aws

開発者ガイド

Amazon Timestream

Amazon Timestream: 開発者ガイド

Table of Contents

Amazon Timestream for とは LiveAnalytics

Amazon Timestream for LiveAnalytics は、高速でスケーラブルなフルマネージドの専用時系列データベースで、1 日あたり数兆の時系列データポイントを簡単に保存および分析できます。Timestream for LiveAnalytics は、最近のデータをメモリに保存し、履歴データをユーザー定義ポリシーに基づいてコスト最適化ストレージ層に移動することで、時系列データのライフサイクルを管理する時間とコストを削減します。 LiveAnalyticsの専用クエリエンジンの Timestream を使用すると、その場所を指定しなくても、最近のデータと履歴データの両方にアクセスして分析できます。Amazon Timestream for LiveAnalytics には時系列分析関数が組み込まれており、データの傾向やパターンをほぼリアルタイムで特定できます。の Timestream LiveAnalytics はサーバーレスで、容量とパフォーマンスを調整するために自動的にスケールアップまたはスケールダウンします。基盤となるインフラストラクチャを管理する必要がないため、アプリケーションの最適化と構築に集中できます。

の Timestream は、データ収集、視覚化、機械学習のために一般的に使用されるサービスと統合 LiveAnalytics されます。、Amazon Kinesis AWS IoT Core、Amazon 、およびオープンソース Telegraf LiveAnalytics を使用してMSK、データを Amazon Timestream に送信できます。 Amazon Kinesis Amazon QuickSight、Grafana、ビジネスインテリジェンスツールを使用してデータを視覚化できますJDBC。Amazon SageMaker を機械学習 LiveAnalytics の Timestream で使用することもできます。

LiveAnalytics 主な利点のタイムストリーム

の Amazon Timestream の主な利点 LiveAnalytics は次のとおりです。

- 自動スケーリングによるサーバーレス - の Amazon Timestream では LiveAnalytics、管理するサーバーはなく、プロビジョニングする容量もありません。アプリケーションのニーズが変化すると、の Timestream LiveAnalytics は自動的にスケーリングされ、容量が調整されます。
- データライフサイクル管理 - Amazon Timestream for は、データライフサイクル管理の複雑なプロセス LiveAnalytics を簡素化します。ストレージ階層化を提供し、最近のデータにはメモリストア、履歴データにはマグネティックストアがあります。Amazon Timestream は、ユーザー設定可能なポリシーに基づいて、メモリストアからマグネティックストアへのデータの転送を自動化します。
- データアクセスの簡素化 - の Amazon Timestream を使用すると LiveAnalytics、異なるツールを使用して最近のデータや履歴データにアクセスする必要がなくなります。 LiveAnalyticsの専用クエ

リエンジンの Amazon Timestream は、データの場所を指定しなくても、ストレージ階層間で透過的にデータにアクセスして結合します。

- 時系列専用 - 平滑化SQL、近似、補間用の時系列関数を組み込み、 を使用して時系列データをすばやく分析できます。の Timestream は、高度な集計、ウィンドウ関数、配列や行などの複雑なデータ型 LiveAnalytics もサポートしています。
- 常に暗号化 - の Amazon Timestream は、保管中でも転送中でも、時系列データを常に暗号化 LiveAnalytics します。Amazon Timestream for では、マグネティックストア内のデータを暗号化するための AWS KMS カスタマーマネージドキー (CMK) を指定 LiveAnalytics することもできます。
- 高可用性 - Amazon Timestream は、データを自動的にレプリケートし、1 つの AWS リージョン内の少なくとも 3 つの異なるアベイラビリティーゾーンにリソースを割り当てることで、書き込みリクエストと読み取りリクエストの高可用性を確保します。詳細については、「Timestream サービスレベル契約」を参照してください。
- 耐久性 - Amazon Timestream は、1 つの AWS リージョン内の異なるアベイラビリティーゾーン間でメモリとマグネティックストアのデータを自動的にレプリケートすることで、データの耐久性を確保します。すべてのデータは、書き込みリクエストの完了を確認する前にディスクに書き込まれます。

LiveAnalytics ユースケースのタイムストリーム

の Timestream のユースケースのリストが増えている例 LiveAnalytics は次のとおりです。

- メトリクスをモニタリングして、アプリケーションのパフォーマンスと可用性を向上させます。
- 産業テレメトリの保存と分析により、機器の管理とメンテナンスを効率化します。
- アプリケーションとのユーザーインタラクションを経時的に追跡します。
- IoT センサーデータのストレージと分析。

の Timestream の開始方法 LiveAnalytics

最初に以下のセクションを読むことをお勧めします。

- チュートリアル - サンプルデータセットが入力されたデータベースを作成し、サンプルクエリを実行します。
- LiveAnalytics 概念の Amazon Timestream - LiveAnalytics 概念に不可欠な Timestream を学ぶため。

- の Timestream へのアクセス LiveAnalytics - コンソール、 AWS CLI、または LiveAnalytics を使用して Timestream にアクセスする方法について説明しますAPI。
- クォータ - プロビジョニングできる LiveAnalytics コンポーネントの Timestream の数に関するクォータについて説明します。

Timestream for のアプリケーションの開発をすばやく開始する方法については LiveAnalytics、以下を参照してください。

- の使用 AWS SDKs
- クエリ言語リファレンス

仕組み

以下のセクションでは、Amazon Timestream for Live Analytics サービスコンポーネントの概要と、それらの相互作用について説明します。

この概要を読み終えたら、 の Timestream へのアクセス LiveAnalytics セクションを参照して、 コンソール AWS CLI、または を使用して Timestream for Live Analytics にアクセスする方法を確認してくださいSDKs。

トピック

- LiveAnalytics 概念の Amazon Timestream
- アーキテクチャ
- 書き込み
- ストレージ
- クエリ
- スケジュールされたクエリ
- Timestream コンピューティングユニット (TCU)

LiveAnalytics 概念の Amazon Timestream

時系列データは、時間間隔で記録された一連のデータポイントです。このタイプのデータは、時間の経過とともに変化するイベントを測定するために使用されます。次に例を示します。

- 経時的な株価

- 経時的な温度測定値
- CPU 時間の経過に伴うEC2インスタンスの使用率

時系列データでは、各データポイントはタイムスタンプ、1 つ以上の属性、および時間の経過とともに変化するイベントで構成されます。このデータを使用して、アプリケーションのパフォーマンスとヘルスに関するインサイトを取得し、異常を検出し、最適化の機会を特定できます。例えば、DevOps エンジニアは、インフラストラクチャのパフォーマンスメトリクスの変化を測定するデータを表示したい場合があります。メーカーは、施設全体の機器の変化を測定する IoT センサーデータを追跡したい場合があります。オンラインマーケターは、ユーザーが時間の経過とともにウェブサイトをナビゲートする方法をキャプチャするクリックストリームデータを分析したい場合があります。時系列データは非常に大量の複数のソースから生成されるため、ほぼリアルタイムでコスト効率の高い方法で収集する必要があるため、データの整理と分析に役立つ効率的なストレージが必要です。

