合は、削除できます。

このチュートリアルで作成した VPC にリソースを追加した場合は、VPC を削除する前にこれらを削除しなければならない場合があります。リソースの例としては、Amazon EC2 インスタンスや DB クラスターなどがあります。詳細については、Amazon VPC ユーザーガイドの「VPC の削除」を参照してください。

VPC と関連リソースを削除する方法

1. DB サブネットグループを削除するには、次のようにします。
    a. Amazon RDS コンソール (https://console.aws.amazon.com/rds/) を開きます。
    b. [ナビゲーション] ペインで、[サブネットグループ] を選択します。
    c. 削除する DB サブネットグループを選択します ([tutorial-db-subnet-group] など)。
    d. [削除] を選択してから、確認ウィンドウの [削除] を選択します。
2. 次のようにして、VPC ID をメモします。
    a. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
    b. [VPC ダッシュボード] を選択してから、[VPC] を選択します。
    c. リストで、作成した VPC を特定します ([tutorial-dual-stack-vpc] など)。
    d. 作成した VPC の [VPC ID] の値をメモします。後続のステップで、この VPC ID が必要になります。
3. セキュリティグループを削除するには、次のようにします。
    a. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。
    b. [VPC ダッシュボード] を選択してから、[セキュリティグループ] を選択します。
    c. Amazon RDS DB インスタンスのセキュリティグループを選択します ([tutorial-dual-stack-db-securitygroup] など)。

d. [Actions] (アクション) で、[Delete security groups] (セキュリティグループの削除) を選択してから、確認ページで [Delete] (削除) を選択します。

e. [Security Groups] (セキュリティグループ) ページで、Amazon EC2 インスタンスのセキュリティグループを選択します ([tutorial-dual-stack-securitygroup] など)。

f. [Actions] (アクション) で、[Delete security groups] (セキュリティグループの削除) を選択してから、確認ページで [Delete] (削除) を選択します。

4. 次のようにして、NAT ゲートウェイを削除します。

a. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。

b. [VPC ダッシュボード] を選択してから、[NAT ゲートウェイ] を選択します。

c. 作成した VPC の NAT ゲートウェイを選択します。VPC ID を使用して、適切な NAT ゲートウェイを識別します。

d. [Actions] (アクション) で、[Delete NAT gateway] (NAT ゲートウェイの削除) を選択します。

e. 確認ページで、「**delete**」を入力してから、[削除] を選択します。

5. VPC の削除

a. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。

b. [VPC ダッシュボード] を選択してから、[VPC] を選択します。

c. 削除する VPC を選択します ([tutorial-dual-stack-vpc] など)。

d. [アクション] で、[VPC の削除] を選択します。

確認ページには、VPC に関連付けられたサブネットを含め、削除される VPC に関連付けられているその他のリソースが表示されます。

e. 確認ページで、「**delete**」を入力してから、[Delete] (削除) を選択します。

6. 次のようにして、Elastic IP アドレスを解放します。

a. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。

b. [EC2 ダッシュボード] を選択してから、[Elastic IP] を選択します。

c. 解放する Elastic IP アドレスを選択します。

d. [Actions] (アクション) で、[Release Elastic IP addresses] (Elastic IP アドレスの解放) を選択します。

e. 確認ページで、[リリース] を選択します。

Amazon Aurora のクォータと制約

Amazon Aurora のリソースのクォータと名前付け制約の説明は次のとおりです。

トピック

Amazon Aurora のクォータ

各 AWS アカウントには、AWS リージョン別に、作成できる Amazon Aurora リソースの数に適用されるクォータがあります。リソースのクォータに達すると、そのリソースを作成するための追加の呼び出しは、失敗して例外が発生します。

次の表に、AWS リージョンごとのリソースとそのクォータを示します。

名前	デフォルト	引き上げ可能	説明
DB セキュリティグループごとの承認	サポートされている各リージョン: 20	いいえ	DB セキュリティグループあたりのセキュリティグループ認可数
カスタムエンジンバージョン	サポートされている各リージョン: 40	はい	現在のリージョンでこのアカウントに許可されるカスタムエンジンバージョンの最大数
DB クラスターのパラメータグループ	サポートされている各リージョン: 50	いいえ	DB クラスターパラメータグループの最大数

名前	デフォルト	引き上げ可能	説明
DB クラスター	サポートされている各リージョン: 40	はい	現在のリージョンでこのアカウントに許可される Aurora クラスターの最大数
DB インスタンス	サポートされている各リージョン: 40	はい	現在のリージョンでこのアカウントに許可される DB インスタンスの最大数
DB サブネットグループ	サポートされている各リージョン: 50	はい	DB サブネットグループの最大数
Data API HTTP リクエスト本文のサイズ	サポートされている各リージョン: 4 MB	いいえ	HTTP リクエスト本文に許可される最大サイズ。
Data API 最大同時実行クラスターシークレットペア	サポートされている各リージョン: 30	いいえ	AWS リージョンの現在のアカウントにおける同時実行 Data API リクエストでの Aurora Serverless v1 DB クラスターとシークレットの一意のペアの最大数。

名前	デフォルト	引き上げ可能	説明
Data API 最大同時実行リクエスト	サポートされている各リージョン: 500	いいえ	Aurora Serverless v1 DB クラスターに対する Data API リクエストのうち、同じシークレットを使用し、同時に処理可能であるものの最大数。追加のリクエストはキューに入れられ、処理中のリクエストが完了すると処理されます。
データ API の結果セットの最大サイズ	サポートされている各リージョン: 1 MB	いいえ	Data API によって返されるデータベース結果セットの最大サイズ。
データ API の JSON レスポンス文字列の最大サイズ	サポートされている各リージョン: 10 MB	いいえ	RDS データ API によって返される簡略化された JSON レスポンス文字列の最大サイズ。
1 秒あたりのデータ API リクエスト数	サポートされている各リージョン: 1,000/秒	いいえ	現在の AWS リージョンにおける現在のアカウントで許可されている、Data API に対するリクエスト/秒の最大数 このクォータは、Amazon Aurora Serverless v1 クラスターにのみ適用されます。

名前	デフォルト	引き上げ可能	説明
イベントサブスクリプション	サポートされている各リージョン: 20	はい	イベントサブスクリプションの最大数
DB クラスターごとの IAM ロール	サポートされている各リージョン : 5	はい	DB クラスターに関連付けられる IAM ロールの最大数
DB インスタンスごとの IAM ロール	サポートされている各リージョン : 5	はい	DB インスタンスに関連付けられる IAM ロールの最大数
手動 DB クラスタースナップショット	サポートされている各リージョン: 100	はい	手動 DB クラスタースナップショットの最大数
手動の DB インスタンスのスナップショット	サポートされている各リージョン: 100	はい	手動 DB インスタンススナップショットの最大数
オプショングループ	サポートされている各リージョン: 20	はい	オプショングループの最大数
パラメータグループ	サポートされている各リージョン: 50	はい	パラメータグループの最大数
プロキシ	サポートされている各リージョン: 20	はい	現在の AWS リージョンでこのアカウントに許可されるプロキシの最大数

名前	デフォルト	引き上げ可能	説明
プライマリあたりのリードレプリカ数	サポートされている各リージョン: 15	はい	プライマリ DB インスタンスあたりのリードレプリカの最大数。このクォータは、Amazon Aurora 用に調整できません。
リザーブド DB インスタンス	サポートされている各リージョン: 40	はい	現在の AWS リージョンでこのアカウントに許可される予約 DB インスタンスの最大数
セキュリティグループあたりのルールの数	サポートされている各リージョン: 20	いいえ	DB セキュリティグループあたりのルールの最大数
セキュリティグループ	サポートされている各リージョン: 25	はい	DB セキュリティグループの最大数
セキュリティグループ (VPC)	サポートされている各リージョン : 5	いいえ	Amazon VPC あたりの DB セキュリティグループの最大数
DB サブネットグループあたりのサブネット	サポートされている各リージョン: 20	いいえ	DB サブネットグループあたりのサブネットの最大数
リソースあたりのタグ	サポートされている各リージョン: 50	いいえ	Amazon RDS リソースあたりのタグの最大数

名前	デフォルト	引き上げ可能	説明
すべての DB インスタンスの合計ストレージ	サポートされている各リージョン: 100,000 GB	はい	一緒に追加されたすべての Amazon RDS DB インスタンスの EBS ボリュームの最大ストレージ合計 (GB 単位)。このクォータは、各 DB クラスターの最大クラスターボリュームが 128 TiB である Amazon Aurora には適用されません。

> **Note**
>
> デフォルトでは、最大で合計 40 の DB インスタンスを持つことができます。RDS DB インスタンス、Aurora DB インスタンス、Amazon Neptune インスタンス、および Amazon DocumentDB インスタンスは、このクォータに該当します。
> アプリケーションでさらに多くの DB インスタンスが必要な場合は、Service Quotas コンソールを開いて、追加の DB インスタンスをリクエストできます。ナビゲーションペインで、[AWS のサービス] を選択します。[Amazon Relational Database Service (Amazon RDS)] を選択してクォータを選択し、指示に従ってクォータの引き上げをリクエストします。詳細については、「Service Quotas ユーザーガイド」の「クォータの引き上げのリクエスト」を参照してください。
> AWS Backup によって管理されるバックアップは手動 DB クラスタースナップショットと見なされますが、手動クラスタースナップショットクォータにはカウントされません。AWS Backup の詳細については、『AWS Backup デベロッパーガイド』を参照してください。

いずれかの RDS API オペレーションを使用して、1 秒あたりの呼び出し数のデフォルトのクォータを超えると、Amazon RDS API では次のようなエラーを発行します。

ClientError: *API_Name* オペレーションの呼び出し時にエラー (ThrottlingException) が発生しました (レート超過)。

この場合、1 秒あたりのコール回数を減らします。クォータは、ほとんどのユースケースをカバーするようにしてあります。より大きなクォータが必要な場合は、次のいずれかのオプションを使用してクォータの引き上げをリクエストできます。

- コンソールで、[Service Quotas コンソール] を開きます。
- AWS CLI で、AWS CLI コマンド request-service-quota-increase を使用します。

詳細については、Service Quotas ユーザーガイドを参照してください。

Amazon Aurora の命名に関する制約

次の表に、Amazon Aurora の命名に関する制約を示します。

リソースまたは項目	制約
DB クラスター識別子	識別子には、以下の命名に関する制約があります。 • 1 ~ 63 個の英数字またはハイフンを使用する必要があります。 • 1 字目は文字である必要があります。 • 文字列の最後にハイフンを使用したり、ハイフンを 2 つ続けて使用したりすることはできません。 • 1 つの AWS アカウント、1 つの AWS リージョンにつき、すべての DB インスタンスにおいて一意である必要があります。
初期データベース 名	データベース名の制約は、 Aurora MySQL と PostgreSQL 間で異なります。詳細については、各 DB クラスターの作成時に使用できる設定を参照してください。
マスターユーザー名	マスターユーザー名の制約は、データベースエンジンごとに異なります。詳細については、各 DB クラスターの作成時に使用できる設定を参照してください。

リソースまたは項目	制約
マスターパスワード	データベースのマスターユーザーのパスワードには、すべての印刷可能な ASCII 文字 (/、'、"、@、またはスペースを除く) を使用できます。Oracle の場合、& は追加の文字制限です。パスワードには、DB エンジンに応じて、次の数の印字可能な ASCII 文字が含まれます。 • Aurora MySQL: 8–41 • Aurora PostgreSQL: 8–99
DB パラメータグループ名	これらの名前には、以下の制約があります。 • 1 ~ 255 個の英数字を使用する必要があります。 • 1 字目は文字である必要があります。 • この名前では、ハイフンを使用できますが、末尾に使用したり、2 つ続けて使用したりすることはできません。
DB サブネットグループ名	これらの名前には、以下の制約があります。 • 1 ~ 255 文字を使用する必要があります。 • 英数字、スペース、ハイフン、アンダースコア、ピリオドを使用できます。

Amazon Aurora サイズ制限

ストレージサイズの制限

次のエンジンバージョンでは、Aurora クラスターボリュームは最大サイズの 128 tebibytes (TiB) まで拡張できます。

- Aurora MySQL バージョン 3 の使用可能なすべてのバージョン、Aurora MySQL バージョン 2 (バージョン 2.09 以降)
- 使用可能なすべての Aurora PostgreSQL バージョン

より低いエンジンバージョンでは、Aurora クラスターボリュームの最大サイズは 64 TiB です。詳細については、「Aurora ストレージのサイズを自動的に変更する方法」を参照してください。

残りのストレージ領域をモニタリングするには、AuroraVolumeBytesLeftTotal メトリクスを使用できます。詳細については、「Amazon Aurora のクラスターレベルのメトリクス」を参照してください。

SQL テーブルサイズ制限

Aurora MySQL DB クラスターの場合、テーブルの最大サイズは 64 テビバイト (TiB) です。Aurora PostgreSQL DB クラスターの場合、テーブルの最大サイズは 32 テビバイト (TiB) です。大きいテーブルの分割など、テーブル設計のベストプラクティスにしたがうことをお勧めします。

テーブルスペース ID の制限事項

Aurora MySQL の最大テーブルスペース ID は 2147483647 です。テーブルの作成および削除を頻繁に行う場合は、テーブルスペース ID を意識し、論理ダンプを計画的に使用してください。詳細については、「mysqldump を使用した MySQL から Amazon Aurora MySQL への論理的移行」を参照してください。

Amazon Aurora のトラブルシューティング

以下のシナリオを使用して、Amazon RDS や Amazon Aurora の DB インスタンスで発生する問題をトラブルシューティングします。

トピック

Amazon RDS API を使用した問題のデバッグについては、「Aurora のアプリケーションのトラブルシューティング」を参照してください。

Amazon RDS DB インスタンスに接続できない

DB インスタンスに接続できない場合、一般的な原因として次のようなものがあります。

- インバウンドルール - ローカルのファイアウォールによって適用されているアクセスルールと DB インスタンスへのアクセスが許可された IP アドレスが一致していない可能性があります。問題の原因として最も多いのは、セキュリティグループのインバウンドルールです。

  デフォルトでは、DB インスタンスへのアクセスは許可されていません。アクセスは、DB インスタンスとの間のトラフィックを許可する VPC に関連付けられたセキュリティグループによって許可されます。必要に応じて、特定の状況のインバウンドルールとアウトバウンドルールをセキュリティグループに追加します。IP アドレス、IP アドレスの範囲、または別の VPC セキュリティグループを指定できます。

  > **Note**
  >
  > 新しいインバウンドルールを追加するときに [出典] に [マイ IP] を選択すると、ブラウザで検出された IP アドレスから DB インスタンスへのアクセスを許可できます。

セキュリティグループの設定の詳細については、「セキュリティグループを作成して VPC 内の DB クラスターへのアクセスを提供する」を参照してください。

> **Note**
>
> 169.254.0.0/16 の範囲内の IP アドレスからのクライアント接続は許可されていません。これは、ローカルリンクのアドレス指定に使用される Automatic Private IP Addressing Range (APIPA) です。

- パブリックアクセス - クライアントアプリケーションを使用するなど、VPC の外部から DB インスタンスに接続するには、インスタンスにパブリック IP アドレスが割り当てられている必要があります。

  インスタンへのパブリックアクセスを許可するには、インスタンスを変更し、[Public accessibility (パブリックアクセス)] で [Yes (はい)] を選択します。詳細については、「VPC 内の DB クラスターをインターネットから隠す」を参照してください。
- ポート - DB インスタンスの作成時に指定したポートが、ローカルのファイアウォールの制限によって通信の送受信に使用できない場合があります。指定したポートをインバウンドおよびアウトバウンドの通信に使用することがネットワークで許可されているかどうかを判断するには、ネットワーク管理者に確認します。
- 可用性 - 新しく作成された DB インスタンスでは、DB インスタンスが使用できるようになるまで、DB インスタンスのステータスは `creating` になります。ステータスが `available` になると、DB インスタンスに接続できます。DB インスタンスのサイズによっては、インスタンスが利用可能になるまでに最大 20 分かかることがあります。
- 内部ゲートウェイ - DB インスタンスをパブリックにアクセス可能にするには、その DB サブネットグループのサブネットにインターネットゲートウェイが必要です。

  サブネットのインターネットゲートウェイを設定するには

  1. AWS Management Console にサインインし、Amazon RDS コンソール https://console.aws.amazon.com/rds/ を開きます。
  2. ナビゲーションペインで、[データベース] を選択し、DB インスタンスの名前を選択します。
  3. [接続とセキュリティ] タブで、[VPC] の下の VPC ID と [サブネット] の下のサブネット ID の値を書き留めます。
  4. Amazon VPC コンソール (https://console.aws.amazon.com/vpc/) を開きます。