以下は、 の Timestream の主要な概念です LiveAnalytics。

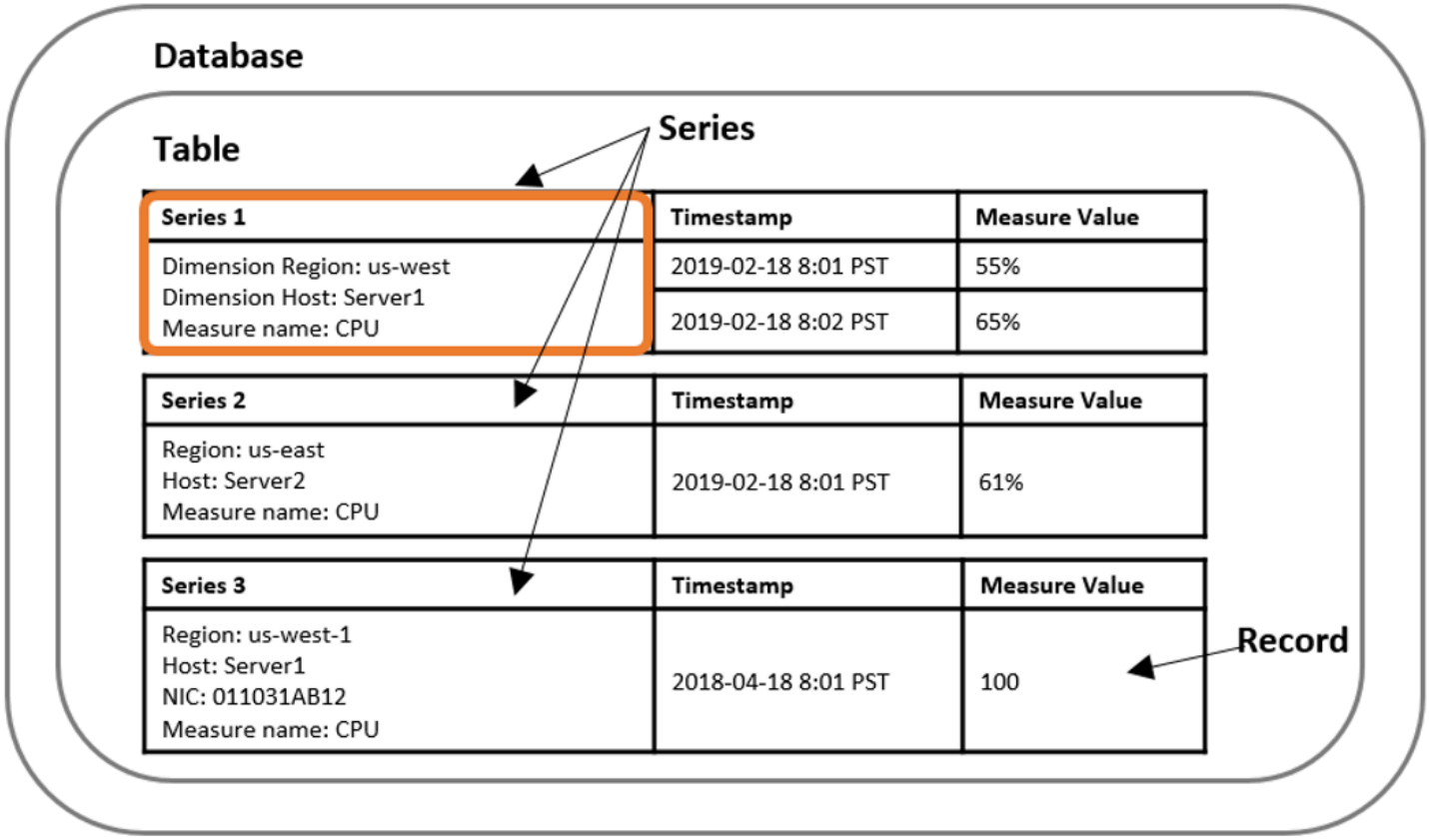
- 時系列 - 時間間隔で記録された 1 つ以上のデータポイント (またはレコード) のシーケンス。 例としては、時間の経過に伴う株式の価格、時間の経過に伴うEC2インスタンスの CPUまたはメモリ使用率、時間の経過に伴う IoT センサーの温度/圧力の読み取りなどがあります。
- レコード - 時系列内の 1 つのデータポイント。
- ディメンション - 時系列のメタデータを記述する属性。ディメンションには、そのディメンションの名前と値を含みます。次の例を考えます。
  - 証券取引所をディメンションと見なす場合、ディメンション名は「株式交換」、ディメンション値は「」ですNYSE。
  - AWS リージョンをディメンションと見なす場合、ディメンション名は「region」で、ディメンション値は「us-east-1」です。
  - IoT センサーの場合、ディメンション名は「デバイス ID」、ディメンション値は「12345」です。
- 測定 - レコードによって測定される実際の値。 例としては、株価、 CPUまたは メモリ使用率、温度または湿度の読み取りなどがあります。メジャーは、メジャー名とメジャー値で構成されます。次の例を考えます。
  - 株価の場合、メジャー名は「株価」で、メジャー値はある時点での実際の株価です。
  - CPU 使用率の場合、測定値名はCPU「使用率」で、測定値は実際のCPU使用率です。

メジャーは、マルチメジャーレコードまたは単一メジャーレコード LiveAnalytics として Timestream でモデル化できます。詳細については、「マルチメジャーレコードと単一メジャーレコードの比較」を参照してください。

- タイムスタンプ - 特定のレコードの測定値が収集された日時を示します。のタイムストリームは、ナノ秒の粒度でタイムスタンプ LiveAnalytics をサポートします。
- 表 - 関連する一連の時系列のコンテナ。
- データベース - テーブルの最上位コンテナ。

LiveAnalytics 概念の Timestream の概要

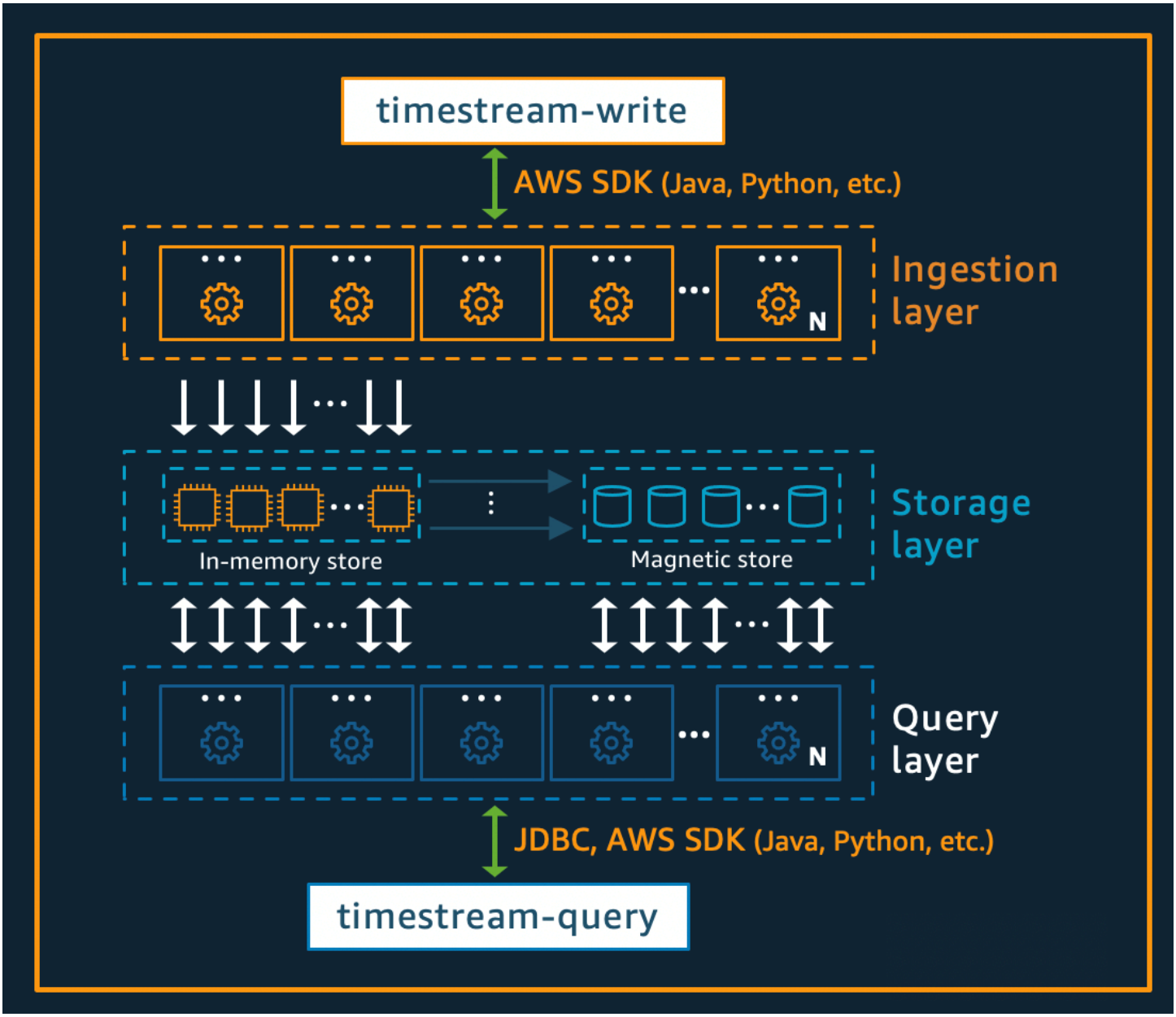
データベースには 0 つ以上のテーブル が含まれています。各テーブルには、0 つ以上の時系列 が含まれます。各時系列は、指定された粒度 の特定の時間間隔の一連のレコードで構成されます。各時系列は、そのメタデータまたはディメンション 、そのデータまたは測定値 、およびそのタイムスタンプ を使用して記述できます。



アーキテクチャ

Amazon Timestream for Live Analytics は、時系列データを大規模に収集、保存、処理するためにゼロから設計されています。サーバーレスアーキテクチャは、個別にスケーリングできる、完全に分離されたデータの取り込み、ストレージ、クエリ処理システムをサポートします。この設計により、各サブシステムが簡素化され、揺るぎない信頼性を実現し、ボトルネックのスケーリングを排除し、相関するシステム障害の可能性を減らすことが容易になります。これらの各要因は、システムがスケールするにつれてより重要になります。

トピック



書き込みアーキテクチャ

時系列データを記述する場合、Amazon Timestream for Live Analytics は、テーブル、パーティションの書き込みを、高スループットのデータ書き込みを処理する耐障害性メモリストアインスタンスにルーティングします。次に、メモリストアは、3 つのアベイラビリティーゾーン () にデータをレプリケートする別のストレージシステムで耐久性を実現しますAZs。レプリケーションはクォーラムベースであるため、ノードまたは AZ 全体の損失によって書き込みの可用性が損なわれることはありません。ほぼリアルタイムでは、クエリを処理するために、他のメモリ内ストレージノードがデータに同期します。リーダーレプリカノードAZsも、高い読み取り可用性を確保するためにスパンします。

Timestream for Live Analytics は、スループットの低い遅延受信データを生成するアプリケーション向けに、データをマグネティックストアに直接書き込むことをサポートしています。遅延受信データは、現在の時刻よりも前のタイムスタンプを持つデータです。メモリストアの高スループット書き込みと同様に、マグネティックストアに書き込まれたデータは 3 つにわたってレプリケートAZsされ、レプリケーションはクォーラムベースです。

データがメモリに書き込まれるか、マグネティックストアに書き込まれるかにかかわらず、Timestream for Live Analytics は、ストレージに書き込む前にデータを自動的にインデックス化してパーティション化します。ライブ分析の 1 つの Timestream テーブルには、数百、数千、さらには数百万のパーティションがある場合があります。個々のパーティションは、直接相互に通信せず、データ (共有なしアーキテクチャ) を共有しません。代わりに、テーブルのパーティショニングは、可用性の高いパーティション追跡およびインデックス作成サービスを通じて追跡されます。これにより、システム内の障害の影響を最小限に抑え、相関する障害の可能性を大幅に低減するために特別に設計された懸念が別の分離されます。

ストレージアーキテクチャ

データが Timestream for Live Analytics に保存されると、データはデータで記述されたコンテキスト属性に基づいて、時間順および時間ごとに整理されます。時系列システムを大規模にスケーリングするには、時間に加えて「スペース」を分割するパーティショニングスキームを持つことが重要です。これは、ほとんどの時系列データが現在の時刻またはその前後に書き込まれるためです。その結果、時間だけでパーティション分割しても、書き込みトラフィックを分散したり、クエリ時にデータを効果的にプルーニングしたりするのに適していません。これは極端なスケールの時系列処理にとって重要であり、Timestream for Live Analytics は、サーバーレス方式で、現在の他の主要なシステムよりも桁違いにスケールアップできるようになりました。結果のパーティションは、「タイル」と呼ばれます。これは、二次元空間 (類似したサイズになるように設計されている) の分割を表すためです。Live Analytics テーブルのタイムストリームは、単一のパーティション (タイル) として始まり、スループットに応じて空間ディメンションに分割されます。タイルが特定のサイズに達すると、データサイズが大きくなるにつれて読み取り並列性を向上させるために、時間ディメンションに分割されます。

Timestream for Live Analytics は、時系列データのライフサイクルを自動的に管理するように設計されています。Timestream for Live Analytics には、インメモリストアとコスト効率の高いマグネティックストアの 2 つのデータストアがあります。また、ストア間でデータを自動的に転送するようにテーブルレベルのポリシーを設定することもサポートしています。受信高スループットデータ書き込みは、データが書き込み用に最適化されているメモリストア、およびダッシュボードに電力を供給し、タイプのクエリにアラートを送信するために現在時間頃に実行される読み取りに実行されます。書き込み、アラート、ダッシュボード作成のメイン時間枠が経過すると、データがメモリ

ストアからマグネティックストアに自動的に流れ、コストが最適化されます。Timestream for Live Analytics では、この目的のためにメモリストアにデータ保持ポリシーを設定できます。遅延受信データのデータ書き込みは、マグネティックストアに直接書き込まれます。

データがマグネティックストアで利用可能になると (メモリストアの保持期間が終了するため、またはマグネティックストアへの直接書き込みのため)、大量のデータ読み取りに高度に最適化された形式に再編成されます。マグネティックストアには、データが有用性を上回っている時間しきい値がある場合に設定できるデータ保持ポリシーもあります。データがマグネティックストアの保持ポリシーに定義されている時間範囲を超えると、自動的に削除されます。したがって、Timestream for Live Analytics では、一部の設定を除き、データのライフサイクル管理はシーンの背後でシームレスに行われます。