5. ナビゲーションペインで、[Internet Gateways] を選択します。ご使用の VPC にアタッチされているインターネットゲートウェイがあることを確認します。それ外の場合は、[Create Internet Gateway] を選択してインターネットゲートウェイを作成します。インターネットゲートウェイを選択し、[VPC にアタッチ] を選択して指示どおりにインターネットゲートウェイを VPC にアタッチします。
6. ナビゲーションペインで [Subnets] を選択し、サブネットを選択します。
7. [Route Table] タブで、送信先として `0.0.0.0/0`、ターゲットとして VPC のインターネットゲートウェイが指定されたルートがあることを確認します。

   IPv6 アドレスを使用してインスタンスに接続する場合は、インターネットゲートウェイを指しているすべての IPv6 トラフィック (`::/0`) 用のルートがあることを確認します。それ以外の場合は、以下の作業を行います。

   a. ルートテーブルの ID (rtb-xxxxxxxx) を選択して、ルートテーブルに移動します。
   b. [Routes] タブで、[Edit routes] を選択します。[Add route] を選択して、`0.0.0.0/0` を送信先として追加し、インターネットゲートウェイをターゲットとして使用します。

      IPv6 の場合は、[Add route] を選択して、`::/0` を送信先として追加し、インターネットゲートウェイをターゲットとして使用します。
   c. [ルートの保存] を選択します。

   また、IPv6 エンドポイントに接続する場合は、クライアントの IPv6 アドレス範囲が DB インスタンスへの接続を認可されていることを確認してください。

詳細については、「VPC 内の DB クラスターの使用」を参照してください。

DB インスタンスへの接続のテスト

一般的な Linux または Microsoft Windows のツールを使用して DB インスタンスへの接続をテストできます。

Linux または UNIX のターミナルからは、次のように入力することで接続をテストできます。*DB-instance-endpoint* をエンドポイントに、*port* を DB インスタンスのポートに置き換えます。

```
nc -zv DB-instance-endpoint port
```

コマンドと戻り値の例を以下に示します。

```
nc -zv postgresql1.c6c8mn7fake0.us-west-2.rds.amazonaws.com 8299

  Connection to postgresql1.c6c8mn7fake0.us-west-2.rds.amazonaws.com 8299 port [tcp/
vvr-data] succeeded!
```

Windows ユーザーは、Telnet を使用して DB インスタンスへの接続をテストできます。Telnet での操作は、接続のテスト以外はサポートされていないことに注意してください。接続に成功した場合、このアクションはメッセージを返しません。接続に失敗した場合は、次のようなエラーメッセージが返されます。

```
C:\>telnet sg-postgresql1.c6c8mntfake0.us-west-2.rds.amazonaws.com 819

  Connecting To sg-postgresql1.c6c8mntfake0.us-west-2.rds.amazonaws.com...Could not
 open
  connection to the host, on port 819: Connect failed
```

Telnet アクションが成功を示している場合、セキュリティグループは適切に設定されています。

> **Note**
>
> Amazon RDS は ping などの ICMP (Internet Control Message Protocol) トラフィックを受け付けません。

接続認証のトラブルシューティング

場合によっては、DB インスタンスに接続できても認証エラーが発生することがあります。このような場合、DB インスタンスのマスターユーザーパスワードのリセットが必要になることがあります。これを行うには、RDS インスタンスを変更します。

Amazon RDS のセキュリティの問題

セキュリティ問題を回避するために、マスター AWS ユーザー名とパスワードをユーザーアカウントに使用しないでください。ベストプラクティスは、マスター AWS アカウント を使用してユーザーを作成し、DB ユーザーアカウントに割り当てることです。また、必要に応じて、マスターアカウントを使用して他のユーザーアカウントを作成することもできます。

ユーザーの作成の詳細については、「AWS アカウント での IAM ユーザーの作成」を参照してください。AWS IAM Identity Center でのユーザー作成の詳細については、「Manage identities in IAM Identity Center」(IAM Identity Center での ID の管理) を参照してください。

エラーメッセージ「アカウント属性の取得に失敗しました。コンソールの特定の機能が損なわれる可能性があります。」

このエラーが発生する原因はいくつかあります。アカウントにアクセス許可がないか、アカウントが正しく設定されていない可能性があります。新規のアカウントの場合は、アカウントの準備がまだ整っていない可能性があります。既存のアカウントの場合は、DB インスタンスの作成などの特定の操作を実行するためのアクセスポリシーでアクセス許可が設定されてない可能性があります。問題を修正するには、管理者が必要なロールをアカウントに与える必要があります。詳細については、「IAM ドキュメント」を参照してください。

DB インスタンス所有者のパスワードのリセット

DB クラスターからロックアウトされた場合は、マスターユーザーとしてログインできます。その後、他の管理ユーザーまたはロールの認証情報をリセットできます。マスターユーザーとしてログインできない場合は、AWS アカウントの所有者がマスターユーザーのパスワードをリセットできます。リセットする必要がある管理アカウントまたはロールの詳細については、「マスターユーザーアカウント権限」を参照してください。

DB インスタンスのパスワードは、Amazon RDS コンソール、AWS CLI コマンドの modify-db-instance、または ModifyDBInstance API オペレーションを使用して変更できます。DB クラスターの DB インスタンスの変更の詳細については、「DB クラスター内の DB インスタンスの変更」を参照してください。

Amazon RDS DB インスタンスの停止または再起動

DB インスタンスの停止は、DB インスタンスが再起動されたときに発生する可能性があります。また、DB インスタンスがアクセスできない状態になったときやデータベースが再開されたときに発生する可能性があります。再起動は、DB インスタンスを手動で再起動した場合に発生することがあります。また、DB インスタンスの設定を設定を変更した場合、その設定を有効にするために再起動が必要になることがあります。

DB インスタンスの再起動は、再起動が必要な設定を変更したとき、または手動で再起動を実行したときにのみ行われます。設定を変更し、その変更を直ちに有効にすることをリクエストした場合、直

ちに再起動を実行できます。または、DB インスタンスのメンテナンス期間中に発生することもあります。

以下のいずれかが発生したとき、DB インスタンスの再起動は直ちに実行されます。

- DB インスタンスのバックアップ保持期間を 0 から 0 以外の値または 0 以外の値から 0 に変更します。次に、[Apply Immediately] (すぐに適用) を `true` に設定します。
- DB インスタンスクラスを変更し、[すぐに適用] を [`true`] に設定する。

以下のいずれかが発生したとき、DB インスタンスの再起動はメンテナンス時間中に実行されます。

- DB インスタンスのバックアップ保持期間を 0 から 0 以外の値または 0 以外の値から 0 に変更し、[すぐに適用] を [`false`] に設定する。
- DB インスタンスクラスを変更し、[すぐに適用] を [`false`] に設定する。

DB パラメータグループの静的パラメータを変更する場合、その変更はパラメータグループに関連付けられている DB インスタンスを再起動するまで有効になりません。この変更には、手動での再起動が必要です。DB インスタンスは、メンテナンスウィンドウ中に自動では再起動されません。

Amazon RDS DB パラメータの変更が有効にならない

場合によっては、DB パラメータグループのパラメータを変更しても、その変更が有効にならないことがあります。その場合は、DB パラメータグループに関連付けられている DB インスタンスの再起動が必要な可能性があります。動的パラメータを変更する場合は、その変更はすぐに有効になります。静的パラメータを変更する場合は、その変更はパラメータグループに関連付けられている DB インスタンスを再起動するまで有効になりません。

RDS コンソールを使用して DB インスタンスを再起動できます。または、`RebootDBInstance` API オペレーションを明示的に呼び出すこともできます。DB インスタンスがマルチ AZ 配置にある場合は、フェイルオーバーなしで再起動できます。静的パラメータの変更後に関連付けられている DB インスタンスの再起動を求める要件は、パラメータの誤った設定が API コールに影響を及ぼすリスクを低減するために役立ちます。例えば、`ModifyDBInstance` を呼び出して DB インスタンスクラスを変更する場合があります。詳細については、「DB パラメータグループのパラメータの変更」を参照してください。

Amazon Aurora の解放可能なメモリの問題

解放可能なメモリは、データベースエンジンで使用できる DB インスタンスの合計ランダムアクセスメモリ (RAM) です。これは、空きオペレーティングシステム (OS) メモリと使用可能なバッファとページキャッシュメモリの合計です。データベースエンジンは、ホスト上のほとんどのメモリを使用しますが、OS プロセスも一部の RAM を使用します。データベースエンジンに現在割り当てられているメモリや OS プロセスによって使用されているメモリは、空きメモリには含まれません。データベースエンジンのメモリが不足している場合、DB インスタンスは、バッファリングとキャッシュに通常使用される一時スペースを使用できます。前述のように、この一時スペースは空きメモリに含まれます。

Amazon CloudWatch の `FreeableMemory` メトリクスを使用して、空きメモリを監視します。詳細については、「Amazon Aurora のメトリクスのモニタリングの概要」を参照してください。

DB インスタンスの解放可能なメモリが常に不足またはスワップ領域を使用する場合、DB インスタンスクラスへのスケールアップを検討する必要があります。詳細については、「Aurora DB インスタンスクラス」を参照してください。

メモリ設定を変更することもできます。例えば、Aurora MySQL と指定すると、`innodb_buffer_pool_size`パラメータのサイズを調整することもできます。このパラメータはデフォルトで物理メモリの 75% に設定されています。MySQL のトラブルシューティングのヒントについては、「Amazon RDS for MySQL データベースの空きメモリ不足のトラブルシューティング方法を教えてください」を参照してください。

Aurora Serverless v2 の場合、`FreeableMemory` 値は、DB インスタンスを最大容量にスケーリングしたときに利用できる未使用のメモリ量を表します。インスタンスが比較的低い容量にスケールダウンしたかもしれませんが、インスタンスがスケールアップできるため、`FreeableMemory` に高い値が報告されます。そのメモリは現在利用できませんが、必要な場合は入手できます。

現在の容量が最大容量を下回る 各 Aurora 容量ユニット (ACU) では、`FreeableMemory` は約 2 GiB 増加します。したがって、DB インスタンスが限りなく大きくスケールアップされるまで、このメトリクスはゼロに近づきません。

このメトリクスが `0` 値に近づいた場合、DB インスタンスは限界までスケールアップしたことになります。これにより、使用可能なメモリの限界に近づいています。クラスターの最大 ACU 設定を引き上げることを検討してください。このメトリクスがリーダー DB インスタンスで `0` 値に近づいた場合、クラスターにリーダー DB インスタンスを追加することを検討してください。これにより、ワークロードの読み取り専用部分のワークロードをより多くの DB インスタンスに分散することで、各

リーダー DB インスタンスのメモリ使用量を軽減できます。詳細については、「Aurora Serverless v2 の重要な Amazon CloudWatch メトリクス」を参照してください。

Aurora Serverless v1 の場合、容量範囲を変更して ACU を使用できます。詳細については、「Aurora Serverless v1 DB クラスターの変更」を参照してください。

Amazon Aurora MySQL レプリケーションの問題

MySQL のレプリケーションの問題のいくつかは、Aurora MySQL にも該当します。これらの問題を診断して修正できます。

トピック

- リードレプリカ間の遅延の診断と解決
- MySQL のリードレプリケーションのエラーの診断と解決
- レプリケーション停止エラー

リードレプリカ間の遅延の診断と解決

MySQL のリードレプリカを作成してそのリードレプリカが使用可能になると、Amazon RDS はリードレプリカの作成オペレーションがスタートされた時点以降に出典の DB インスタンスに対して行われた変更を初期にレプリケートします。このフェーズでは、リードレプリカのレプリケーション遅延時間が 0 より大きくなります。これは、Amazon CloudWatch で Amazon RDS の `AuroraBinlogReplicaLag` メトリクスを参照することによってモニタリングできます。

`AuroraBinlogReplicaLag` メトリクスには、MySQL `Seconds_Behind_Master` コマンドの `SHOW REPLICA STATUS` フィールドの値が報告されます。詳細については、MySQL ドキュメントの「SHOW REPLICA STATUS ステートメント」を参照してください。

`AuroraBinlogReplicaLag` メトリクスが 0 に達すると、レプリカがソース DB インスタンスに追いついています。`AuroraBinlogReplicaLag` メトリクスが -1 を返す場合、レプリケーションがアクティブではない可能性があります。レプリケーションのエラーをトラブルシューティングする場合は、「MySQL のリードレプリケーションのエラーの診断と解決」を参照してください。`AuroraBinlogReplicaLag` の値が -1 である場合は、`Seconds_Behind_Master` の値が特定できないか `NULL` であることも示しています。

> **Note**
>
> Aurora MySQL の旧バージョンは SHOW REPLICA STATUS の代わりに SHOW SLAVE STATUS を使用していました。Aurora MySQL バージョン 1 または 2 を使用している場合は、SHOW SLAVE STATUS を使用してください。Aurora MySQL バージョン 3 以降は、SHOW REPLICA STATUS を使います。

AuroraBinlogReplicaLag メトリクスは、ネットワークが停止しているとき、またはメンテナンス時間にパッチを適用しているときには、-1 を返します。この場合、ネットワーク接続が復元されるか、メンテナンス時間が終了するまで待ってから、 AuroraBinlogReplicaLag メトリクスを再度確認します。

MySQL のリードレプリケーションテクノロジーは非同期です。そのため、出典の DB インスタンスの BinLogDiskUsage メトリクスやリードレプリカの AuroraBinlogReplicaLag メトリクスが増加する場合があります。例えば、出典の DB インスタンスへの大量の書き込みオペレーションがパラレルで実行される場合を例にします。このとき、リードレプリカへの書き込みオペレーションは単一の I/O スレッドでシリアルで行われます。このような場合、出典のインスタンスとリードレプリカの間に遅延が発生する可能性があります。

リードレプリカと MySQL の詳細については、MySQL ドキュメントの「レプリケーション実装の詳細」を参照してください。

出典の DB インスタンスに対する更新とそれに続くリードレプリカに対する更新の間の遅延を低減するには、次のような方法があります。

- リードレプリカの DB インスタンスクラスが出典の DB インスタンスと同程度のストレージサイズを持つように設定します。
- 出典の DB インスタンスとリードレプリカによって使用される DB パラメータグループのパラメータ設定の互換性を保ちます。詳細と例については、次のセクションにある max_allowed_packet パラメータの説明を参照してください。
- クエリのキャッシュを無効にします。頻繁に変更されるテーブルでは、クエリのキャッシュを使用すると、キャッシュが頻繁にロックされ、更新されるため、レプリカの遅延が増加する可能性があります。このような場合、クエリのキャッシュを無効にすると、レプリカの遅延が小さくなることがあります。DB インスタンスの DB パラメータグループで query_cache_type parameter を 0 に設定することによって、クエリのキャッシュを無効にできます。クエリのキャッシュの詳細については、「クエリのキャッシュの設定」を参照してください。

- InnoDB for MySQL のリードレプリカのバッファプールをウォームします。例えば、頻繁に更新される一連の小さなテーブルがあり、InnoDB または XtraDB のテーブルスキーマを使用しているとします。その場合、それらのテーブルをリードレプリカにダンプします。そうすることで、データベースエンジンはそれらのテーブルの行をディスクからスキャンしてバッファプールにキャッシュします。これにより、レプリカの遅延を低減することができます。例を以下に示します。

  Linux、macOS、Unix の場合:

```
PROMPT> mysqldump \
    -h <endpoint> \
    --port=<port> \
    -u=<username> \
    -p <password> \
    database_name table1 table2 > /dev/null
```

  Windows の場合:

```
PROMPT> mysqldump ^
    -h <endpoint> ^
    --port=<port> ^
    -u=<username> ^
    -p <password> ^
    database_name table1 table2 > /dev/null
```

MySQL のリードレプリケーションのエラーの診断と解決

Amazon RDS では、リードレプリカのレプリケーションステータスをモニタリングします。RDS では、何らかの理由でレプリケーションが停止した場合、リードレプリカのインスタンスの [Replication State] (レプリケーションステータス) フィールドを `Error` に更新します。MySQL または MariaDB エンジンによりスローされた関連するエラーの詳細は、[] フィールドを参照することで確認できます。リードレプリカのステータスを示すイベントが生成されます (RDS-EVENT-0045、RDS-EVENT-0046、RDS-EVENT-0057 など)。イベントについてとイベントへのサブスクライブの詳細については、「Amazon RDS イベント通知の操作」を参照してください。MySQL のエラーメッセージが返された場合は、MySQL のエラーメッセージのドキュメントでエラーを確認してください。

レプリケーションエラーを引き起こす可能性がある一般的な状況は次のとおりです。

- リードレプリカの max_allowed_packet パラメータの値が出典の DB インスタンスの max_allowed_packet パラメータの値より小さい。

  max_allowed_packet パラメータは、DB パラメータグループで設定できるカスタムパラメータです。max_allowed_packet パラメータは、データベースで実行できるデータ操作言語 (DML) の最大サイズを指定するために使用されます。ソースの DB インスタンスの max_allowed_packet の値がリードレプリカの max_allowed_packet の値より大きい場合があります。その場合、レプリケーションプロセスはエラーをスローし、レプリケーションを停止する可能性があります。最も一般的なエラーは「packet bigger than 'max_allowed_packet' bytes」です。出典とリードレプリカで同じ max_allowed_packet パラメータ値を持つ DB パラメータグループが使用されるように設定することにより、エラーを修正できます。
- リードレプリカのテーブルに書き込んでいる。リードレプリカでインデックスを作成する場合、read_only パラメータを 0 に設定してインデックスを作成する必要があります。リードレプリカのテーブルに書き込んだ場合、レプリケーションが中断する可能性があります。
- MyISAM などの非トランザクションストレージエンジンを使用している。リードレプリカにはトランザクションストレージエンジンが必要です。レプリケーションは、InnoDB for MySQL または MariaDB のストレージエンジンでのみサポートされています。