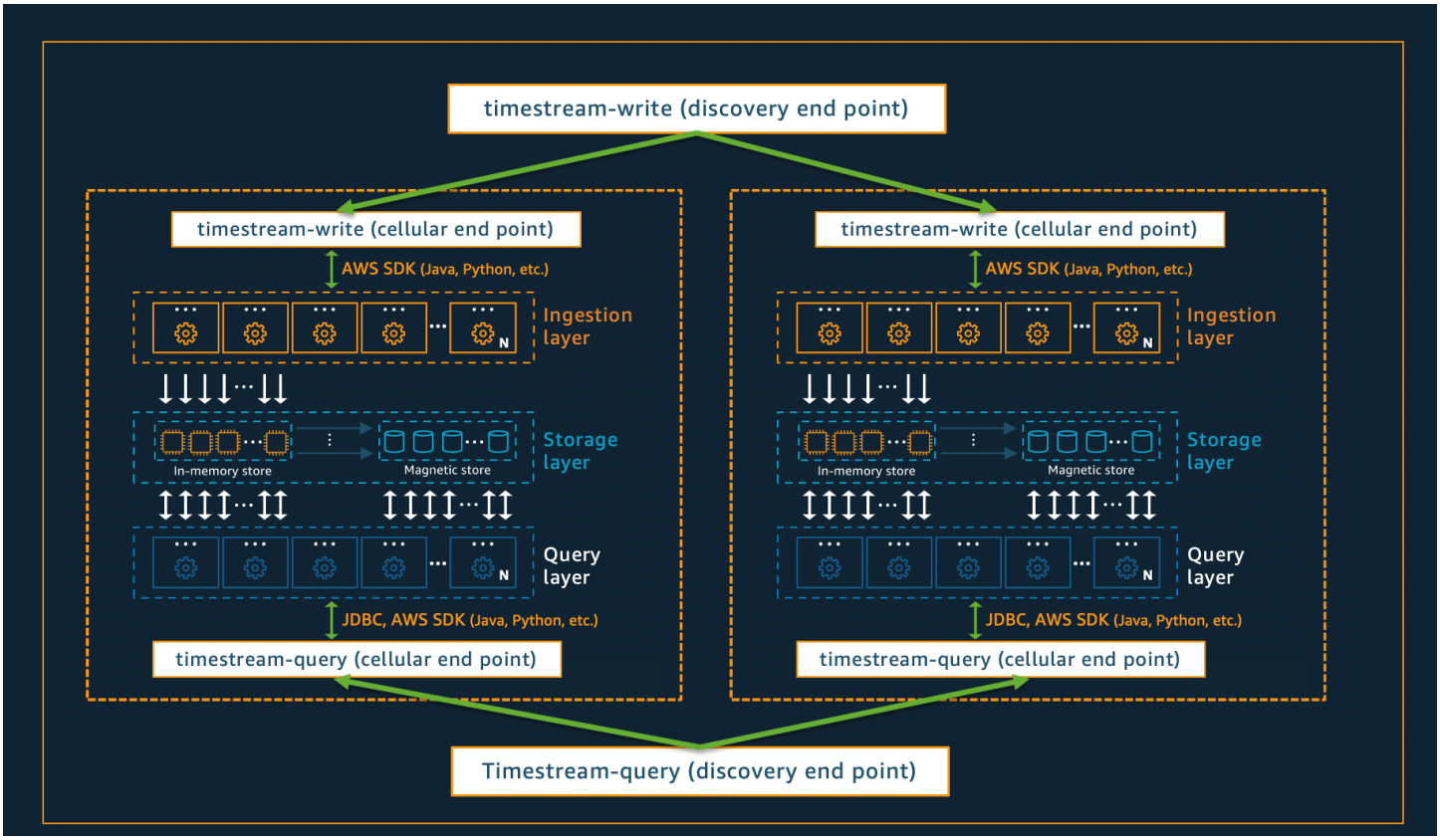
クエリアーキテクチャ

Live Analytics クエリの Timestream SQL は、時系列固有のサポート (時系列固有のデータ型と関数) を拡張した文法で表されるため、 に精通しているデベロッパーにとって学習曲線は簡単ですSQL。その後、クエリは適応型の分散クエリエンジンによって処理されます。このエンジンは、タイル追跡およびインデックス作成サービスのメタデータを使用して、クエリの発行時にデータストア間でシームレスにデータにアクセスし、組み合わせます。これにより、Rube Goldberg の複雑さの多くをシンプルで使い慣れたデータベース抽象化にまとめるため、お客様とうまく共鳴するエクスペリエンスが得られます。

クエリは専用のワーカーフリートによって実行され、特定のクエリを実行するために登録されたワーカーの数は、クエリの複雑さとデータサイズによって決まります。大規模なデータセットに対する複雑なクエリのパフォーマンスは、クエリランタイムフリートとシステムのストレージフリートの両方で、大規模な並列処理を通じて達成されます。大量のデータを迅速かつ効率的に分析できることは、Timestream for Live Analytics の最大の強みの 1 つです。テラバイト、さらにはペタバイトを超えるデータを実行する 1 つのクエリでは、数千台のマシンが同時に動作する可能性があります。

セルラーアーキテクチャ

Timestream for Live Analytics がアプリケーションに事実上無限のスケールを提供しながら、同時に 99.99% の可用性を確保するために、システムはセルラーアーキテクチャを使用して設計されています。Timestream for Live Analytics は、システム全体をスケーリングするのではなく、それ自体の複数の小さなコピーにセグメント化し、セル と呼ばれます。これにより、セルをフルスケールでテストでき、1 つのセルのシステム問題が特定のリージョンの他のセルのアクティビティに影響を与えるのを防ぐことができます。Timestream for Live Analytics はリージョンごとに複数のセルをサポートするように設計されていますが、リージョンに 2 つのセルがある架空のシナリオを検討してください。



上記のシナリオでは、データ取り込みリクエストとクエリリクエストは、最初にデータ取り込みとクエリの検出エンドポイントによって処理されます。次に、検出エンドポイントは、顧客データを含むセルを識別し、そのセルの適切な取り込みまたはクエリエンドポイントにリクエストを指示します。を使用する場合SDKs、これらのエンドポイント管理タスクは透過的に処理されます。

> **Note**
>
> Timestream for Live Analytics でVPCエンドポイントを使用する場合、または Timestream for Live Analytics のRESTAPIオペレーションに直接アクセスする場合は、セルラーエンドポイントと直接やり取りする必要があります。これを行う方法については、VPC「エンドポイントのセットアップ方法」のVPC「エンドポイント」とRESTAPI「オペレーションの直接呼び出し方法」の「エンドポイント検出パターン」を参照してください。

書き込み

接続されたデバイス、IT システム、産業機器から時系列データを収集し、それを Timestream for Live Analytics に書き込むことができます。Timestream for Live Analytics では、時系列が同じテーブルに属する場合、単一の時系列のデータポイントや、単一の書き込みリクエスト内の多数の系列の

データポイントを書き込むことができます。Timestream for Live Analytics では、データベースへの書き込みを呼び出すときに指定した測定値のディメンション名とデータ型に基づいて、Timestream for Live Analytics テーブルの列名とデータ型を自動的に検出する柔軟なスキーマが用意されています。データのバッチを Timestream for Live Analytics に書き込むこともできます。

> **Note**
>
> Timestream for Live Analytics は、読み取りの最終的な整合性セマンティクスをサポートします。つまり、データのバッチを Timestream for Live Analytics に書き込んだ直後にデータをクエリすると、クエリ結果は最近完了した書き込みオペレーションの結果を反映しない可能性があります。結果には、古いデータが含まれている場合もあります。同様に、1 つ以上の新しいディメンションで時系列データを記述している間、クエリは列の部分的なサブセットを短期間返すことができます。これらのクエリリクエストを短時間後に繰り返すと、結果は最新のデータを返します。

AWS SDKs、AWS CLI、または を使用して、AWS Lambda、AWS IoT Core、Amazon Managed Service for Apache Flink、Amazon KinesisAmazon MSK、および を介してデータを書き込むことができますオープンソース Telegraf。

トピック

- データ型
- 事前スキーマ定義なし
- データの作成 (挿入とアップサート)
- 読み取りの最終的な整合性
- による書き込みのバッチ処理 WriteRecords API
- バッチロード
- オペレーションとバッチロードの選択 WriteRecords API

データ型

Timestream for Live Analytics では、書き込み用に次のデータ型がサポートされています。

データ型	説明
BIGINT	64 ビット符号付き整数を表します。

データ型	説明
BOOLEAN	ロジックの 2 つの真理値、つまり true と false を表します。
DOUBLE	IEEE Standard 754 for Binary Floating-Point Arithmetic を実装する 64 ビット可変精度。 **Note** `Infinity` および のクエリ言語関数は、クエリで使用できる と の 2 `NaN`重値があります。ただし、これらの値を Timestream に書き込むことはできません。
VARCHAR	オプションの最大長を持つ可変長の文字データ。最大制限は 2 KB です。
MULTI	マルチメジャーレコードのデータ型。このデータ型には、タイプ `BIGINT`、、`BOOLEAN`、`VARCHAR`、および `DOUBLE`の 1 つ以上のメジャーが含まれます`TIMESTAMP` 。
TIMESTAMP	でナノ秒の精度時間を使用してインスタンスを時間単位で表し UTC、Unix 時間からの時間を追跡します。このデータ型は現在、マルチメジャーレコード (タイプ の測定値内`MULTI`) でのみサポートされています。 *YYYY*-*MM*-*DD* *hh*:*mm*:*ss*.*sssssssss* ~ の範囲のサポートタイムスタンプを書き込み`1970-01-01 00:00:00.000000000` ます`2262-04-11 23:47:16.854775807` 。

事前スキーマ定義なし

Amazon Timestream for Live Analytics にデータを送信する前に、、Timestream for Live Analytics AWS Management Console、SDKsまたは Timestream for Live Analytics APIオペレーションを使用してデータベースとテーブルを作成する必要があります。詳細については、「データベースを作成する」および「テーブルを作成する」を参照してください。テーブルの作成中に、スキーマを事前に定義する必要はありません。Amazon Timestream for Live Analytics は、送信されるデータポイント

の測定値とディメンションに基づいてスキーマを自動的に検出するため、急速に変化する時系列データに合わせてスキーマをオフラインで変更する必要がなくなります。

データの作成 (挿入とアップサート)

Amazon Timestream for Live Analytics の書き込みオペレーションを使用すると、データを挿入およびアップサートできます。デフォルトでは、Amazon Timestream for Live Analytics での書き込みは、最初のライターがセマンティクスを獲得した後、データは追加としてのみ保存され、重複レコードは拒否されます。最初のライターがセマンティクスを勝ち取ると、多くの時系列アプリケーションの要件は満たされますが、アプリケーションが既存のレコードを冪等な方法で更新したり、最後のライターがセマンティクスを勝ち取ったりしてデータを書き込む必要があるシナリオがあり、バージョンが最も高いレコードはサービスに保存されます。これらのシナリオに対処するために、Amazon Timestream for Live Analytics ではデータをアップサートできます。Upsert は、レコードが存在しない場合にレコードをシステムに挿入するか、存在する場合にレコードを更新するオペレーションです。レコードが更新されると、冪等な方法で更新されます。

削除するためのレコードレベルのオペレーションはありません。ただし、テーブルとデータベースは削除できます。

メモリストアとマグネティックストアへのデータの書き込み

Amazon Timestream for Live Analytics では、メモリストアとマグネティックストアにデータを直接書き込む機能を提供します。メモリストアは高スループットのデータ書き込みに最適化されており、マグネティックストアは遅延到着データのスループット書き込みを減らすように最適化されています。

遅延受信データとは、現在の時刻より前のタイムスタンプがあり、メモリストアの保持期間外のデータです。テーブルのマグネティックストア書き込みを有効にすることで、遅延受信データをマグネティックストアに書き込む機能を明示的に有効にする必要があります。また、 `MagneticStoreRejectedDataLocation` はテーブルの作成時に定義されます。マグネティックストアに書き込むには、 の呼び出し元がテーブルの作成`MagneticStoreRejectedDataLocation`中に で指定された S3 バケットへのアクセス`S3:PutObject`許可を持っている`WriteRecords`必要があります。詳細については、、CreateTable、WriteRecordsおよび を参照してくださいPutObject。

単一メジャーレコードと複数メジャーレコードを使用したデータの書き込み

Amazon Timestream for Live Analytics では、単一測定レコードと複数測定レコードの 2 種類のレコードを使用してデータを書き込むことができます。

単一測定レコード

単一測定レコードを使用すると、レコードごとに 1 つのメジャーを送信できます。この形式を使用してデータを Timestream for Live Analytics に送信すると、Timestream for Live Analytics はレコードごとに 1 つのテーブル行を作成します。つまり、デバイスが 4 つのメトリクスを出力し、各メトリクスが 1 つのメジャーレコードとして送信されると、Timestream for Live Analytics はテーブルに 4 つの行を作成してこのデータを保存し、デバイス属性は行ごとに繰り返されます。この形式は、アプリケーションから 1 つのメトリクスをモニタリングする場合や、アプリケーションが同時に複数のメトリクスを出力しない場合に推奨されます。

マルチメジャーレコード

マルチメジャーレコードでは、テーブル行ごとに 1 つのメジャーを保存する代わりに、複数のメジャーを 1 つのテーブル行に保存できます。したがって、マルチメジャーレコードを使用すると、最小限の変更で既存のデータをリレーショナルデータベースから Amazon Timestream for Live Analytics に移行できます。

また、単一メジャーレコードよりも多くのデータを単一の書き込みリクエストにバッチ処理することもできます。これにより、データ書き込みスループットとパフォーマンスが向上し、データ書き込みのコストを削減できます。これは、書き込みリクエストでより多くのデータをバッチ処理することで、Amazon Timestream for Live Analytics は 1 回の書き込みリクエスト (該当する場合) でより反復可能なデータを識別し、繰り返しデータに対して 1 回だけ課金できるためです。