  次のコマンドで MyISAM テーブルを InnoDB に変換できます。

  ```
  alter table <schema>.<table_name> engine=innodb;
  ```
- SYSDATE() など、安全でない非決定的クエリを使用している。詳細については、MySQL ドキュメントの「バイナリロギングでの安全および安全でないステートメントの判断」を参照してください。

以下のステップは、レプリケーションエラーを解決するのに役立ちます。

- 論理的なエラーが発生し、安全にエラーをスキップできる場合は、「現在のレプリケーションエラーのスキップ」のステップに従ってください。Aurora MySQL DB インスタンスは、mysql_rds_skip_repl_error プロシージャを含むバージョンを実行している必要があります。詳細については、「mysql_rds_skip_repl_error」を参照してください。
- バイナリログ (binlog) の位置の問題が発生した場合は、レプリカの再生位置を変更できます。これを行うには、Aurora MySQL バージョン 1 および 2 の mysql.rds_next_master_log コマンドを使用します。これを行うには、Aurora MySQL バージョン 3 以降の mysql.rds_next_source_log コマンドを使用します。レプリカの再生位置を変更するには、Aurora MySQL DB インスタンスがこのコマンドをサポートしているバージョンで実行されて

いる必要があります。バージョン情報については、「mysql_rds_next_master_log」を参照してください。

- DML の高負荷によって一時パフォーマンスの問題が発生した場合は、リードレプリカの DB パラメータグループで innodb_flush_log_at_trx_commit パラメータを 2 に設定できます。これを行うことによって、一時的に ACID 特性 (不可分性、整合性、分離性、耐久性の高い) が低下しますが、リードレプリカの遅延を解消するのに役立ちます。
- リードレプリカを削除し、同じ DB インスタンス識別子を使用してインスタンスを作成できます。この方法では、エンドポイントは前のリードレプリカと同じままになります。

レプリケーションエラーが解決すると、[Replication State] は [replicating] に変化します。詳細については、「MySQL リードレプリカに関する問題のトラブルシューティング」を参照してください。

レプリケーション停止エラー

mysql.rds_skip_repl_error コマンドを呼び出すと、レプリケーションがダウンまたは無効であることを示すエラーメッセージが表示されることがあります。

このエラーメッセージは、レプリケーションが停止して再開できないために表示されます。

多数のエラーをスキップする必要がある場合は、レプリケーションの遅延により、バイナリログファイルがデフォルトの保持期間を超えて増大する場合があります。この場合、バイナリログファイルがレプリカで再生される前に破棄され、致命的なエラーが発生することがあります。この破棄によりレプリケーションが停止し、mysql.rds_skip_repl_error コマンドを呼び出してレプリケーションエラーをスキップすることができなくなります。

この問題は、レプリケーション出典でバイナリログファイルの保持時間を増加させることで軽減できます。バイナリログ保持時間を長くすると、レプリケーションを再開し、必要に応じて mysql.rds_skip_repl_error コマンドを使用できるようになります。

バイナリログの保持期間を設定するには、mysql_rds_set_configuration プロシージャを使用します。設定パラメータの「binlog retention hours」と DB クラスターでバイナリログファイルを保持する時間数を最大 2,160 時間 (90 日) で指定します。Aurora MySQL のデフォルトは 24 (1 日) です。以下の例では、バイナリログファイルの保持期間を 48 時間に設定しています。

```
CALL mysql.rds_set_configuration('binlog retention hours', 48);
```

Amazon RDS API リファレンス

AWS Management Console と AWS Command Line Interface (AWS CLI) の他に、Amazon RDS には API も用意されています。API を使用して、DB インスタンスと Amazon RDS の他のオブジェクトを管理するためのタスクを自動化できます。

- API オペレーションのアルファベット順リストについては、「アクション」を参照してください。
- データ型のアルファベット順リストについては、「データ型」を参照してください。
- 共通クエリパラメータのリストについては、「共通パラメータ」を参照してください。
- エラーコードの説明については、「共通エラー」を参照してください。

AWS CLI の詳細については、「Amazon RDS の AWS Command Line Interface リファレンス」を参照してください。

トピック

- クエリ API の使用
- Aurora のアプリケーションのトラブルシューティング

クエリ API の使用

以下のセクションでは、Query API で使用されるパラメータおよびリクエスト認証について簡単に説明します。

Query API の動作に関する一般的な情報については、Amazon EC2 API Referenceの「クエリリクエスト」を参照してください。

クエリパラメータ

HTTP クエリベースのリクエストとは、HTTP 動詞 (GET または POST) とクエリパラメータ `Action` で記述する HTTP リクエストです。

各クエリリクエストに、アクションの認証と選択を処理するための一般的なパラメータがいくつか含まれている必要があります。

オペレーションの中にはパラメータのリストを取るものがあります。これらのリストは、`param.`*n* 表記を使用して指定されます。*n* 値は、1 から始まる整数です。

Amazon RDS のリージョンとエンドポイントの詳細については、Amazon Web Services 全般のリファレンス の「リージョンとエンドポイント」セクションの「Amazon Relational Database Service (RDS)」を参照してください。

クエリリクエストの認証

HTTPS 経由でのみリクエストを送信できます。また、各クエリリクエストには署名を含める必要があります。AWS 署名バージョン 4 または署名バージョン 2 のどちらかを使用してください。詳細については、「署名バージョン 4 の署名プロセス」および「署名バージョン 2 の署名プロセス」を参照してください。

Aurora のアプリケーションのトラブルシューティング

Amazon RDS では、Amazon RDS API とのやり取りで発生する問題をトラブルシューティングする際に役立つ、具体的でわかりやすいエラーを提供します。

トピック

- エラーの取得
- トラブルシューティングのヒント

Amazon RDS DB インスタンスのトラブルシューティングの詳細については、「Amazon Aurora のトラブルシューティング」を参照してください。

エラーの取得

通常、アプリケーションでは、結果を処理する前にリクエストでエラーが生成されたかどうかを必ず確認します。エラーが発生したかどうかを確認する最も簡単な方法は、Amazon RDS API からのレスポンスで `Error` ノードを検索することです。

XPath 構文を使用すると、簡単な方法で 1 つの `Error` ノードがあるかどうかを検索します。また、エラーコードとメッセージを比較的簡単に取得できます。次のコードでは、Perl および XML::XPath モジュールによって、リクエスト時のエラーの発生を判定しています。エラーが発生した場合、レスポンス内の最初のエラーコードとメッセージが表示されます。

```
use XML::XPath;
    my $xp = XML::XPath->new(xml =>$response);
    if ( $xp->find("//Error") )
    {print "There was an error processing your request:\n", " Error code: ",
```

```
    $xp->findvalue("//Error[1]/Code"), "\n", " ",
    $xp->findvalue("//Error[1]/Message"), "\n\n"; }
```

トラブルシューティングのヒント

Amazon RDS API の問題を診断して解決するには、次の手順を実行することをお勧めします。

- http://status.aws.amazon.com を確認し、対象とする AWS リージョンで Amazon RDS が正常に動作しているかどうかを確認します。
- リクエストの構文を確認します。

  『Amazon RDS API リファレンス』には、各 Amazon RDS オペレーションについてのリファレンスページがあります。パラメータを正しく使用していることをもう一度確認してください。間違っている可能性がある部分を判断するヒントとして、同様のオペレーションを実行しているサンプルのリクエストやユーザーシナリオを調べてください。
- AWS re:Post を確認する

  Amazon RDS には開発コミュニティあり、これまでに他のデベロッパーが経験した問題に対する解決策を探すことができます。トピックを表示するには、AWS re:Post に移動してください。

ドキュメント履歴

現在の API バージョン: 2014 年 10 月 31 日

以下の表は、Amazon Aurora ユーザーガイドの重要な変更点をまとめたものです。このドキュメントの更新に関する通知については、RSS フィードにサブスクライブできます。Amazon Relational Database Service (Amazon RDS) については、「Amazon Relational Database Service ユーザーガイド」を参照してください。

> **Note**
>
> 2018 年 8 月 31 日以前は、Amazon Aurora は、 Amazon Relational Database Service ユーザーガイドに記載されています。Aurora に関する以前のドキュメント履歴については、Amazon Relational Database Service ユーザーガイドの「ドキュメント履歴」を参照してください。

[What's New with Database?] (データベースの新機能) ページで新しい Amazon Aurora 機能をフィルタリングできます。[製品] で、[Amazon Aurora] を選択します。その後、**`global database`** や **`Serverless`** などのキーワードを使用して検索します。

変更	説明	日付
一般提供されている AWS Python ドライバー	Amazon Web Services (AWS) Python ドライバーは、高度な Python ラッパーとして設計されています。このラッパーは、オープンソースの Psycopg ドライバーの機能を補完し、拡張します。詳細については、「AWS ドライバーを使用した Aurora DB クラスターへの接続」を参照してください。	2024 年 5 月 23 日
中国リージョンで利用可能なゼロ ETL 統合	ゼロ ETL 統合が、AWS リージョン 中国 (北京) および中	2024 年 5 月 21 日

	国 (寧夏) で利用可能になりました。詳細については、「Amazon Redshift とのゼロ ETL 統合」を参照してください。	
RDS Proxy がさらに多くのリージョンで利用可能	RDS Proxy は、アジアパシフィック (ハイデラバード)、アジアパシフィック (メルボルン)、中東 (アラブ首長国連邦)、イスラエル (テルアビブ)、カナダ西部 (カルガリー)、欧州 (チューリッヒ) の各リージョンで利用可能になりました。RDS Proxy の詳細については、「Amazon RDS Proxy の使用」を参照してください。	2024 年 5 月 21 日
Amazon RDS 延長サポート	Aurora MySQL バージョン 2 または 3、または Aurora PostgreSQL バージョン 11 データベースを作成または復元すると、自動的にデータベースが Amazon RDS 延長サポートに登録されるようになりました。これにより、既存のアプリケーションはそのまま引き続き動作します。RDS 延長サポートをオプトアウトして、データベースエンジンの Aurora 標準サポート終了日後の料金を避けることができます。詳細については、「Amazon RDS 延長サポートの使用」を参照してください。	2024 年 3 月 21 日

ゼロ ETL 統合でのデータフィルタリング	Amazon RDS は、Amazon Redshift とのゼロ ETL 統合向けの、データベースとテーブルレベルでのデータフィルタリングをサポートしています。詳細については、「Amazon Redshift との Aurora のゼロ ETL 統合向けのデータフィルタリング」を参照してください。	2024 年 3 月 20 日
Amazon Bedrock と Aurora MySQL の統合	Amazon Aurora MySQL データベースを Amazon Bedrock と統合して、生成 AI アプリケーションを強化できるようになりました。詳細については、「Aurora MySQL での Amazon Aurora 機械学習の使用」を参照してください。	2024 年 3 月 8 日
新しい AWS マネージドポリシー	Amazon RDS は、RDS Custom が EC2 インスタンスプロファイルを介して自動化アクションとデータベース管理タスクを実行できるように `AmazonRDSCustomInstanceProfileRolePolicy` という名前の新しいマネージドポリシーを追加しました。詳細については、「Amazon RDS の AWS マネージドポリシーに関する更新」を参照してください。	2024 年 2 月 27 日

イスラエル (テルアビブ) リージョンでの Amazon RDS の AWS Secrets Manager へのサポート	Amazon RDS は、イスラエル (テルアビブ) リージョンにおいて Secrets Manager をサポートしています。詳細については、「Amazon RDS と AWS Secrets Manager によるパスワード管理」を参照してください。	2024 年 2 月 21 日
Amazon RDS 延長サポート	DB クラスターとグローバルクラスター内の Aurora MySQL と Aurora PostgreSQL メジャー エンジンバージョンが Aurora の標準サポート終了日に達すると、Amazon RDS は自動的に Amazon RDS 延長サポートを有効にするようになりました。詳細については、「Amazon RDS 延長サポートの使用」を参照してください。	2024 年 2 月 15 日
Aurora PostgreSQL 16.1 が Babelfish for Aurora PostgreSQL 4.0.0 をサポート	Aurora PostgreSQL 13.5 は Babelfish 4.0.0 をサポートしています。新機能の一覧については、「16.1」を参照してください。Babelfish の各リリースでサポートされている機能のリストについては、「バージョンごとに Babelfish でサポートされている機能」を参照してください。詳細については、「Aurora PostgreSQL 用の Babelfish の使用」を参照してください。	2024 年 1 月 31 日

デフォルトの CA 証明書の更新	デフォルトの CA 証明書は `rds-ca-rsa2048-g1` に設定されます。詳細については、「SSL/TLS を使用して DB クラスターへの接続を暗号化する」を参照してください。	2024 年 1 月 26 日
RDS Proxy が 欧州 (スペイン) リージョンで利用可能に	RDS Proxy が 欧州 (スペイン) リージョンで利用可能になりました。RDS Proxy の詳細については、「Amazon RDS Proxy の使用」を参照してください。	2024 年 1 月 8 日
Aurora PostgreSQL Serverless v2 とプロビジョニングされた RDS Data API	RDS Data API を Aurora PostgreSQL Serverless v2 およびプロビジョニングされた DB クラスターで使用できるようになりました。RDS Data API を使用すると、安全な HTTP エンドポイントを介して Aurora クラスターにアクセスし、データベースドライバーを使用したり、接続を管理したりすることなく、SQL ステートメントを実行できます。詳細については、「RDS Data API の使用 」を参照してください。	2023 年 12 月 21 日

Amazon Bedrock と Aurora PostgreSQL の統合	Amazon Aurora PostgreSQL データベースを Amazon Bedrock と統合して、生成 AI アプリケーションを強化できるようになりました。詳細については、「Aurora PostgreSQL での Amazon Aurora 機械学習の使用」を参照してください。	2023 年 12 月 21 日
Amazon Aurora がカナダ西部 (カルガリー) リージョンで使用可能に	Amazon Aurora がカナダ西部 (カルガリー) リージョンで使用可能になりました。詳細については、「リージョンとアベイラビリティーゾーン」を参照してください。	2023 年 12 月 20 日
Amazon RDS が推奨事項の表示と対応をサポート	Amazon Aurora の推奨事項には、しきい値ベースの事前対応型推奨事項と機械学習ベースの事後対応型推奨事項が含まれるようになりました。詳細については、「Amazon Aurora 推奨事項の表示と対応」を参照してください。	2023 年 12 月 19 日

Amazon Redshift (プレビュー) との Amazon Aurora ゼロ ETL 統合	Aurora PostgreSQL ソース DB クラスターを使用して、Amazon Redshift とのゼロ ETL 統合を作成できます。プレビューリリースでは、米国東部 (オハイオ) (us-east-2) AWS リージョン の Amazon RDS データベースプレビュー環境内にすべての統合を作成する必要があります。詳細については、「Amazon Redshift との Aurora ゼロ ETL 統合の操作」を参照してください。	2023 年 11 月 28 日
Amazon Aurora PostgreSQL によるグローバルデータベースの書き込み転送のサポート	Aurora PostgreSQL ベースのグローバルデータベースのセカンダリクラスターで書き込み転送を有効にできるようになりました。詳細については、「Aurora PostgreSQL グローバルデータベースでの書き込み転送の使用」を参照してください。	2023 年 11 月 9 日
Aurora PostgreSQL による Optimized Reads のサポート	Aurora Optimized Reads で Aurora PostgreSQL のクエリ処理を高速化できます。詳細については、「Aurora PostgreSQL with Aurora Optimized Reads のクエリパフォーマンスの向上」を参照してください。	2023 年 11 月 8 日

Amazon RDS による Performance Insights カウンターメトリクスの Amazon CloudWatch へのエクスポート	Performance Insights では、事前設定済みまたはカスタムメトリックスダッシュボードを Amazon CloudWatch にエクスポートできます。CloudWatch コンソールでエクスポートしたメトリクスを表示できます。選択した Performance Insights メトリクスウィジェットをエクスポートして、CloudWatch コンソールでメトリクスデータを表示することもできます。詳細については、「Performance Insights メトリクスの CloudWatch へのエクスポート」を参照してください。	2023 年 11 月 8 日
Amazon Redshift との MySQL Aurora ゼロ ETL 統合の一般的な可用性	Amazon Redshift とのゼロ ETL 統合が Aurora MySQL で一般的に利用できるようになりました。詳細については、「Amazon Redshift との Aurora ゼロ ETL 統合の操作」を参照してください。	2023 年 11 月 7 日