トピック

- マルチメジャーレコード
- 過去または未来に存在するタイムスタンプ付きのデータの書き込み

マルチメジャーレコード

マルチメジャーレコードを使用すると、時系列データをメモリとマグネティックストアにコンパクトな形式で保存できるため、データストレージのコストを削減できます。また、コンパクトなデータストレージは、データ取得のためのより簡単なクエリの作成、クエリのパフォーマンスの向上、クエリのコスト削減に役立ちます。

さらに、マルチメジャーレコードは、時系列レコードに複数のタイムスタンプを保存するための TIMESTAMPデータ型もサポートしています。TIMESTAMP マルチメジャーレコードの属性は、将来または過去のタイムスタンプをサポートします。したがって、マルチメジャーレコードは、パフォーマンス、コスト、クエリの簡素化の向上に役立ちます。また、相関するさまざまなタイプのメジャーを保存するための柔軟性も向上します。

利点

マルチメジャーレコードを使用する利点は次のとおりです。

- パフォーマンスとコスト – マルチメジャーレコードを使用すると、1 回の書き込みリクエストで複数の時系列メジャーを書き込むことができます。これにより、書き込みスループットが向上し、書き込みのコストを削減できます。マルチメジャーレコードを使用すると、データをよりコンパクトな方法で保存できるため、データストレージのコストを削減できます。マルチメジャーレコードのコンパクトなデータストレージにより、クエリによって処理されるデータが少なくなります。これは、クエリ全体のパフォーマンスを向上させ、クエリコストを削減するように設計されています。
- クエリのシンプルさ – マルチメジャーレコードでは、同じタイムスタンプで複数のメジャーを読み取るために、クエリに複雑な共通テーブル式 (CTEs) を記述する必要はありません。これは、測定値が 1 つのテーブル行に列として保存されるためです。したがって、マルチメジャーレコードを使用すると、クエリをより簡単に記述できます。
- データモデリングの柔軟性 – TIMESTAMP データ型とマルチメジャーレコードを使用して、将来のタイムスタンプを Timestream for Live Analytics に書き込むことができます。マルチメジャーレコードには、レコードの時間フィールドに加えて、TIMESTAMPデータ型の複数の属性を含めることができます。TIMESTAMP マルチメジャーレコードの 属性は、将来または過去のタイムスタンプを持つことができ、Timestream for Live Analytics がマルチメジャーレコードの タイプの値に対してインデックスを作成しないことを除いてTIMESTAMP、時間フィールドのように動作します。

ユースケース

マルチメジャーレコードは、同じデバイスから一度に複数の測定値を生成する任意の時系列アプリケーションに使用できます。以下はアプリケーション例です。

- 一度に数百のメトリクスを生成するビデオストリーミングプラットフォーム。
- 血中酸素濃度、心拍数、パルスなどの測定値を生成する医療デバイス。
- メトリクス、温度、気象センサーを生成するオイルリグなどの産業機器。
- 1 つ以上のマイクロサービスで設計されているその他のアプリケーション。

例: ビデオストリーミングアプリケーションのパフォーマンスとヘルスのモニタリング

200 EC2インスタンスで実行されているビデオストリーミングアプリケーションを考えてみましょう。Amazon Timestream for Live Analytics を使用してアプリケーションから出力されるメトリクスを保存および分析すると、アプリケーションのパフォーマンスと状態を理解し、異常をすばやく特定し、問題を解決し、最適化の機会を発見できます。

このシナリオを単一測定レコードと複数測定レコードでモデル化し、両方のアプローチを比較/コントラストします。各アプローチについて、次の前提を行います。

- 各EC2インスタンスは、1 秒あたり 4 つのメジャー (video_startup_time、rebuffering_ratio、video_playback_failures、および average_frame_rate) と 4 つのディメンション (device_id、device_type、os_version、および region) を出力します。
- 6 時間のデータをメモリストアに、6 か月のデータをマグネティックストアに格納したいとします。
- 異常を特定するには、過去数分間の異常なアクティビティを特定するために 1 分ごとに実行されるクエリを 10 個設定します。また、過去 6 時間のデータを表示する 8 つのウィジェットを備えたダッシュボードを構築して、アプリケーションを効果的にモニタリングできるようにしました。このダッシュボードには、いつでも 5 人のユーザーがアクセスし、1 時間ごとに自動更新されます。

単一メジャーレコードの使用

データモデリング : 単一メジャーレコードでは、4 つのメジャー (ビデオの起動時間、再バッファリング率、ビデオ再生失敗、平均フレームレート) ごとに 1 つのレコードを作成します。各レコードには、4 つのディメンション (device_id、device_type、os_version、リージョン) とタイムスタンプがあります。

書き込み: Amazon Timestream for Live Analytics にデータを書き込むと、レコードは次のように構築されます。

```
public void writeRecords() {
    System.out.println("Writing records");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();

    final Dimension device_id = new
 Dimension().withName("device_id").withValue("12345678");
    final Dimension device_type = new
 Dimension().withName("device_type").withValue("iPhone 11");
    final Dimension os_version = new
 Dimension().withName("os_version").withValue("14.8");
    final Dimension region = new Dimension().withName("region").withValue("us-east-1");

    dimensions.add(device_id);
```

```
    dimensions.add(device_type);
    dimensions.add(os_version);
    dimensions.add(region);

    Record videoStartupTime = new Record()
        .withDimensions(dimensions)
        .withMeasureName("video_startup_time")
        .withMeasureValue("200")
        .withMeasureValueType(MeasureValueType.BIGINT)
        .withTime(String.valueOf(time));
    Record rebufferingRatio = new Record()
        .withDimensions(dimensions)
        .withMeasureName("rebuffering_ratio")
        .withMeasureValue("0.5")
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time));
    Record videoPlaybackFailures = new Record()
        .withDimensions(dimensions)
        .withMeasureName("video_playback_failures")
        .withMeasureValue("0")
        .withMeasureValueType(MeasureValueType.BIGINT)
        .withTime(String.valueOf(time));
    Record averageFrameRate = new Record()
        .withDimensions(dimensions)
        .withMeasureName("average_frame_rate")
        .withMeasureValue("0.5")
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time));

    records.add(videoStartupTime);
    records.add(rebufferingRatio);
    records.add(videoPlaybackFailures);
    records.add(averageFrameRate);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withRecords(records);

    try {
      WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
      System.out.println("WriteRecords Status: " +
writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
```

```
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.getRejectedRecords()) {
        System.out.println("Rejected Index " + rejectedRecord.getRecordIndex() + ": "
            + rejectedRecord.getReason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

単一測定レコードを保存すると、データは次のように論理的に表されます。

時間	device_id	device_type	os_version	region	measure_name	measure_value::bigint	measure_value::double
2021-09-07 21:48:44.00	12345678	iPhone 11	14.8	us-east-1	video_startup_time	200	
2021-09-07 21:48:44.00	12345678	iPhone 11	14.8	us-east-1	rebuffering_ratio		0.5
2021-09-07 21:48:44.00	12345678	iPhone 11	14.8	us-east-1	video_playback_failures	0	
2021-09-07 21:48:44.00	12345678	iPhone 11	14.8	us-east-1	平均フレームレート		0.85
2021-09-07	12345678	iPhone 11	14.8	us-east-1	video_startup_time	500	

時間	device_id	device_type	os_version	region	measure_name	measure_value::bigint	measure_value::double
21:53:44 .00							
2021-09-07 21:53:44 .00	12345678	iPhone 11	14.8	us-east-1	rebuffering_ratio		1.5
2021-09-07 21:53:44 .00	12345678	iPhone 11	14.8	us-east-1	video_playback_failures	10	
2021-09-07 21:53:44 .00	12345678	iPhone 11	14.8	us-east-1	average_frame_rate		0.2

クエリ : 過去 15 分間に受信したのと同じタイムスタンプを持つすべてのデータポイントを取得するクエリを次のように記述できます。