Aurora PostgreSQL による RDS ブルー/グリーンデプロイのサポート	Aurora PostgreSQL DB クラスターからブルー/グリーンデプロイを作成できるようになりました。詳細については、「Using Amazon RDS Blue/Green Deployments for database updates」(データベース更新のために Amazon RDS ブルー/グリーンデプロイを使用する) を参照してください。	2023 年 10 月 26 日
AWS KMS keys (SSE-KMS) による Aurora MySQL のサーバー側の暗号化のサポート	Aurora MySQL バージョン 3.05 以降では、Amazon S3 からロードまたは Amazon S3 に保存するデータのサーバー側の暗号化に、AWS マネージドキー とカスタマーマネージドキーを含む SSE-KMS を使用できます。詳細については、「Amazon S3 バケット内のテキストファイルから Amazon Aurora MySQL DB クラスターへのデータのロード」および「Amazon S3 バケット内のテキストファイルから Amazon Aurora MySQL DB クラスターへのデータの保存」を参照してください。	2023 年 10 月 25 日

Aurora MySQL の最適化によるデータベースの再起動時間の短縮	Aurora MySQL バージョン 3.05 以降では、データベースの再起動時間を短縮するための最適化が導入されました。これらの最適化により、最適化を行わない場合と比べてダウンタイムが最大 65% 短縮され、再起動後のデータベースワークロードの中断も少なくなります。詳細については、「データベースの再起動時間を短縮するための最適化」を参照してください。	2023 年 10 月 25 日
AWS マネージドポリシーの更新	`AmazonRDSPerformanceInsightsReadOnly` および `AmazonRDSPerformanceInsightsFullAccess` マネージドポリシーには、`Sid` (ステートメント ID) がポリシーステートメントの識別子として含まれます。詳細については、「Amazon RDS の AWS マネージドポリシーに関する更新」を参照してください。	2023 年 10 月 23 日

Amazon RDS が Performance Insights カウンターメトリクスを Amazon CloudWatch に発行	CloudWatch コンソールの DB_PERF_INSIGHTS メトリクス数学関数を使用すると、Amazon RDS にクエリを実行してPerformance Insights カウンターメトリクスを取得できます。詳細については、「Amazon Aurora をモニタリングするための CloudWatch アラームの作成」を参照してください。	2023 年 9 月 20 日
Amazon Aurora は、AWS Backup のポイントインタイムリカバリ (PITR) をサポート	AWS Backup で Aurora 自動 (連続) バックアップを管理して、それらから指定した時間に復元できます。詳細については、「AWS Backup を使用して DB クラスターを指定の時点の状態に復元する」を参照してください。	2023 年 9 月 7 日
Amazon RDS 延長サポート	Amazon Aurora は、Aurora の標準サポート終了日を過ぎても、DB インスタンスで Aurora MySQL と Aurora PostgreSQL のメジャーエンジンバージョンを引き続き実行できるようになることを発表しました。詳細については、「Amazon RDS 延長サポートの使用」を参照してください。	2023 年 9 月 1 日

Amazon Aurora MySQL が Percona XtraBackup のサポートを延長	MySQL 8.0 データベースを Aurora MySQL バージョン 3 DB クラスターに物理的に移行できるようになりました。詳細については、「Percona XtraBackup と Amazon S3 を使用した MySQL からの物理的な移行」を参照してください。	2023 年 8 月 24 日
Aurora グローバルデータベースはグローバルデータベースフェイルオーバーをサポート	Aurora グローバルデータベースで、マネージドグローバルレベルのフェイルオーバーがサポートされるようになりました。これにより、真の地域災害からの復旧が簡単になります。この機能の詳細については、「Aurora Global Database のマネージドフェイルオーバーを実行する」を参照してください。これまで「マネージドプランフェイルオーバー」と呼ばれていた機能は、「スイッチオーバー」と呼ばれるようになりました。スイッチオーバーについては、「Amazon Aurora Global Database スイッチオーバーの実行」を参照してください。	2023 年 8 月 21 日

AWS マネージドポリシーアクセス許可の更新	AmazonRDSFullAccess 管理ポリシーに、一定期間のパフォーマンス分析レポートの生成、表示、削除を許可する新しいアクセス許可が追加されました。詳細については、「Amazon RDS の AWS マネージドポリシーに関する更新」を参照してください。	2023 年 8 月 17 日
AWS マネージドポリシーアクセス許可の更新	AmazonRDSPerformanceInsightsReadOnly 管理ポリシーへの新しいアクセス許可の追加と、新しい管理ポリシー AmazonRDSPerformanceInsightsFullAccess の追加により、一定期間の DB 負荷分析レポートを生成できるようになりました。詳細については、「Amazon RDS の AWS マネージドポリシーに関する更新」を参照してください。	2023 年 8 月 16 日
Amazon RDS は、Performance Insights による DB 負荷期間の分析をサポート	Performance Insights では、特定の期間のパフォーマンス分析レポートを作成できます。レポートには、特定されたインサイトと、パフォーマンス問題を解決するための推奨事項が記載されています。詳細については、「一定期間の DB 負荷の分析」を参照してください。	2023 年 8 月 16 日

Amazon Aurora は DB クラスターの自動バックアップの保持をサポート	削除した Aurora クラスターの自動バックアップを保持し、指定した時点に復元できるようになりました。詳細については、「自動バックアップの保持」を参照してください。	2023 年 8 月 4 日
Amazon Aurora がイスラエル (テルアビブ) リージョンで使用可能に	Amazon Aurora がイスラエル (テルアビブ) リージョンで使用可能になりました。詳細については、「リージョンとアベイラビリティーゾーン」を参照してください。	2023 年 8 月 1 日
Amazon Aurora MySQL はローカル (クラスター内) 書き込み転送をサポート	リーダー DB インスタンスから Aurora MySQL DB クラスター内のライター DB インスタンスに書き込み操作を転送できるようになりました。詳細については、「Amazon Aurora MySQL DB クラスターでの書き込み転送の使用」を参照してください。	2023 年 7 月 31 日
Amazon Aurora は、追加の AWS リージョン で Aurora Serverless v2 をサポートします	アジアパシフィック (メルボルン) AWS リージョン で Aurora Serverless v2 DB クラスターを作成できるようになりました。Aurora Serverless v2 の詳細については、「Aurora Serverless v2 の使用」を参照してください。	2023 年 6 月 28 日

Amazon Aurora が Amazon Redshift とのゼロ ETL 統合を導入 (プレビュー)	ゼロ ETL 統合は、トランザクションデータが Aurora MySQL DB クラスターに書き込まれてから数秒以内に Amazon Redshift で使用可能にするためのフルマネージドソリューションです。詳細については、「Amazon Redshift との Aurora ゼロ ETL 統合の操作」を参照してください。	2023 年 6 月 28 日
Amazon RDS は、Performance Insights ダッシュボードに Performance Insights と CloudWatch メトリクスの統合ビューを提供しています	Amazon RDS の Performance Insights ダッシュボードで、Performance Insights と CloudWatch メトリクスの統合ビューが提供されるようになりました。詳細については、「Amazon RDS コンソールでの組み合わせたメトリクスの表示」を参照してください。	2023 年 5 月 24 日
Amazon Aurora は db.r7g インスタンスクラスをサポートします	db.r7g インスタンスクラスを使用して Aurora DB クラスターを作成できるようになりました。詳細については、「Aurora DB インスタンスクラス」を参照してください。	2023 年 5 月 11 日

Amazon Aurora は、新しい DB クラスターストレージ設定をサポートします	Aurora I/O-Optimizedでは、お支払いいただくのは DB クラスターの使用量とストレージに対してのみで、読み取りと書き込み I/O オペレーションに追加料金はかかりません。詳細については、「Amazon Aurora DB クラスターのストレージ設定」を参照してください。	2023 年 5 月 11 日
Amazon Aurora は、追加の AWS リージョン で Aurora Serverless v2 をサポートします	Aurora Serverless v2 DB クラスターは次の AWS リージョン で作成できるようになりました: アジアパシフィック (ハイデラバード)、欧州 (スペイン)、欧州 (チューリッヒ)、中東 (アラブ首長国連邦)。Aurora Serverless v2 の詳細については、「Aurora Serverless v2 の使用」を参照してください。	2023 年 5 月 4 日
Aurora Serverless v1 がプロビジョニング済みへの変換をサポート	Aurora Serverless v1 DB クラスターをプロビジョニング済み DB クラスターに直接変換できます。詳細については、「Aurora Serverless v1 DB クラスターの変更」を参照してください。	2023 年 4 月 27 日

Aurora Serverless v1 が Amazon Aurora PostgreSQL バージョン 13 をサポート	Aurora PostgreSQL バージョン 13 を実行するAurora Serverless v1 DB クラスターを作成できるようになりました。詳細については、「Aurora Serverless v1」を参照してください。	2023 年 4 月 27 日
Amazon Aurora が中国リージョンで AWS Secrets Manager をサポート	Amazon Aurora が、中国 (北京) および中国 (寧夏) リージョンで Secrets Manager をサポートします。詳細については、「Amazon RDS と AWS Secrets Manager によるパスワード管理」を参照してください。	2023 年 4 月 20 日
Amazon Aurora が、トピックサブスクライバーへのタグ付きイベントの発行をサポート	Amazon Simple Notification Service (Amazon SNS) または Amazon EventBridge に送信された Amazon Aurora イベント通知に、メッセージ本文にイベントタグが含まれるようになりました。これらのタグは、サービスイベントの影響を受けたリソースデータを提供します。詳細については、「Amazon RDS event notification tags and attributes」(Amazon Redshift イベント通知タグと属性) を参照してください。	2023 年 4 月 17 日

IAM サービスリンクロール許可に対する更新	AmazonRDSFullAccess および AmazonRDSReadOnlyAccess ポリシーは、RDS コンソールで Amazon DevOps Guru の検出結果を表示できる追加のアクセス許可を付与するようになりました。詳細については、「Amazon RDS の AWS マネージドポリシーに関する更新」を参照してください。	2023 年 3 月 30 日
Amazon Aurora が、アジアパシフィック (メルボルン) リージョンの Global Databases をサポート	アジアパシフィック (メルボルン) リージョンで Aurora Global Databasses を作成できるようになりました。Aurora Global Database については、「Amazon Aurora Global Database の使用」を参照してください。	2023 年 3 月 22 日
AWS マネージドポリシーアクセス許可の更新	AmazonRDSFullAccess および AmazonRDSReadOnlyAccess ポリシーは、Amazon CloudWatch に追加のアクセス許可を付与するようになりました。詳細については、「Amazon RDS の AWS マネージドポリシーに関する更新」を参照してください。	2023 年 3 月 16 日

RDS Proxy が中国リージョンで使用可能	RDS Proxy が中国 (北京) および中国 (寧夏) リージョンで利用可能になりました。RDS Proxy の詳細については、「Amazon RDS Proxy の使用」を参照してください。	2023 年 3 月 15 日
Amazon Aurora が 中国リージョンで Aurora Serverless v2 をサポート	Aurora Serverless v2 が中国 (北京) および中国 (寧夏) リージョンで利用可能になりました。詳細については、「Aurora Serverless v2」を参照してください。	2023 年 3 月 15 日
RDS Proxy がアジアパシフィック (ジャカルタ) リージョンで利用可能に	RDS Proxy がアジアパシフィック (ジャカルタ) リージョンで利用可能になりました。RDS Proxy の詳細については、「Amazon RDS Proxy の使用」を参照してください。	2023 年 3 月 8 日
Amazon Aurora MySQL が Kerberos 認証をサポート	Aurora MySQL DB クラスターに接続するユーザーを Kerberos 認証を使用して認証できるようになりました。詳細については、「Aurora MySQL で Kerberos 認証を使用する」を参照してください。	2023 年 3 月 8 日

Amazon Aurora が追加の AWS リージョン で Global Databases サポート	以下のリージョンで Aurora Globasl Databasses を作成できるようになりました: アフリカ (ケープタウン)、アジアパシフィック (香港)、アジアパシフィック (ハイデラバード)、アジアパシフィック (ジャカルタ)、欧州 (ミラノ)、欧州 (スペイン)、欧州 (チューリッヒ)、中東 (バーレーン)、および中東 (UAE)。Aurora Global Database については、「Amazon Aurora Global Database の使用」を参照してください。	2023 年 3 月 6 日
Amazon Aurora が追加の AWS リージョン で DB クラスタースナップショットのコピーをサポート	以下のリージョンで DB クラスタースナップショットをコピーできるようになりました: アフリカ (ケープタウン)、アジアパシフィック (香港)、アジアパシフィック (ハイデラバード)、アジアパシフィック (ジャカルタ)、アジアパシフィック (メルボルン)、欧州 (ミラノ)、欧州 (スペイン)、欧州 (チューリッヒ)、中東 (バーレーン)、および中東 (UAE) です。DB クラスターのスナップショットのコピーについては、「DB クラスターのスナップショットのコピー」を参照してください。	2023 年 3 月 6 日

Amazon DevOps Guru for RDS はプロアクティブインサイトをサポートします	Amazon DevOps Guru for RDS は、Aurora データベースの問題が発生すると予測される前に、問題の対処に役立つ推奨事項とともにプロアクティブインサイトを発行します。詳細については、「DevOps Guru for RDS の動作」を参照してください。	2023 年 2 月 28 日
Amazon Aurora MySQL バージョン 1 は廃止されました	Aurora MySQL バージョン 1 (MySQL 5.6 との互換性があります) は廃止されました。詳細については、「Amazon Aurora メジャーバージョンが利用可能な期間」を参照してください。	2023 年 2 月 28 日
Aurora Serverless v1 は、DB クラスターのメンテナンス期間の設定をサポートします	Aurora Serverless v1 DB クラスターのメンテナンス期間を設定できるようになりました。詳細については、「DB クラスターの適切なメンテナンス期間の調整」を参照してください。	2023 年 2 月 27 日
Amazon Aurora が、アジアパシフィック (ハイデラバード)、欧州 (スペイン)、中東 (アラブ首長国連邦) でデータベースアクティビティストリームをサポート	詳細については、「データベースアクティビティストリーム」を参照してください。	2023 年 1 月 27 日

Amazon Aurora がアジアパシフィック (メルボルン) リージョンで使用可能に	Amazon Aurora がアジアパシフィック (メルボルン) リージョンで使用可能になりました。詳細については、「リージョンとアベイラビリティーゾーン」を参照してください。	2023 年 1 月 23 日
DB クラスターの作成時に認証局 (CA) を指定する	DB クラスターの作成時に、DB クラスターのサーバー証明書に使用する CA を指定できるようになりました。詳細については「Certificate authorities」(認証局) を参照してください。	2023 年 1 月 5 日
Aurora MySQL 3.* でバックトラッキングをサポート	Aurora MySQL 3.* バージョンでは、テーブルや行の誤った削除などのユーザーエラーから迅速に回復できるようになりました。バックトラックを使用すると、バックアップから復元せずに、データベースを過去の時点に戻すことができます。この処理は、大規模なデータベースであっても数秒で完了します。詳細については、「Aurora DB クラスターのバックトラック」を参照してください。	2023 年 1 月 4 日

Amazon RDS ブルー/グリーンデプロイが追加の AWS リージョン で利用可能に	ブルー/グリーンデプロイ機能が、中国 (北京) および中国 (寧夏) リージョンで利用可能になりました。詳細については、「Using Amazon RDS Blue/Green Deployments for database updates」(データベース更新のために Amazon RDS ブルー/グリーンデプロイを使用する) を参照してください。	2022 年 12 月 22 日
IAM サービスリンクロール許可に対する更新	AmazonRDSServiceRolePolicy ポリシーによって、AWS Secrets Manager に追加のアクセス許可を付与するようになりました。詳細については、「Amazon RDS の AWS マネージドポリシーに関する更新」を参照してください。	2022 年 12 月 22 日
パスワード管理用に Amazon Aurora と AWS Secrets Manager が統合	Aurora は DB クラスターのマスターユーザーパスワードをシークレットマネージャーで管理できます。詳細については、「Amazon RDS と AWS Secrets Manager によるパスワード管理」を参照してください。	2022 年 12 月 22 日

Amazon Aurora は、追加の AWS リージョン で Aurora Serverless v2 をサポートします	Aurora Serverless v2 がアフリカ (ケープタウン) および EU (ミラノ) リージョンで利用できるようになりました。詳細については、「Aurora Serverless v2」を参照してください。	2022 年 12 月 21 日
Aurora PostgreSQL が PostgreSQL 14 で RDS Proxy をサポート	Aurora PostgreSQL 14 DB クラスターで RDS Proxy を作成できるようになりました。RDS Proxy の詳細については、「Amazon RDS Proxy の使用」を参照してください。	2022 年 12 月 13 日
Amazon Aurora は、Amazon DevOps Guru によって検出された最近の異常についてアラートを送信します	コンソールのデータベース詳細ページには、現在の異常と過去 24 時間に発生した異常の両方が通知されます。詳細については、「DevOps Guru for RDS の動作」を参照してください。	2022 年 12 月 13 日
Amazon RDS Proxy はグローバルデータベースをサポートします	Aurorora グローバルデータベースで RDS Proxy を使用できるようになりました。詳細については、「Using RDS Proxy with Aurora global databases」(Aurorora グローバルデータベースで RDS Proxy を使用する) を参照してください。	2022 年 12 月 7 日

Aurora PostgreSQL DB クラスターは Trusted Language Extensions for PostgreSQL をサポートしています	Trusted Language Extensions for PostgreSQL は、高性能の PostgreSQL 拡張機能を構築して、Aurora PostgreSQL DB クラスターで安全に実行できるようにするオープンソースの開発キットです。詳細については「Working with Trusted Language Extensions for PostgreSQL」(Trusted Language Extensions for PostgreSQL の使用) を参照してください。	2022 年 11 月 30 日
Amazon GuardDuty RDS Protection はアクセス脅威をモニタリングします	GuardDuty RDS Protection を有効にすると、GuardDuty は Aurora データベースから RDS ログインイベントを使用し、これらのイベントをモニタリングして、潜在的な内部脅威や外部の攻撃者がないかプロファイリングします。GuardDuty RDS Protection が潜在的な脅威を検出すると、GuardDuty は侵害の可能性があるデータベースに関する詳細を含む新しい結果を生成します。詳細については、「Monitoring threats with GuardDuty RDS Protection」(GuardDuty RDS Protection による脅威のモニタリング) を参照してください。	2022 年 11 月 30 日

データベース更新のための Amazon RDS ブルー/グリーンデプロイの使用	ステージング環境で DB クラスターを変更し、本稼働環境の DB クラスターに影響を与えずに変更をテストできます。準備ができたら、ダウンタイムを最小限に抑えながら、ステージング環境を新しい本稼働環境に昇格できます。詳細については、「Using Amazon RDS Blue/Green Deployments for database updates」(データベース更新のために Amazon RDS ブルー/グリーンデプロイを使用する) を参照してください。	2022 年 11 月 27 日
Amazon Aurora がアジアパシフィック (ハイデラバード) リージョンで使用可能に	Amazon Aurora がアジアパシフィック (ハイデラバード) リージョンで使用可能になりました。詳細については、「リージョンとアベイラビリティーゾーン」を参照してください。	2022 年 11 月 22 日
Amazon Aurora が欧州 (スペイン) リージョンで利用可能に	Amazon Aurora が欧州 (スペイン) リージョンで使用可能になりました。詳細については、「リージョンとアベイラビリティーゾーン」を参照してください。	2022 年 11 月 16 日

Amazon Aurora が欧州 (チューリッヒ) リージョンで使用可能に	Amazon Aurora が欧州 (チューリッヒ) リージョンで使用可能になりました。詳細については、「リージョンとアベイラビリティーゾーン」を参照してください。	2022 年 11 月 9 日
Amazon Aurora は、DB クラスターから Amazon S3 へのデータのエクスポートをサポートしています	最初にスナップショットを作成せずに、Aurora クラスターデータを直接 S3 にエクスポートできるようになりました。詳細については、「xporting DB cluster data to Amazon S3」(Amazon S3 への DB クラスターデータのエクスポート) を参照してください。	2022 年 10 月 27 日
Amazon Aurora MySQL は Amazon S3 へのより高速なエクスポートをサポートしています	MySQL 5.7 および 8.0 互換の Aurora MySQL クラスターで DB クラスタースナップショットデータを S3 にエクスポートする場合のパフォーマンスの速度が最大 10 倍向上しました。詳細については、「Amazon S3 への DB クラスタースナップショットデータのエクスポート」を参照してください。	2022 年 10 月 20 日

Amazon Aurora は、Aurora DB クラスターと EC2 インスタンスとの接続を自動的にセットアップすることをサポートしています	AWS Management Console を使用して、既存の Aurora DB クラスターと EC2 インスタンス間の接続をセットアップできます。詳細については、「EC2 インスタンスと Aurora DB クラスターへの自動接続」を参照してください。	2022 年 10 月 14 日
PostgreSQL 用 AWS JDBC ドライバーが一般利用可能に	PostgreSQL 用 AWS JDBC ドライバーは、Aurora PostgreSQL 用に設計されたクライアントドライバーです。PostgreSQL 用 AWS JDBC ドライバーの一般利用が可能になりました。詳細については、「PostgreSQL 用 AWS JDBC ドライバーを使用した接続」を参照してください。	2022 年 10 月 6 日
Amazon Aurora が MySQL 5.7 互換の Aurora MySQL のインプレースアップグレードをサポート	インプレースアップグレードを実行して、既存の MySQL 5.7 互換の Aurora MySQL クラスターを MySQL 8.0 互換の Aurora MySQL クラスターに変更できます。詳細については、「Aurora MySQL 2.x から 3.x にアップグレードする」を参照してください。	2022 年 9 月 26 日
パフォーマンスインサイトは上位 25 行の SQL クエリを表示します	[Top SQL] (上位の SQL) タブは DB の負荷に最も影響している 25 行の SQL クエリを表示します。詳細については、「上位の SQL タブの概要」を参照してください。	2022 年 9 月 13 日

Aurora MySQL が新しい DB インスタンスクラスをサポート	Aurora MySQL DB クラスターに db.r6i DB インスタンスクラスを使用できるようになりました。詳細については、「DB インスタンスクラス」を参照してください。	2022 年 9 月 13 日
Amazon Aurora が中東 (UAE) リージョン で使用可能に	Amazon Aurora が中東 (UAE) リージョン で使用可能になりました。詳細については、「リージョンとアベイラビリティーゾーン」を参照してください。	2022 年 8 月 30 日
Amazon Aurora は、EC2 インスタンスとの接続を自動的にセットアップすることをサポートしています。	Aurora DB クラスターを作成する場合は、AWS Management Console を使用して Amazon Amazon Elastic Compute Cloud インスタンスと新しい DB クラスター間の接続をセットアップできます。詳細については、「EC2 インスタンスとの自動ネットワーク接続を設定する」を参照してください。	2022 年 8 月 22 日
Amazon Aurora はデュアルスタックモードをサポート	DB クラスターが、デュアルスタックモードで実行できるようになりました。デュアルスタックモードでは、リソースは IPv4、IPv6、またはその両方で DB クラスターと通信できます。詳細については、「Amazon Aurora の IP アドレス指定」を参照してください。	2022 年 8 月 17 日

Amazon Aurora が PostgreSQL 互換 Aurora Serverless v1 のインプレースアップグレードをサポート	PostgreSQL 10 互換 Aurora Serverless v1 クラスターのインプレースアップグレードを実行して、既存のクラスターを PostgreSQL 11 互換 Aurora Serverless v1 クラスターに変更できます。インプレースアップグレードの手順については、「Aurora Serverless v1 DB クラスターの変更」を参照してください。	2022 年 8 月 8 日
Performance Insights がアジアパシフィック (ジャカルタ) リージョンをサポート	以前は、アジアパシフィック (ジャカルタ) リージョンでは Performance Insights を使用できませんでした。この制限は解除されました。詳細については、「AWS リージョン による Performance Insights のサポート」を参照してください。	2022 年 7 月 21 日
Amazon Aurora が新しい DB インスタンスクラスをサポート	Aurora PostgreSQL DB クラスターに db.r6i DB インスタンスクラスを使用できるようになりました。詳細については、「DB インスタンスクラス」を参照してください。	2022 年 7 月 14 日

RDS Performance Insights が追加の保持期間をサポート	以前は、Performance Insights の保持期間は 7 日 (デフォルト) と 2 年 (731 日) の 2 つのみでした。これで、パフォーマンスデータを 7 日以上保持する必要がある場合は、1 ~ 24 か月を指定できます。詳細については、「Performance Insights の料金とデータ保持」を参照してください。	2022 年 7 月 1 日
Amazon Aurora が MySQL 互換 Aurora Serverless v1 のインプレースアップグレードをサポート	MySQL 5.6 互換の Aurora Serverless v1 クラスターのインプレースアップグレードを実行して、既存のクラスターを MySQL 5.7 互換の Aurora Serverless v1 クラスターに変更できます。インプレースアップグレードの手順については、「Aurora Serverless v1 DB クラスターの変更」を参照してください。	2022 年 6 月 16 日
Aurora は RDS コンソールで Amazon DevOps Guru をオンにすることをサポート	RDS コンソールから Aurora データベースの DevOps Guru カバレッジを有効にできます。詳細については、「RDS 用の DevOps Guru のセットアップ」を参照してください。	2022 年 6 月 9 日

Amazon Aurora が、暗号化された Amazon SNS トピックへのイベントの発行をサポート	Amazon Aurora では、サーバー側の暗号化 (SSE) が有効になっている Amazon Simple Notification Service (Amazon SNS) トピックにイベントを発行できるようになりました。これにより、機密データを伝送するイベントの保護を強化できます。詳細については、「Amazon Redshift イベント通知にサブスクライブする」を参照してください。	2022 年 6 月 1 日
Amazon RDS は使用状況メトリクスを Amazon CloudWatch に公開します	Amazon CloudWatch の `AWS/Usage` 名前空間には、Amazon RDS サービスクォータのアカウントレベルの使用状況メトリクスが含まれています。詳細については、「Amazon Aurora の Amazon CloudWatch 使用状況メトリクス」を参照してください。	2022 年 4 月 28 日
JSON 形式のデータ API 結果セット	`ExecuteStatement` 関数のオプションのパラメータを使用すると、クエリの結果セットが JSON 形式の文字列として返されます。JSON 結果セットは、アプリケーションの言語でデータ構造に変換するのに簡単で便利です。詳細については、「JSON 形式でクエリ結果を処理する」を参照してください。	2022 年 4 月 27 日

Amazon Aurora Serverless v2 の一般提供を開始	Amazon Aurora Serverless v2 は、一般的にはすべてのユーザーが利用できます。詳細については、「Aurora Serverless v2 の使用」を参照してください。	2022 年 4 月 21 日
Aurora MySQL は設定可能な暗号スイートをサポート	Aurora MySQL では、設定可能な暗号スイートを使用して、データベースサーバーが受け入れる接続暗号化を制御できるようになりました。詳細については、「Aurora MySQL DB クラスターへの接続用暗号スイートを設定する」を参照してください。	2022 年 4 月 15 日
Aurora PostgreSQL は PostgreSQL 13 で RDS Proxy をサポート	Aurora PostgreSQL 13 DB クラスターで RDS Proxy を作成できるようになりました。RDS Proxy の詳細については、「Amazon RDS Proxy の使用」を参照してください。	2022 年 4 月 4 日
Aurora PostgreSQL リリースノート	現在、Amazon Aurora PostgreSQL リリースノートには、別のガイドがあります。詳細については、「Aurora PostgreSQL のリリースノート」を参照してください。	2022 年 3 月 22 日

Aurora MySQL のリリースノート	現在、Amazon Aurora MySQL リリースノートには、別のガイドがあります。詳細については、「Aurora MySQL のリリースノート」を参照してください。	2022 年 3 月 22 日
Aurora PostgreSQL でマルチメジャーバージョンアップグレードをサポート	Aurora PostgreSQL DB クラスターのバージョンアップグレードを複数のメジャーバージョンで実行できるようになりました。詳細については、「メジャーバージョンのアップグレードを実施する方法」を参照してください。	2022 年 3 月 4 日
Aurora PostgreSQL で設定可能な暗号スイートをサポート	Aurora PostgreSQL バージョン 11.8 以降では、設定可能な暗号スイートを使用して、データベースサーバーで使用できる接続の暗号化を管理できるようになりました。Aurora PostgreSQL で設定可能な暗号スイートを使用する方法については、「Aurora PostgreSQL DB クラスターへの接続用暗号スイートを設定する」を参照してください。	2022 年 3 月 4 日

MySQL 用 AWS JDBC ドライバーが一般利用可能に	MySQL 用 AWS JDBC ドライバーは、Aurora MySQL の高可用性を実現するために設計されたクライアントドライバーです。MySQL 用 AWS JDBC ドライバーの一般利用が可能になりました。詳細については、「Connecting with the Amazon Web Services JDBC Driver for MySQL」(MySQL 用アマゾン ウェブサービス JDBC ドライバーとの接続) を参照してください。	2022 年 3 月 2 日
Aurora PostgreSQL 13.5 が Babelfish for Aurora PostgreSQL 1.1.0 をサポート	Aurora PostgreSQL 13.5 は Babelfish 1.1.0 をサポートします。新機能の一覧については、「13.5」を参照してください。Babelfish の各リリースでサポートされている機能のリストについては、「バージョンごとに Babelfish でサポートされている機能」を参照してください。詳細については、「Aurora PostgreSQL 用の Babelfish の使用」を参照してください。	2022 年 2 月 28 日
Amazon Aurora が、アジアパシフィック (ジャカルタ) リージョンで Database Activity Streams をサポート	詳細については、「データベースアクティビティストリーミングのための AWS リージョン のサポート」を参照してください。	2022 年 2 月 16 日

Performance Insights は新しい API オペレーションをサポート	Performance Insights は、`GetResourceMetadata` 、`ListAvailableResourceDimensions` 、および `ListAvailableResourceMetrics` の API オペレーションをサポートするようになりました。詳細については、本マニュアルならびに「Amazon RDS Performance Insights API Reference」(Amazon RDS Performance Insights API リファレンス) の、「Retrieving metrics with the Performance Insights API」(Performance Insights API を使用したメトリクスの取得) を参照してください。	2022 年 1 月 12 日
Amazon RDS Proxy はイベントをサポート	RDS Proxy でイベントを生成し、それにサブスクライブして CloudWatch イベント内に表示したり、設定することで Amazon EventBridge への送信ができるようになりました。詳細については、「Working with RDS Proxy events」(RDS Proxy イベントの使用) を参照してください。	2022 年 1 月 11 日

RDS Proxy が追加の AWS リージョン で使用可能に	RDS Proxy が、アフリカ (ケープタウン)、アジアパシフィック (香港)、アジアパシフィック (大阪)、欧州 (ミラノ)、欧州 (パリ)、欧州 (ストックホルム)、中東 (バーレーン)、南米 (サンパウロ) の各リージョンで利用可能になりました。RDS Proxy の詳細については、「Amazon RDS Proxy の使用」を参照してください。	2022 年 1 月 5 日
Amazon Aurora がアジアパシフィック (ジャカルタ) リージョンで使用可能に	Amazon Aurora がアジアパシフィック (ジャカルタ) リージョンで使用可能になりました。詳細については、「リージョンとアベイラビリティーゾーン」を参照してください。	2021 年 12 月 13 日

Amazon RDS の DevOps Guru は、詳細なインサイトとレコメンデーションをAmazon Aurora に提供します	RDS の DevOps Guru は、パフォーマンス関連データの Performance Insights を掘り出します。このデータを使用して、サービスは Amazon Aurora DB インスタンスのパフォーマンスを分析し、パフォーマンスの問題解決を手助けします。詳細については、このガイドの RDS 用の DevOps Guru でパフォーマンスの異常を分析するおよび Amazon DevOps Guru ユーザーガイドの「RDS 用の DevOps Guru の概要」を参照してください。	2021 年 12 月 1 日
Aurora PostgreSQL が PostgreSQL 12 で RDS Proxy をサポート	Aurora PostgreSQL 12 データベース クラスターを使用して RDS Proxy を作成できるようになりました。RDS Proxy の詳細については、「Amazon RDS Proxy の使用」を参照してください。	2021 年 11 月 22 日
Aurora はデータベースアクティビティストリーミング に AWS Graviton2 インスタンスクラスをサポートしています	データベースアクティビティストリーミングは、Aurora MySQL および Aurora PostgreSQL の db.r6g インスタンスクラスで使用できます。詳細については、「サポートされている DB インスタンスクラス」を参照してください。	2021 年 11 月 3 日

クロスアカウント AWS KMS keys に対する Amazon Aurora のサポート	Amazon S3 に DB スナップショットをエクスポートする際、暗号化に別の AWS アカウント の KMS キーを使用することができます。詳細については、「Amazon S3 への DB スナップショットデータのエクスポート」を参照してください。	2021 年 11 月 3 日
Amazon Aurora は Aurora PostgreSQL での Babelfish をサポートしています	Aurora PostgreSQL の Bablefish は、Microsoft SQL Server クライアントからのデータベース接続を受け入れる機能を使用して Amazon Aurora PostgreSQL 互換エディションデータベースを拡張します。詳細については、「Aurora PostgreSQL での Babelfish の使用」を参照してください。	2021 年 10 月 28 日
Aurora Serverless v1 では接続に SSL が必要になる場合があります	PostgreSQL の Aurora クラスターパラメータ `force_ssl` および MySQL の `require_secure_transport` は、Aurora Serverless v1 で現在サポートされています。詳細については、「Aurora Serverless v1 での TLS/SSL の使用」を参照してください。	2021 年 10 月 26 日