```
with cte_video_startup_time as ( SELECT time, device_id, device_type, os_version,
 region, measure_value::bigint as video_startup_time FROM table where time >= ago(15m)
 and measure_name=”video_startup_time”),
cte_rebuffering_ratio as ( SELECT time, device_id, device_type, os_version, region,
 measure_value::double as rebuffering_ratio FROM table where time >= ago(15m) and
 measure_name=”rebuffering_ratio”),
cte_video_playback_failures as ( SELECT time, device_id, device_type, os_version,
 region, measure_value::bigint as video_playback_failures FROM table where time >=
 ago(15m) and measure_name=”video_playback_failures”),
cte_average_frame_rate as ( SELECT time, device_id, device_type, os_version, region,
 measure_value::double as average_frame_rate FROM table where time >= ago(15m) and
 measure_name=”average_frame_rate”)
SELECT a.time, a.device_id, a.os_version, a.region, a.video_startup_time,
 b.rebuffering_ratio, c.video_playback_failures, d.average_frame_rate FROM
```

```
 cte_video_startup_time a, cte_buffering_ratio b, cte_video_playback_failures c,
 cte_average_frame_rate d WHERE
a.time = b.time AND a.device_id = b.device_id AND a.os_version = b.os_version AND
 a.region=b.region AND
a.time = c.time AND a.device_id = c.device_id AND a.os_version = c.os_version AND
 a.region=c.region AND
a.time = d.time AND a.device_id = d.device_id AND a.os_version = d.os_version AND
 a.region=d.region
```

ワークロードコスト : このワークロードのコストは、単一メジャーレコードでは 1 か月あたり 373.23 USD と推定されます。

マルチメジャーレコードの使用

データモデリング : マルチメジャーレコードでは、4 つのすべてのメジャー (ビデオの起動時間、再バッファリング率、ビデオ再生の失敗、平均フレームレート) 、4 つのすべてのディメンション (device_id、device_type、os_version、リージョン) 、タイムスタンプを含む 1 つのレコードを作成します。

書き込み： Amazon Timestream for Live Analytics にデータを書き込むと、レコードは次のように構築されます。

```
public void writeRecords() {
    System.out.println("Writing records");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();

    final Dimension device_id = new
 Dimension().withName("device_id").withValue("12345678");
    final Dimension device_type = new
 Dimension().withName("device_type").withValue("iPhone 11");
    final Dimension os_version = new
 Dimension().withName("os_version").withValue("14.8");
    final Dimension region = new Dimension().withName("region").withValue("us-east-1");

    dimensions.add(device_id);
    dimensions.add(device_type);
    dimensions.add(os_version);
    dimensions.add(region);
```

```
    Record videoMetrics = new Record()
        .withDimensions(dimensions)
        .withMeasureName("video_metrics")
        .withTime(String.valueOf(time));
        .withMeasureValueType(MeasureValueType.MULTI)
        .withMeasureValues(
          new MeasureValue()
         .withName("video_startup_time")
         .withValue("0")
         .withValueType(MeasureValueType.BIGINT),
         new MeasureValue()
  .withName("rebuffering_ratio")
         .withValue("0.5")
         .withType(MeasureValueType.DOUBLE),
          new MeasureValue()
         .withName("video_playback_failures")
         .withValue("0")
         .withValueType(MeasureValueType.BIGINT),
   new MeasureValue()
          .withName("average_frame_rate")
         .withValue("0.5")
         .withValueType(MeasureValueType.DOUBLE))

    records.add(videoMetrics);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withRecords(records);

    try {
      WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
      System.out.println("WriteRecords Status: " +
writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.getRejectedRecords()) {
        System.out.println("Rejected Index " + rejectedRecord.getRecordIndex() + ": "
            + rejectedRecord.getReason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
      System.out.println("Error: " + e);
```

```
    }
  }
```

マルチメジャーレコードを保存すると、データは次のように論理的に表されます。

時間	device_id	device_type	os_version	region	measure_name	video_startup_time	rebuffering_ratio	video_playback_failures	average_frame_rate
2021-09-7 21:48:440	1234567	iPhone 11	14.8	us-east-1	video_metrics	200	0.5	0	0.85
2021-09-7 21:53:440	1234567	iPhone 11	14.8	us-east-1	video_metrics	500	1.5	10	0.2

クエリ : 過去 15 分間に受信したのと同じタイムスタンプを持つすべてのデータポイントを取得するクエリを次のように記述できます。