Amazon Aurora は追加の AWS リージョン で Performance Insights をサポート	Performance Insights は中東 (バーレーン)、アフリカ (ケープタウン)、欧州 (ミラノ)、およびアジアパシフィック (大阪) のリージョンで利用できます。詳細については、「AWS リージョン による Performance Insights のサポート」を参照してください。	2021 年 10 月 5 日
Aurora Serverless v1 の設定可能なオートスケーリングのタイムアウト	オートスケーリングポイントが見つかるまで Aurora Serverless v1 が待機する時間を選択できます。その期間中にオートスケーリングポイントが見つからなかった場合、Aurora Serverless v1 は、選択したタイムアウトアクションに従って、スケーリングイベントをキャンセルするか、容量の変更を強制します。詳細については、「Aurora Serverless v1 のオートスケーリング」を参照してください。	2021 年 9 月 10 日

Aurora が X2g および T4g インスタンスクラスをサポート	Aurora MySQL と Aurora PostgreSQL の両方で X2g と T4g インスタンスクラスを使用できるようになりました。使用できるインスタンスクラスは、Aurora MySQL や Aurora PostgreSQL のバージョンによって異なります。サポートされているインスタンスタイプの詳細については、「DB インスタンスクラス」を参照してください。	2021 年 9 月 10 日
Amazon RDS が共有 VPC で RDS Proxy をサポート	共有仮想プライベートクラウド (VPC) に RDS Proxy を作成できるようになりました。RDS Proxy の詳細については、Amazon RDS ユーザーガイドの「Amazon RDS Proxy による接続の管理」、または Aurora ユーザーガイドを参照してください。	2021 年 8 月 6 日
Aurora バージョンポリシーページ	Amazon Aurora ユーザーガイドに、Aurora のバージョンと関連ポリシーに関する一般的な情報を含むセクションが追加されました。詳細については、「Amazon Aurora バージョン」を参照してください。	2021 年 7 月 14 日

AWS CloudTrail 証跡からデータ API イベントを除外	CloudTrail 証跡からデータ API イベントを除外できます。詳細については、「データ API イベントを AWS CloudTrail 証跡から除外する」を参照してください。	2021 年 7 月 2 日
Amazon Aurora PostgreSQL 互換エディションが、追加の拡張機能をサポート	新たにサポートされる拡張には、pg_bigm、pg_cron、pg_partman、pg_proctab などがあります。詳細については、「Amazon Aurora PostgreSQL 互換エディションの拡張バージョン」を参照してください。	2021 年 6 月 17 日
Aurora Serverless クラスターのクローン作成	Aurora Serverless のクラスターのクローンを作成できるようになりました。クローン作成については、「Aurora DB クラスターのボリュームのクローン作成」を参照してください。	2021 年 6 月 16 日
中国 (北京) および 中国 (寧夏) リージョンで利用可能な Aurora Global Database	中国 (北京) および 中国 (寧夏) リージョンで Aurora Global Database を作成できるようになりました。Aurora Global Database については、「Amazon Aurora Global Database の使用」を参照してください。	2021 年 5 月 19 日

FIPS 140-2 でのデータ API のサポート	データ API は、SSL/TLS 接続に対し、Federal Information Processing Standard Publication 140-2 (FIPS 140-2) をサポートしています。詳細については、「データ API の可用性」を参照してください。	2021 年 5 月 14 日
PostgreSQL 用 AWS JDBC ドライバー (プレビュー)	PostgreSQL 用 AWS JDBC ドライバー (プレビューで利用可能) は、Aurora PostgreSQL の高可用性を実現するために設計されたクライアントドライバーです。詳細については、PostgreSQL 用 Amazon Web Services JDBC ドライバー を使用した接続 (プレビュー) を参照してください。	2021 年 4 月 27 日
Data API が追加の AWS リージョン で使用可能に	データ API が、アジアパシフィック (ソウル) リージョンと カナダ (中部) リージョンでご利用いただけるようになりました。詳細については、「Data API の提供状況」を参照してください。	2021 年 4 月 9 日
Amazon Aurora は Graviton2 DB インスタンスクラスをサポート	Graviton2 DB インスタンスクラス db.r6g.x を使用して、MySQL または PostgreSQL を実行する DB クラスターを作成できるようになりました。詳細については、「DB インスタンスクラスクラスのタイプ」を参照してください。	2021 年 3 月 12 日

RDS Proxy エンドポイントの拡張機能	各 RDS Proxy に関連付けられた追加のエンドポイントを作成できます。別の VPC にエンドポイントを作成すると、プロキシの VPC 間アクセスが有効になります。Aurora MySQL クラスターのプロキシは、読み取り専用エンドポイントを持つこともできます。これらのリーダーエンドポイントは、クラスター内のリーダー DB インスタンスに接続し、クエリを多用するアプリケーションの読み取りスケーラビリティと可用性を向上させることができます。RDS Proxy の詳細については、Amazon RDS ユーザーガイドの「Amazon RDS Proxy による接続の管理」、または Aurora ユーザーガイドを参照してください。	2021 年 3 月 8 日
Amazon Aurora がアジアパシフィック (大阪) リージョンで使用可能に	Amazon Aurora がアジアパシフィック (大阪) リージョンで使用可能になりました。詳細については、「リージョンとアベイラビリティーゾーン」を参照してください。	2021 年 3 月 1 日

Aurora PostgreSQL は、同じ DB クラスターで IAM 認証と Kerberos 認証の両方をサポート	Aurora PostgreSQL で、同じ DB クラスターで IAM 認証と Kerberos 認証の両方を有効にできるようになりました。詳細については、「Amazon Aurora でのデータベース認証」を参照してください。	2021 年 2 月 24 日
Aurora Global Database で計画したフェイルオーバーの管理をサポートスタート	Aurora Global Database で、管理された計画済みのフェイルオーバーがサポートされるようになりました。これにより、Aurora Global Database のプライマリ AWS リージョンをより簡単に変更できます。管理された計画済みのフェイルオーバーは、正常な Aurora Global Database でのみ使用できます。詳細については、「災害対策および Amazon Aurora グローバルデータベース」を参照してください。参照情報については、Amazon RDS API リファレンス の「`FailoverGlobalCluster`」を参照してください。	2021 年 2 月 11 日

Aurora Serverless のデータ API が、より多くのデータ型のサポートをスタート	Aurora Serverless の Data API を使用して、データベースへの入力として UUID および JSON データ型を使用できるようになりました。また、Aurora Serverless の Data API を使用して、データベースから LONG タイプの値を STRING 値として返すことができるようになりました。詳細については、「Data API の呼び出し」を参照してください。サポートされているデータタイプの参照情報については、「Amazon RDS データサービス API リファレンス」の SqlParameter を参照してください。	2021 年 2 月 2 日
Aurora PostgreSQL は、PostgreSQL 12 へのメジャーバージョンアップグレードをサポート	Aurora PostgreSQL では、DB エンジンをメジャーバージョン 12 にアップグレードできるようになりました。詳細については、「Aurora PostgreSQL の PostgreSQL DB エンジンのアップグレード」を参照してください。	2021 年 1 月 28 日

Aurora MySQL はインプレースアップグレードをサポート	Aurora MySQL 1.x クラスターを Aurora MySQL 2.x にアップグレードすると、元のクラスターの DB インスタンスやエンドポイントなどを保持できます。このインプレースアップグレード手法により、スナップショットを復元して新しいクラスター全体を設定する際の不便さを回避できます。また、すべてのテーブルのデータを新しいクラスターにコピーするオーバーヘッドも回避できます。詳細については、「Aurora MySQL DB クラスターのメジャーバージョンを 1.x から 2.x にアップグレードする」を参照してください。	2021 年 1 月 11 日
MySQL 用の AWS JDBC ドライバー (プレビュー)	MySQL 用 AWS JDBC ドライバー (プレビューで利用可能) は、Aurora MySQL の高可用性を実現するために設計されたクライアントドライバーです。詳細については、「Connecting with the Amazon Web Services JDBC Driver for MySQL」(MySQL 用アマゾンウェブ サービス JDBC ドライバーとの接続) を参照してください。	2021 年 1 月 7 日

Aurora が、グローバルデータベースのセカンダリクラスターでのデータベースアクティビティストリーミングをサポート	データベースアクティビティストリーミングは、Aurora PostgreSQL または Aurora MySQL のプライマリクラスターまたはセカンダリクラスターでスタートできます。サポートされるエンジンのバージョンについては、「Aurora Global Database の制約事項」を参照してください。	2020 年 12 月 22 日
4 つの DB インスタンスを使用するマルチマスタークラスター	Aurora MySQL マルチマスタークラスター内の DB インスタンスの最大数が 4 つになりました。以前は、最大でも 2 つの DB インスタンスでした。詳細については、「Aurora マルチマスタークラスターの使用」を参照してください。	2020 年 12 月 17 日
Aurora PostgreSQL は AWS Lambda 関数をサポート	Aurora PostgreSQL DB クラスターの AWS Lambda 関数を呼び出すことができるようになりました。詳細については、「Aurora PostgreSQL の DB クラスターから Lambda 関数を呼び出す」を参照してください。	2020 年 12 月 11 日

Amazon Aurora はプレビューで Graviton2 DB インスタンスクラスをサポート	プレビューで Graviton2 DB インスタンスクラス db.r6g.x を使用して、MySQL または PostgreSQL を実行する DB クラスターを作成できるようになりました。詳細については、「DB インスタンスクラスクラスのタイプ」を参照してください。	2020 年 12 月 11 日
Amazon Aurora Serverless v2 はプレビューで使用できるようになりました。	Amazon Aurora Serverless v2 はプレビューで利用可能です。Amazon Aurora Serverless v2 を使用するには、アクセスの申し込みが必要です。詳細については、Aurora Serverless v2 のページを参照してください。	2020 年 12 月 1 日

Aurora PostgreSQL がより多くの AWS リージョン で Aurora Serverless に使用可能になりました。	Aurora PostgreSQL がより多くの Aurora Serverless で AWS リージョン に使用可能になりました。Aurora PostgreSQL Serverless v1 は、Aurora MySQL Serverless v1 を提供するのと同じ AWS リージョン での実行を選択できるようになりました。Aurora Serverless をサポートする追加の AWS リージョン には、米国西部 (北カリフォルニア)、アジアパシフィック (シンガポール)、アジアパシフィック (シドニー)、アジアパシフィック (ソウル)、アジアパシフィック (ムンバイ)、カナダ (中部)、欧州 (ロンドン)、および欧州 (パリ) が含まれます。Aurora Serverless でサポートされているすべてのリージョンと Aurora DB エンジンのリストについては、「Aurora Serverless v1」を参照してください。Aurora Serverless 向けの Amazon RDS データ API は、これらの同じ AWS リージョン でも使用可能になりました。Aurora Serverless のデータ API をサポートするすべてのリージョンのリストについては、「Aurora MySQL Serverless v1 で Data API を使用する」を参照してください。	2020 年 11 月 24 日

Amazon RDS Performance Insights が新しいディメンションを導入	データベース、アプリケーション (PostgreSQL) およびセッションタイプ (PostgreSQL) のディメンショングループに従って、データベースのロードをグループ化できます。Amazon RDS は、db.name、db.application.name (PostgreSQL)、および db.session_type.name (PostgreSQL) のディメンションもサポートしています。詳細については、「トップロードテーブル」を参照してください。	2020 年 11 月 24 日
Aurora Serverless が Aurora PostgreSQL バージョン 10.12 をサポート	Aurora Serverless 用 Aurora PostgreSQL がサポートされているすべての AWS リージョンで、Aurora Serverless 用 Aurora PostgreSQL は Aurora PostgreSQL バージョン 10.12 にアップグレードされました。詳細については、「Aurora Serverless v1」を参照してください。	2020 年 11 月 4 日

Data API がタグベースの認証をサポートするようになりました	Data API では、タグベースの認証がサポートされます。RDS クラスターリソースにタグでラベル付けした場合、ポリシーステートメントでそれらのタグを使用して、Data API によるアクセスを制御できます。詳細については、「Data API へのアクセスを承認」を参照してください。	2020 年 10 月 27 日
Amazon Aurora が Amazon S3 へのスナップショットのエクスポートのサポートを拡張	すべての商用 AWS リージョン で DB スナップショットデータを Amazon S3 にエクスポートできるようになりました。詳細については、「Amazon S3 への DB スナップショットデータのエクスポート」を参照してください。	2020 年 10 月 22 日

Aurora Global Database がクローンの作成をサポート	Aurora Global Database のプライマリ DB クラスターとセカンダリ DB クラスターのクローンを作成できるようになりました。これを行うには、AWS Management Console で [クローンの作成] メニューオプションを選択します。また、AWS CLI で `restore-db-cluster-to-point-in-time` オプションを指定して `--restore-type copy-on-write` コマンドを実行することもできます。AWS Management Console または AWS CLI を使用して、複数の AWS アカウントをわたって Aurora Global Database から DB クラスターのクローンを作成することもできます。クローン作成の詳細については、「Aurora DB クラスターボリュームのクローン作成」を参照してください。	2020 年 10 月 19 日

Amazon Aurora は、クラスターボリュームの動的なサイズ変更をサポート	Aurora MySQL 1.23/2.09、Aurora PostgreSQL 3.3.0、Aurora PostgreSQL 2.6.0 以降では、DROP TABLE などのオペレーションでデータを削除すると、Aurora によってクラスターボリュームのサイズが縮小されます。この拡張機能を利用するには、クラスターで使用しているデータベースエンジンに応じて、いずれかの適切なバージョンにアップグレードします。この機能と、Aurora クラスターの使用済みストレージ領域および使用可能なストレージ領域を確認する方法については、「Aurora DB クラスターのパフォーマンスとスケーリングの管理」を参照してください。	2020 年 10 月 13 日
Amazon Aurora は最大 128 TiB のボリュームサイズをサポート	新しい Aurora クラスターボリュームと既存の Aurora クラスターボリュームの最大サイズを 128 tebibytes (TiB) に拡張できるようになりました。詳細については、「Aurora ストレージの拡張方法」を参照してください。	2020 年 9 月 22 日

Aurora PostgreSQL は、中国 (寧夏) リージョンで db.r5 および db.t3 DB インスタンスクラスをサポート	db.r5 および db.t3 DB インスタンスクラスを使用する Aurora PostgreSQL DB クラスターを 中国 (寧夏) リージョンで作成できるようになりました。詳細については、「DB インスタンスクラス」を参照してください。	2020 年 9 月 3 日

Aurora パラレルクエリの強化	Aurora MySQL 2.09 および 1.23 以降では、パラレルクエリ機能の拡張機能を利用できます。パラレルクエリクラスターを作成する際に、特別なエンジンモードが不要になりました。互換性のある Aurora MySQL バージョンを実行しているプロビジョニング済みクラスターの `aurora_parallel_query` 構成オプションを使用して、パラレルクエリのオンとオフを切り替えることができるようになりました。新しいクラスターを作成してデータをインポートする代わりに、既存のクラスターを互換性のある Aurora MySQL バージョンにアップグレードし、パラレルクエリを使用できます。Performance Insights は、パラレルクエリクラスターに使用できます。パラレルクエリクラスターを停止およびスタートできます。MySQL 5.7 と互換性のある Aurora パラレルクエリクラスターを作成できます。パラレルクエリは、`DYNAMIC` 行形式を使用するテーブルに対して機能します。パラレルクエリクラスターは AWS Identity and Access Management (IAM) 認証を使用できます。パラレルクエリクラスター内の リーダー DB インスタ	2020 年 9 月 2 日

	ンスは、READ COMMITTED 分離レベルを利用することができます。また、追加の AWS リージョン でパラレルクエリクラスターを作成できるようになりました。パラレルクエリ機能およびこれらの機能強化の詳細については、「Aurora MySQL のパラレルクエリを使用する」を参照してください。	
Aurora MySQL パラメータ binlog_rows_query_log_events の変更	Aurora MySQL 設定パラメータ binlog_rows_query_log_events の値を変更できるようになりました。これまで、このパラメータは変更できませんでした。	2020 年 8 月 26 日

Aurora MySQL のマイナーバージョン自動アップグレードのサポート	Aurora MySQL で Aurora MySQL DB クラスターに対して [マイナーバージョン自動アップグレードの有効化] を指定すると、この設定が有効になります。マイナーバージョン自動アップグレードを有効にすると、新しいマイナーバージョンがリリースされたときに Aurora によって自動的にアップグレードされます。自動アップグレードは、データベースのメンテナンス期間中に実行されます。Aurora MySQL で、この機能が適用されるのは Aurora MySQL バージョン 2 (MySQL 5.7 と互換) のみです。初期に、自動アップグレード手順によって Aurora MySQL DB クラスターはバージョン 2.07.2 に移行されます。Aurora MySQL におけるこの機能の動作の詳細については、「Amazon Aurora MySQL のデータベースアップグレードおよびパッチ」を参照してください。	2020 年 8 月 3 日
Aurora PostgreSQL は PostgreSQL バージョン 11 へのメジャーバージョンアップグレードをサポート	Aurora PostgreSQL では、DB エンジンをメジャーバージョン 11 にアップグレードできるようになりました。詳細については、「Aurora PostgreSQL の PostgreSQL DB エンジンのアップグレード」を参照してください。	2020 年 7 月 28 日

Amazon Aurora が AWS PrivateLink をサポート	Amazon Aurora は、AWS ネットワーク内でアプリケーションと Aurora 間のトラフィックを維持するために、Amazon RDS API コールの Amazon VPC エンドポイントの作成をサポートするようになりました。詳細については、「Amazon Aurora とインターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。	2020 年 7 月 9 日
RDS Proxy の一般公開	RDS Proxy が一般公開されました。RDS Proxy を RDS for MySQL、Aurora MySQL、RDS for PostgreSQL、Aurora PostgreSQL で本番稼働のワークロードに使用できます。RDS Proxy の詳細については、Amazon RDS ユーザーガイドの「Amazon RDS Proxy による接続の管理」、または Aurora ユーザーガイドを参照してください。	2020 年 6 月 30 日

Aurora Global Database の書き込み転送	グローバルデータベースのセカンダリクラスターで書き込み機能を有効にできるようになりました。書き込み転送では、セカンダリクラスターで DML ステートメントを発行し、Aurora が書き込みをプライマリクラスターに転送し、更新されたデータがすべてのセカンダリクラスターにレプリケートされます。詳細については「Aurora グローバルデータベースを使用したセカンダリ AWS リージョン の書き込み転送」を参照してください。	2020 年 6 月 18 日
Aurora が AWS Backup との統合をサポート	AWS Backup を使用して、Aurora DB クラスターのバックアップを管理できます。詳細については、「Aurora DB クラスターのバックアップと復元の概要」を参照してください。	2020 年 6 月 10 日
Aurora PostgreSQL は db.t3.large DB インスタンスクラスをサポート	db.t3.large DB インスタンスクラスを使用する Aurora PostgreSQL DB クラスターを作成できるようになりました。詳細については、「DB インスタンスクラス」を参照してください。	2020 年 6 月 5 日

Aurora Global Database は、PostgreSQL バージョン 11.7 およびマネージド目標復旧時点 (RPO) をサポート	PostgreSQL データベースエンジンバージョン 11.7 用の Aurora Global Database を作成できるようになりました。また、PostgreSQL グローバルデータベースが目標復旧時点 (RPO) を使用して障害から回復する方法を管理することもできます。詳細については、「Aurora Global Database のクロスリージョンの災害対策」を参照してください。	2020 年 6 月 4 日
Aurora MySQL が、データベースアクティビティストリーミングによるデータベースのモニタリングのサポートをスタート	Aurora MySQL に、データベースアクティビティストリーミングが追加されました。これにより、リレーショナルデータベースのデータベースアクティビティのデータストリーミングをほぼリアルタイムで取得できるようになりました。詳細については、「データベースアクティビティストリーミングを使用する」を参照してください。	2020 年 6 月 2 日
クエリエディタが追加の AWS リージョン で使用可能に	Aurora Serverless のクエリエディタが追加の AWS リージョン で使用可能になりました。詳細については、「クエリエディタの提供状況」を参照してください。	2020 年 5 月 28 日

Data API が追加の AWS リージョン で使用可能に	Data API が追加の AWS リージョン で使用可能になりました。詳細については、「Data API の提供状況」を参照してください。	2020 年 5 月 28 日
RDS Proxy が カナダ (中部) リージョン で使用可能に	RDS Proxy プレビューが カナダ (中部) リージョン で利用できるようになりました。RDS Proxy の詳細については、「Amazon RDS Proxy による接続の管理 (プレビュー)」を参照してください。	2020 年 5 月 28 日
Aurora Global Database とクロスリージョンリードレプリカ	Aurora Global Database の場合、セカンダリクラスターと同じリージョンのプライマリクラスターから Aurora MySQL クロスリージョンリードレプリカを作成できるようになりました。Aurora Global Database とクロスリージョンリードレプリカの詳細については、「Amazon Aurora Global Database の使用」および「Amazon Aurora MySQL DB のレプリケーション」を参照してください。	2020 年 5 月 18 日

RDS Proxy が AWS リージョン で使用可能に	RDS Proxy のプレビューは、米国西部 (北カリフォルニア) リージョン、欧州 (ロンドン) リージョン、欧州 (フランクフルト) リージョン、アジアパシフィック (ソウル) リージョン、アジアパシフィック (ムンバイ) リージョン、アジアパシフィック (シンガポール) リージョン、および アジアパシフィック (シドニー) リージョン で利用できるようになりました。RDS Proxy の詳細については、「Amazon RDS Proxy による接続の管理 (プレビュー)」を参照してください。	2020 年 5 月 13 日
Aurora PostgreSQL 互換エディション がオンプレミスまたはセルフホストの Microsoft Active Directory をサポート	Aurora PostgreSQL DB クラスターに接続するときに、ユーザーの Kerberos 認証にオンプレミスまたはセルフホストの Active Directory を使用できるようになりました。詳細については、「Aurora PostgreSQL で Kerberos 認証を使用する」を参照してください。	2020 年 5 月 7 日

Aurora MySQL マルチマスタークラスターがさらに多くの AWS リージョン で使用可能に	Aurora マルチマスタークラスターを アジアパシフィック (ソウル) リージョン、アジアパシフィック (東京) リージョン、アジアパシフィック (ムンバイ) リージョン、および欧州 (フランクフルト) リージョン で作成できるようになりました。マルチマスタークラスターの詳細については、「Aurora マルチマスタークラスターを使用する」を参照してください。	2020 年 5 月 7 日
Performance Insights が実行中の Aurora MySQL クエリの統計分析をサポート	Aurora MySQL DB インスタンスの Performance Insights を使用して、実行中のクエリの統計を分析できるようになりました。詳細については、「クエリの実行の統計を分析する」を参照してください。	2020 年 5 月 5 日
Data API 用の Java クライアントライブラリが一般利用可能に	Data API 用の Java クライアントライブラリが一般利用可能になりました。Data API 用の Java クライアントライブラリをダウンロードして使用できます。このライブラリを使用すると、クライアント側のクラスを Data API のリクエストとレスポンスにマッピングできます。詳細については、「Data API 用の Java クライアントライブラリの使用」を参照してください。	2020 年 4 月 30 日

Amazon Aurora が欧州 (ミラノ) リージョンで使用可能に	Amazon Aurora が欧州 (ミラノ) リージョンで使用可能になりました。詳細については、「リージョンとアベイラビリティーゾーン」を参照してください。	2020 年 4 月 28 日
Amazon Aurora が欧州 (ミラノ) リージョンで使用可能に	Amazon Aurora が欧州 (ミラノ) リージョンで使用可能になりました。詳細については、「リージョンとアベイラビリティーゾーン」を参照してください。	2020 年 4 月 27 日
Amazon Aurora がアフリカ (ケープタウン) リージョンで使用可能に	Amazon Aurora がアフリカ (ケープタウン) リージョンで使用可能になりました。詳細については、「リージョンとアベイラビリティーゾーン」を参照してください。	2020 年 4 月 22 日
Aurora PostgreSQL は、db.r5.16xlarge および db.r5.8xlarge の DB インスタンスクラスをサポートするように	db.r5.16xlarge および db.r5.8xlarge の DB インスタンスクラスを使用する PostgreSQL を実行する Aurora PostgreSQL DB クラスターを作成できるようになりました。詳細については、「Aurora の DB インスタンスクラスのハードウェア仕様」を参照してください。	2020 年 4 月 8 日

PostgreSQL 用の Amazon RDS Proxy	Amazon RDS Proxy が PostgreSQL で利用可能になりました。RDS Proxy を使用すると、クラスターでの接続管理のオーバーヘッドを削減し、「接続が多すぎます」というエラーが発生する可能性も減らすことができます。RDS Proxy は、現在 PostgreSQL でパブリックプレビュー中です。詳細については、「Amazon RDS Proxy による接続の管理 (プレビュー)」を参照してください。	2020 年 4 月 8 日
Aurora Global Database が Aurora PostgreSQL をサポートするようになりました	PostgreSQL データベースエンジン用の Aurora Global Database を作成できるようになりました。Aurora グローバルデータベースは、複数の AWS リージョン リージョンにまたがって、低レイテンシーのグローバル読み取りとリージョン全体の停止からの災害対策を可能にします。詳細については、「Amazon Aurora Global Database の使用」を参照してください。	2020 年 3 月 10 日

Aurora PostgreSQL のメジャーバージョンアップグレードのサポート	Aurora PostgreSQL では、DB エンジンをメジャーバージョンにアップグレードできるようになりました。それによって、特定の PostgreSQL エンジンのバージョンのアップグレード時に、新しいメジャーバージョンにスキップすることができます。詳細については、「Aurora PostgreSQL の PostgreSQL DB エンジンのアップグレード」を参照してください。	2020 年 3 月 4 日
Aurora PostgreSQL が Kerberos 認証をサポート	Aurora PostgreSQL DB クラスターに接続するユーザーを Kerberos 認証を使用して認証できるようになりました。詳細については、「Aurora PostgreSQL で Kerberos 認証を使用する」を参照してください。	2020 年 2 月 28 日
Data API が AWS PrivateLink をサポート	Data API では、AWS ネットワーク内でアプリケーションと Data API 間のトラフィックを維持するために、Data API コールの Amazon VPC エンドポイントの作成がサポートされるようになりました。詳細については、「Data API の Amazon VPC エンドポイント (AWS PrivateLink) を作成する」を参照してください。	2020 年 2 月 6 日

Aurora PostgreSQL での Aurora 機械学習サポート	この aws_ml Aurora PostgreSQL拡張機能は、感情分析のために Amazon Comprehend を呼び出し、独自の機械学習モデルを実行するために SageMaker を呼び出すための、データベースクエリで使用する関数を提供します。詳細については、「Aurora での機械学習 (ML) 機能の使用」を参照してください。	2020 年 2 月 5 日
Aurora PostgreSQL は、Amazon S3 へのデータのエクスポートをサポート	Aurora PostgreSQL DB クラスターからデータをクエリし、Amazon S3 バケットに保存されているファイルに直接エクスポートできます。詳細については、「データを Aurora PostgreSQL DB クラスターから Amazon S3 にエクスポートする」を参照してください。	2020 年 2 月 5 日
DB スナップショットデータの Amazon S3 へのエクスポートに関するサポート	Amazon Aurora では、DB スナップショットデータを Amazon S3 for MySQL および PostgreSQL にエクスポートできます。詳細については、「Amazon S3 への DB スナップショットデータのエクスポート」を参照してください。	2020 年 1 月 9 日

ドキュメント履歴の Aurora MySQL リリースノート	このセクションには、2018 年 8 月 31 日以降にリリースされたバージョンの Aurora MySQL 互換エディション リリースノートに関する履歴項目が含まれています。特定のバージョンの完全なリリースノートについては、履歴項目の初期の列にあるリンクを選択してください。	2019 年 12 月 13 日
Amazon RDS Proxy	Amazon RDS Proxy を使用すると、クラスターでの接続管理のオーバーヘッドを減らし、「接続が多すぎます」エラーが発生する可能性を減らすことができます。各プロキシを RDS DB インスタンスまたは Aurora DB クラスターに関連付けます。次に、アプリケーションの接続文字列でプロキシエンドポイントを使用します。Amazon RDS Proxy は現在パブリックプレビュー状態です。これは、Aurora MySQL データベースエンジンをサポートしています。詳細については、「Amazon RDS Proxy による接続の管理 (プレビュー)」を参照してください。	2019 年 12 月 3 日

Aurora Serverless v1 の Data API がデータ型マッピングのヒントをサポート	ヒントを使用して、Aurora Serverless v1 値を異なる値としてデータベースに送信するように `String` の Data API に対して指示できるようになりました。詳細については、「Data API の呼び出し」を参照してください。	2019 年 11 月 26 日
Aurora Serverless v1 の Data API が Java クライアントライブラリ (プレビュー) をサポート	Data API 用の Java クライアントライブラリをダウンロードして使用できます。このライブラリを使用すると、クライアント側のクラスを Data API のリクエストとレスポンスにマッピングできます。詳細については、「Data API 用の Java クライアントライブラリの使用」を参照してください。	2019 年 11 月 26 日
Aurora PostgreSQL が FedRAMP HIGH に対応	Aurora PostgreSQL は FedRAMP HIGH に対応しています。AWS およびコンプライアンスの取り組みの詳細については、「コンプライアンスプログラムによる AWS 対象範囲内のサービス」を参照してください。	2019 年 11 月 26 日

Amazon Aurora MySQL レプリカで READ COMMITTED 分離レベルを有効化	Aurora MySQL レプリカで READ COMMITTED 分離レベルを有効にできるようになりました。有効にするには、セッションレベルで aurora_read_replica_read_committed_isolation_enabled 構成設定を有効にする必要があります。OLTP クラスターで長時間実行されるクエリに READ COMMITTED 分離レベルを使用すると、履歴リストの長さに関する問題に対処できます。この設定を有効にする前に、Aurora レプリカの分離動作が、READ COMMITTED の通常の MySQL 実装とどのように異なるかを理解しておいてください。詳細については、「Aurora MySQL の分離レベル」を参照してください。	2019 年 11 月 25 日
Performance Insights が実行中の Aurora PostgreSQL クエリの統計分析をサポート	Performance Insights for Aurora PostgreSQL DB インスタンスを使用して、実行中のクエリの統計を分析できるようになりました。詳細については、「クエリの実行の統計を分析する」を参照してください。	2019 年 11 月 25 日

Aurora Global Database に追加できるクラスター数が増加	Aurora Global Database に複数のセカンダリリージョンを追加できるようになりました。低レイテンシーのグローバル読み取りと災害対策を活用できる地理的領域が拡大されました。Aurora Global Database については、「Amazon Aurora Global Database の使用」を参照してください。	2019 年 11 月 25 日
Aurora MySQL での Aurora 機械学習サポート	Aurora MySQL 2.07 以降では、Amazon Comprehend を呼び出してセンチメント分析を行い、SageMaker を呼び出してさまざまな機械学習アルゴリズムを実行することができます。データベースアプリケーションで結果を直接使用するには、クエリにストアド関数の呼び出しを埋め込みます。詳細については、「Aurora での機械学習 (ML) 機能の使用」を参照してください。	2019 年 11 月 25 日

Aurora Global Database でエンジンモード設定が不要に	Aurora Global Database の一部となるクラスターを作成するときに、`--engine-mode=global` を指定する必要がなくなりました。互換性要件を満たすすべての Aurora クラスターを、グローバルデータベースに追加することができます。例えば、クラスターは現在、MySQL 5.6 と互換性のある Aurora MySQL バージョン 1 を使用する必要があります。Aurora Global Database については、「Amazon Aurora Global Database の使用」を参照してください。	2019 年 11 月 25 日
Aurora Global Database が Aurora MySQL バージョン 2 で利用可能	Aurora MySQL 2.07 以降では、MySQL 5.7 との互換性を持つ Aurora Global Database を作成できます。プライマリクラスターまたはセカンダリクラスターに `global` エンジンモードを指定する必要はありません。Aurora MySQL 2.07 以降では、プロビジョニングされた新しいクラスターを Aurora Global Database に追加できます。Aurora Global Database については、「Amazon Aurora Global Database の使用」を参照してください。	2019 年 11 月 25 日

ラボモードなしで Aurora MySQL ホット行の競合の最適化を使用可能	ホット行の競合の最適化は現在、Aurora MySQL で一般的に使用可能であり、Aurora ラボモード設定をオンにする必要はありません。この機能により、同じページの列で多くのトランザクションが競合しているワークロードのスループットが大幅に向上します。この向上には、Aurora MySQL で使用されるロック解除アルゴリズムの変更も含まれます。	2019 年 11 月 19 日
ラボモードなしで Aurora MySQL ハッシュ結合を使用可能	ハッシュ結合機能は現在、Aurora MySQL で一般的に使用可能であり、Aurora ラボモード設定をオンにする必要はありません。等価結合を使用して大量のデータを結合する必要がある場合は、この機能によりクエリのパフォーマンスが向上することがあります。この機能の使用方法については、「Aurora MySQL でのハッシュ結合の使用」を参照してください。	2019 年 11 月 19 日

Aurora MySQL 2.* でさらに多くの db.r5 インスタンスクラスをサポート	Aurora MySQL クラスターで、インスタンスタイプ db.r5.8xlarge、db.r5.16xlarge、および db.r5.24xlarge がサポートされるようになりました。Aurora MySQL クラスターのインスタンスタイプの詳細については、「DB インスタンスクラスの選択」を参照してください。	2019 年 11 月 19 日
Aurora MySQL 2.* でバックトラッキングをサポート	Aurora MySQL 2.* バージョンでは、テーブルや行の誤った削除などのユーザーエラーから迅速に回復できるようになりました。バックトラックを使用すると、バックアップから復元せずに、データベースを過去の時点に戻すことができます。この処理は、大規模なデータベースであっても数秒で完了します。詳細については、「Aurora DB クラスターのバックトラック」を参照してください。	2019 年 11 月 19 日

Aurora の請求タグサポート	タグを使用した Aurora クラスター、Aurora クラスター内の DB インスタンス、I/O、バックアップ、スナップショットなどのリソースコストの割り当てを追跡できるようになりました。AWS Cost Explorer を使用すると、各タグに関連付けられたコストを確認できます。Aurora でのタグの使用の詳細については、「Amazon RDS リソースにタグを付ける」を参照してください。タグの一般的な情報やコスト分析に使用する方法については、「コスト配分タグの使用」および「ユーザー定義のコスト配分タグ」を参照してください。	2019 年 10 月 23 日
Aurora PostgreSQL の Data API	Aurora PostgreSQL は、Amazon Aurora Serverless v1 DB クラスターでの Data API の使用をサポートするようになりました。詳細については、「Aurora Serverless v1 の Data API の使用」を参照してください。	2019 年 9 月 23 日

Aurora PostgreSQL は、CloudWatch Logs へのデータベースログのアップロードをサポート	ログデータを Amazon CloudWatch Logs のロググループに発行するように、Aurora PostgreSQL DB クラスターを設定することができます。CloudWatch Logs を使用すると、ログデータのリアルタイム分析や、CloudWatch を使用したアラームの作成、メトリクスの表示を行うことができます。CloudWatch Logs を使用して、耐久性の高いストレージにログレコードを格納できます。詳細については、「Amazon CloudWatch Logs への Aurora PostgreSQL ログの発行」を参照してください。	2019 年 8 月 9 日
Aurora MySQL のマルチマスタークラスター	Aurora MySQL のマルチマスタークラスターを設定することができます。これらのクラスターでは、DB インスタンスごとに読み書き機能が備わっています。詳細については、「Aurora マルチマスタークラスターの使用」を参照してください。	2019 年 8 月 8 日

Aurora PostgreSQL が Aurora Serverless v1 をサポート	Aurora PostgreSQL で Amazon Aurora Serverless v1 を使用できるようになりました。Aurora Serverless DB クラスターは、アプリケーションニーズに応じて、コンピューティング容量を自動的に起動、シャットダウン、およびスケールアップまたはスケールダウンします。詳細については、「Amazon Aurora Serverless v1 の使用」を参照してください。	2019 年 7 月 9 日
Aurora MySQL のクロスアカウントのクローン作成	これで、AWS アカウント間で Aurora MySQL DB クラスターのクラスターボリュームのクローンを作成できるようになりました。AWS Resource Access Manager (AWS RAM) を介して共有を承認します。クローン作成されたクラスターボリュームは、コピーオンライトメカニズムを使用します。このメカニズムでは、新規データまたは変更されたデータ用の追加のストレージのみ必要となります。Aurora のクローン作成の詳細については、「Aurora DB クラスターでのデータベースのクローン作成」を参照してください。	2019 年 7 月 2 日

Aurora PostgreSQL で db.t3 DB インスタンスクラスをサポート	db.t3 DB インスタンスクラスを使用する Aurora PostgreSQL DB クラスターを作成できるようになりました。詳細については、「DB インスタンスクラス」を参照してください。	2019 年 6 月 20 日
Amazon S3 からの Aurora PostgreSQL のデータをインポートするためのサポート	Amazon S3 ファイルから、Aurora PostgreSQL DB クラスターのテーブルにデータをインポートできるようになりました。詳細については、「Aurora PostgreSQL DB クラスターへの Amazon S3 データのインポート」を参照してください。	2019 年 6 月 19 日
Aurora PostgreSQL のクラスターキャッシュ管理で高速なフェイルオーバーを実現	Aurora PostgreSQL は、フェイルオーバーの発生時にプライマリ DB インスタンスを高速リカバリするためのクラスターキャッシュ管理機能の提供をスタートしました。詳細については、「クラスターキャッシュ管理によるフェイルオーバー後の高速リカバリ」を参照してください。	2019 年 6 月 11 日
Aurora Serverless v1 の Data API が一般利用可能に	ウェブサービスベースのアプリケーションで Data API を使用して Aurora Serverless v1 クラスターにアクセスできます。詳細については、「Aurora Serverless v1 の Data API の使用」を参照してください。	2019 年 5 月 30 日

Aurora PostgreSQL では、データベースアクティビティストリーミングによるデータベースのモニタリングのサポートをスタートしました	Aurora PostgreSQL に、データベースアクティビティストリーミングが追加されました。これにより、リレーショナルデータベースのデータベースアクティビティのデータストリーミングをほぼリアルタイムで取得できるようになりました。詳細については、「データベースアクティビティストリーミングを使用する」を参照してください。	2019 年 5 月 30 日
Amazon Aurora の推奨事項	Amazon Aurora では、Aurora リソースに対して自動化された推奨事項の提供をスタートしました。詳細については、「Amazon Aurora 推奨事項を使用する」を参照してください。	2019 年 5 月 22 日
Performance Insights が Aurora Global Database をサポート	Aurora Global Database で Performance Insights を使用できるようになりました。Aurora の Performance Insights については、「Amazon RDS Performance Insights の使用」を参照してください。Aurora Global Database については、「Aurora Global Database の使用」を参照してください。	2019 年 5 月 13 日

Performance Insights が Aurora MySQL 5.7 で利用可能に	Amazon RDS Performance Insights は、Aurora MySQL 2.x バージョンのサポートをスタートしました。このバージョンは、MySQL 5.7 との互換性があります。詳細については、「Amazon RDS Performance Insights の使用」を参照してください。	2019 年 5 月 3 日
Aurora Global Database で利用可能な AWS リージョン を拡大	Aurora が使用可能なほとんどの AWS リージョン で Aurora グローバルデータベースを作成できるようになりました。Aurora Global Database については、「Amazon Aurora Global Database の使用」を参照してください。	2019 年 4 月 30 日
Aurora Serverless v1 の最小容量が 1	Aurora Serverless v1 クラスターに使用できる最小容量設定は 1 です。以前の最小容量は 2 でした。Aurora Serverless の容量値の指定については、「Aurora Serverless v1 DB クラスターの容量の設定」を参照してください。	2019 年 4 月 29 日
Aurora Serverless v1 のタイムアウトアクション	Aurora Serverless v1 の容量変更がタイムアウトしたときに実行するアクションを指定できるようになりました。詳細については、「容量の変更のタイムアウトアクション」を参照してください。	2019 年 4 月 29 日

1 秒単位の請求	オンデマンドインスタンスの場合、Amazon RDS は、すべての AWS リージョン (AWS GovCloud (米国) は除く) で 1 秒単位で請求されるようになりました。詳細については、「Aurora DB インスタンスの請求」を参照してください。	2019 年 4 月 25 日
AWS リージョン 間での Aurora Serverless v1 スナップショットの共有	Aurora Serverless v1 では、スナップショットは常に暗号化されます。スナップショットを独自の AWS KMS key で暗号化すると、スナップショットをコピーしたり、AWS リージョン リージョン間で共有したりできます。Aurora Serverless v1 DB クラスターのスナップショットについては、「Aurora Serverless v1 とスナップショット」を参照してください。	2019 年 4 月 17 日
Amazon S3 から MySQL 5.7 バックアップを復元	MySQL バージョン 5.7 データベースのバックアップを作成して Amazon S3 に保存し、そのバックアップファイルを新しい Aurora MySQL DB クラスターに復元できるようになりました。詳細については、「外部の MySQL データベースから Aurora MySQL DB クラスターへのデータ移行」を参照してください。	2019 年 4 月 17 日

リージョン間で Aurora Serverless v1 スナップショットを共有	Aurora Serverless v1 では、スナップショットは常に暗号化されます。スナップショットを独自の AWS KMS key で暗号化すると、スナップショットをコピーしたり、リージョン間で共有したりできます。Aurora Serverless v1 DB クラスターのスナップショットについては、「Aurora Serverless とスナップショット」を参照してください。	2019 年 4 月 16 日
Aurora の PoC (概念実証) に関するチュートリアル	Aurora を使用してアプリケーションとワークロードを試すための PoC (概念実証) の実施方法を示します。チュートリアル全体については、「Aurora の PoC (概念実証) の実行」を参照してください。	2019 年 4 月 16 日
Aurora Serverless v1 が Amazon S3 バックアップからの復元をサポート	Amazon S3 から Aurora Serverless クラスターにバックアップをインポートできるようになりました。手順の詳細については、「Amazon S3 バケットを使用した MySQL からのデータの移行」を参照してください。	2019 年 4 月 16 日

Aurora Serverless v1 の新しい変更可能なパラメータ	Aurora Serverless v1 クラスターの DB パラメータ innodb_file_format 、innodb_file_per_table 、innodb_large_prefix 、innodb_lock_wait_timeout 、innodb_monitor_disable 、innodb_monitor_enable 、innodb_monitor_reset 、innodb_monitor_reset_all 、innodb_print_all_deadlocks 、log_warnings 、net_read_timeout 、net_retry_count 、net_write_timeout 、sql_mode、tx_isolation を変更できるようになりました。Aurora Serverless v1 クラスターの設定パラメータの詳細については、「Aurora Serverless v1 とパラメータグループ」を参照してください。	2019 年 4 月 4 日
Aurora PostgreSQL は db.r5 DB インスタンスクラスをサポート	db.r5 DB インスタンスクラスを使用する Aurora PostgreSQL DB クラスターを作成できるようになりました。詳細については、「DB インスタンスクラス」を参照してください。	2019 年 4 月 4 日

Aurora PostgreSQL 論理的なレプリケーション	これで、論理的なレプリケーションを使用して、Aurora PostgreSQL クラスターのデータベース部分をレプリケートできるようになりました。詳細については、「PostgreSQL 論理的なレプリケーションを使用する」を参照してください。	2019 年 3 月 28 日
Aurora MySQL 2.04 での GTID のサポート	MySQL 5.7 のグローバルトランザクション ID (GTID) 機能でレプリケーションを使用できるようになりました。この機能では、Aurora MySQL と外部の MySQL 5.7 互換データベースの間で、簡単にバイナリログ (binlog) レプリケーションを実行できます。このレプリケーションでは、Aurora MySQL クラスターを送信元または送信先として使用することができます。この機能は Aurora MySQL 2.04 以降で使用できます。GTID ベースのレプリケーションと Aurora MySQL の詳細については、「Aurora MySQL の GTID ベースレプリケーションを使用する」を参照してください。	2019 年 3 月 25 日

Amazon CloudWatch への Aurora Serverless v1 ログのアップロード	Aurora で、Aurora Serverless v1 クラスターの CloudWatch にデータベースログをアップロードできるようになりました。詳細については、「Aurora Serverless DB クラスターの表示」を参照してください。この機能強化の一環として、DB クラスターパラメータグループでインスタンスレベルのパラメータの値を定義できるようになりました。これらの値は、DB パラメータグループで上書きしない限り、クラスター内のすべての DB インスタンスに適用されます。詳細については、「DB パラメータグループと DB クラスターパラメータグループを使用する」を参照してください。	2019 年 2 月 25 日
Aurora MySQL で db.t3 DB インスタンスクラスをサポート	db.t3 DB インスタンスクラスを使用する Aurora MySQL DB クラスターを作成できるようになりました。詳細については、「DB インスタンスクラス」を参照してください。	2019 年 2 月 25 日
Aurora MySQL で db.r5 DB インスタンスクラスをサポート	db.r5 DB インスタンスクラスを使用する Aurora MySQL DB クラスターを作成できるようになりました。詳細については、「DB インスタンスクラス」を参照してください。	2019 年 2 月 25 日

Aurora MySQL の Performance Insights のカウンター	Aurora MySQL DB インスタンスの Performance Insights 図にパフォーマンスカウンターを追加できるようになりました。詳細については、「Performance Insights ダッシュボードのコンポーネント」を参照してください。	2019 年 2 月 19 日
Amazon RDS Performance Insights で Aurora MySQL の SQL テキストをさらに表示可能に	Amazon RDS Performance Insights では、Aurora MySQL DB インスタンスの Performance Insights ダッシュボードで SQL テキストをより多く表示できるようになりました。詳細については、「Performance Insights ダッシュボードでの SQL テキストの表示量を増やす」を参照してください。	2019 年 2 月 6 日
Amazon RDS Performance Insights で Aurora PostgreSQL の SQL テキストをさらに表示可能に	Amazon RDS Performance Insights では、Aurora PostgreSQL DB インスタンスの Performance Insights ダッシュボードで SQL テキストをより多く表示できるようになりました。詳細については、「Performance Insights ダッシュボードでの SQL テキストの表示量を増やす」を参照してください。	2019 年 1 月 24 日

Aurora バックアップの料金	Amazon CloudWatch の TotalBackupStorageBilled 、SnapshotStorageUsed 、BackupRetentionPeriodStorageUsed の各メトリクスを使用して Aurora バックアップの領域使用量をモニタリングできます。CloudWatch のメトリクスの詳しい使用法については、「モニタリングの概要」を参照してください。バックアップデータのストレージの詳しい管理方法については、「Aurora バックアップストレージの使用情報を確認する」を参照してください。	2019 年 1 月 3 日
Performance Insights のカウンター	Performance Insights 図にパフォーマンスカウンターを追加できるようになりました。詳細については、「Performance Insights ダッシュボードのコンポーネント」を参照してください。	2018 年 12 月 6 日

Aurora Global Database	Aurora Global Database を作成できるようになりました。Aurora グローバルデータベースは、複数の AWS リージョン リージョンにまたがって、低レイテンシーのグローバル読み取りとリージョン全体の停止からの災害対策を可能にします。詳細については、「Amazon Aurora Global Database の使用」を参照してください。	2018 年 11 月 28 日
Aurora PostgreSQL のクエリ計画管理	Aurora PostgreSQL で、PostgreSQL クエリ実行計画を管理するために使用できるクエリ計画管理を利用できるようになりました。詳細については、「Aurora PostgreSQL のクエリ実行計画の管理」を参照してください。	2018 年 11 月 20 日
Aurora Serverless v1 のクエリエディタ (ベータ版)	Aurora Serverless v1 クラスターの Amazon RDS コンソールで SQL ステートメントを実行できます。詳細については、「Aurora Serverless v1 のクエリエディタの使用」を参照してください。	2018 年 11 月 20 日

Aurora Serverless v1 の Data API (ベータ版)	ウェブサービスベースのアプリケーションで Data API を使用して Aurora Serverless v1 クラスターにアクセスできます。詳細については、「Aurora サーバーレスの Data API の使用」を参照してください。	2018 年 11 月 20 日
TLS による Aurora Serverless v1 のサポート	Aurora Serverless v1 クラスターは TLS/SSL 暗号化をサポートします。詳細については、「Aurora サーバーレスの TLS/SSL」を参照してください。	2018 年 11 月 19 日
カスタムエンドポイント	任意の DB インスタンスのセットに関連付けられたエンドポイントを作成できるようになりました。この機能は、Aurora クラスターで一部の DB インスタンスの容量や設定が他のインスタンスと異なる場合に、ロードバランシングと高可用性を強化します。インスタンスエンドポイントを通じて特定の DB インスタンスに接続する代わりに、カスタムエンドポイントを使用できます。詳細については、「Amazon Aurora の接続管理」を参照してください。	2018 年 11 月12 日

Aurora PostgreSQL での IAM 認証のサポート	Aurora PostgreSQL で IAM 認証をサポートするようになりました。詳細については、「IAM データベース認証」を参照してください。	2018 年 11 月 8 日
復元およびポイントインタイムリカバリのカスタムパラメータグループ	スナップショットを復元するとき、またはポイントインタイムリカバリ操作を実行するときに、カスタムパラメータグループを指定できるようになりました。詳細については、「DB クラスターのスナップショットの復元」および「特定の時間への DB クラスターの復元」を参照してください。	2018 年 10 月 15 日
Aurora DB クラスターの削除保護	DB クラスターの削除保護を有効にすると、どのユーザーもデータベースを削除できません。詳細については、「DB クラスターの削除」を参照してください。	2018 年 9 月 26 日
Aurora 機能の停止/スタート	Aurora クラスター全体を 1 回のオペレーションで停止またはスタートできるようになりました。詳細については、「Aurora クラスターの停止とスタート」を参照してください。	2018 年 9 月 24 日

Aurora MySQL の並列クエリ機能	Aurora MySQL は、Aurora ストレージインフラストラクチャ全体のクエリで I/O 作業を並列処理するオプションの提供をスタートしました。この機能により、データ集約型の分析クエリが高速化されます。多くの場合、これはワークロードで最も時間がかかる操作です。詳細については、「Aurora MySQL の並列クエリの操作」を参照してください。	2018 年 9 月 20 日
新しいガイド	これは Amazon Aurora ユーザーガイドの初期のリリースです。	2018 年 8 月 31 日

AWS 用語集

AWS の最新の用語については、「AWS の用語集 リファレンス」の「AWS 用語集」を参照してください。