```
SELECT time, device_id, device_type, os_version, region, video_startup_time,
 rebuffering_ratio, video_playback_failures, average_frame_rate FROM table where time
 >= ago(15m)
```

ワークロードコスト : ワークロードのコストは、マルチメジャーレコードでは 127.43 ドルと推定されます。

> **Note**
>
> この場合、マルチメジャーレコードを使用すると、全体的な推定月額支出が 2.5 倍削減され、データ書き込みコストが 3.3 倍、ストレージコストが 3.3 倍、クエリコストが 1.2 倍削減されます。

過去または未来に存在するタイムスタンプ付きのデータの書き込み

Timestream for Live Analytics では、いくつかの異なるメカニズムを使用して、メモリストアの保持期間外にあるタイムスタンプを使用してデータを書き込むことができます。

- マグネティックストアの書き込み – 遅延到着データをマグネティックストアの書き込みを通じてマグネティックストアに直接書き込むことができます。マグネティックストア書き込みを使用するには、まずテーブルのマグネティックストア書き込みを有効にする必要があります。その後、メモリストアへのデータの書き込みに使用されるのと同じメカニズムを使用して、テーブルにデータを取り込むことができます。Amazon Timestream for Live Analytics は、タイムスタンプに基づいてデータを自動的にマグネティックストアに書き込みます。

  > **Note**
  >
  > 磁気ストアのレイテンシーは、 write-to-readレイテンシーが 1 write-to-read秒未満の範囲であるメモリストアにデータを書き込む場合とは異なり、最大 6 時間です。

- TIMESTAMP 測定用のデータ型 – TIMESTAMP データ型を使用して、過去、現在、または将来のデータを保存できます。マルチメジャーレコードには、レコードの時間フィールドに加えて、TIMESTAMPデータ型の複数の属性を含めることができます。TIMESTAMP マルチメジャーレコードの 属性は、将来または過去のタイムスタンプを持つことができ、Timestream for Live Analytics がマルチメジャーレコードの タイプの値に対してインデックスを作成しないことを除いてTIMESTAMP、時間フィールドのように動作します。

  > **Note**
  >
  > TIMESTAMP データ型は、マルチメジャーレコードでのみサポートされています。

読み取りの最終的な整合性

Timestream for Live Analytics は、読み取りの最終的な整合性セマンティクスをサポートします。つまり、データのバッチを Timestream for Live Analytics に書き込んだ直後にデータをクエリすると、クエリ結果は最近完了した書き込みオペレーションの結果を反映しない可能性があります。これらのクエリリクエストを短時間後に繰り返すと、結果は最新のデータを返します。

による書き込みのバッチ処理 WriteRecords API

Amazon Timestream for Live Analytics を使用すると、単一の時系列のデータポイントや、複数の系列のデータポイントを単一の書き込みリクエストに書き込むことができます。1 回の書き込みオペレーションで複数のデータポイントをバッチ処理することは、パフォーマンスとコストの観点から有益です。詳細については書き込み、「計測と料金」セクションの「」を参照してください。

> **Note**
>
> Timestream for Live Analytics への書き込みリクエストは、Timestream for Live Analytics がアプリケーションのデータ取り込みニーズに適応するようにスケールすると、スロットリングされる場合があります。アプリケーションでスロットリングの例外が発生した場合は、引き続き同じ (またはそれ以上の) スループットでデータを送信して、Timestream for Live Analytics がアプリケーションのニーズに自動的にスケーリングできるようにする必要があります。

バッチロード

の Amazon Timestream のバッチロードでは LiveAnalytics、Amazon S3 に保存されているCSVファイルをバッチで Timestream に取り込むことができます。 Amazon S3 この新しい機能により、他のツールに頼ったり、カスタムコードを記述 LiveAnalytics したりすることなく、 の Timestream にデータを取得できます。バッチロードを使用して、クエリや分析にすぐに必要ではないデータなど、柔軟な待機時間でデータをバックフィルできます。

バッチロードタスクは、 AWS Management Console、、 AWS CLIおよび AWS を使用して作成できますSDKs。詳細については、コンソールでのバッチロードの使用、でのバッチロードの使用 AWS CLI、およびでのバッチロードの使用 AWS SDKsを参照してください。

バッチロードの詳細については、「」を参照してくださいの Timestream でのバッチロードの使用 LiveAnalytics。

オペレーションとバッチロードの選択 WriteRecords API

WriteRecords API オペレーションを使用すると、ストリーミング時系列データをシステムによって生成された LiveAnalytics の Timestream に書き込むことができます。を使用すると WriteRecords、1 つのデータポイントまたはより小さなデータバッチをリアルタイムで継続的に取り込むことができます。Timestream for LiveAnalytics は、データベースへの書き込みを呼び出すと

きに指定したデータポイントのディメンション名とデータ型に基づいて、 LiveAnalytics テーブルの Timestream の列名とデータ型を自動的に検出する柔軟なスキーマを提供します。

対照的に、バッチロードを使用すると、定義したデータモデルを使用して LiveAnalytics、ソースファイル (CSV ファイル) から Timestream for へのバッチ時系列データの堅牢な取り込みが可能になります。ソースファイルでバッチロードを使用する場合の例には、概念実証 LiveAnalytics による Timestream の評価のために時系列データを一括インポートする、しばらくオフラインだった IoT デバイスから時系列データを一括インポートする、Amazon S3 から の Timestream に履歴時系列データを移行するなどがあります LiveAnalytics。バッチロードの詳細については、「」を参照してくださいの Timestream でのバッチロードの使用 LiveAnalytics。

どちらのソリューションも安全で信頼性が高く、パフォーマンスに優れています。

次の WriteRecords 場合に を使用します。

- リクエストあたりの少量 (10 MB 未満) のデータをストリーミングします。
- 既存のテーブルの入力。
- ログストリームからのデータの取り込み。
- リアルタイム分析の実行。
- 低レイテンシーが必要です。

バッチロードは、次の場合に使用します。

- CSV ファイルに Amazon S3 で発生するデータのより大きなロードを取り込む。制限事項の詳細については、「クォータ」を参照してください。
- データ移行などの新しいテーブルの入力。
- 履歴データ (新しいテーブルへの取り込み) でデータベースを強化します。
- ソースデータの変更が遅いかまったくない。
- 特に大量のデータをロードする場合、バッチロードタスクはリソースが使用可能になるまで保留状態になる可能性があるため、柔軟な待機時間があります。バッチロードは、より明確にするためにクエリや分析にすぐに利用できる必要がないデータに適しています。

ストレージ

Timestream for Live Analytics は、時系列データを保存および整理して、クエリ処理時間を最適化し、ストレージコストを削減します。データストレージ階層化を提供し、メモリストアとマグネ

ティックストアの 2 つのストレージ階層をサポートします。メモリストアは、高スループットのデータ書き込みと高速 point-in-timeクエリ用に最適化されています。マグネティックストアは、スループットの遅延受信データ書き込み、長期データストレージ、高速分析クエリ用に最適化されています。

Timestream for Live Analytics は、1 つの 内の異なるアベイラビリティーゾーンにメモリとマグネティックストアのデータを自動的にレプリケートすることで、データの耐久性を確保します AWS リージョン。すべてのデータは、書き込みリクエストの完了を確認する前にディスクに書き込まれます。

Timestream for Live Analytics では、メモリストアからマグネティックストアにデータを移動するように保持ポリシーを設定できます。データが設定値に達すると、Timestream for Live Analytics は自動的にデータをマグネティックストアに移動します。マグネティックストアに保持値を設定することもできます。データがマグネティックストアから期限切れになると、データは完全に削除されます。

例えば、1 週間分のデータを保持するようにメモリストアを設定し、1 年分のデータを保持するようにマグネティックストアを設定するシナリオを考えてみましょう。データの経過時間は、データポイントに関連付けられたタイムスタンプを使用して計算されます。メモリストア内のデータが 1 週間経過すると、自動的にマグネティックストアに移動します。その後、マグネティックストアに 1 年間保存されます。データが 1 年経過すると、Timestream for Live Analytics から削除されます。メモリストアとマグネティックストアの保持値は、データが Timestream for Live Analytics に保存される時間を累積的に定義します。つまり、上記のシナリオでは、データ到着時から、データは Timestream for Live Analytics に合計 1 年 1 週間保存されます。

> **Note**
>
> メモリまたはマグネティックストアの保持期間をアップグレードすると、その時点から保持の変更が有効になります。例えば、メモリストアの保持期間を 2 時間に設定し、テーブル保持ポリシーを更新して 24 時間に変更した場合、メモリストアは 24 時間のデータを保持できますが、この変更が行われた 22 時間後に 24 時間のデータが入力されます。Timestream for Live Analytics は、メモリストアに入力するために磁気ストアからデータを取得しません。

時系列データのセキュリティを確保するために、Timestream for Live Analytics のデータはデフォルトで常に暗号化されます。これは、転送中および保管中のデータに適用されます。さらに、Timestream for Live Analytics を使用すると、カスタマーマネージドキーを使用して、マグネティックストア内のデータを保護できます。カスタマーマネージドキーの詳細については、「」を参照してくださいAWS KMS keys。

クエリ

Timestream for Live Analytics を使用すると、 のメトリクス DevOps、IoT アプリケーションのセンサーデータ、機器のメンテナンス用の産業用テレメトリデータ、およびその他の多くのユースケースを簡単に保存および分析できます。Timestream for Live Analytics の専用アダプティブクエリエンジンを使用すると、1 つのSQLステートメントを使用してストレージ階層間でデータにアクセスできます。データの場所を指定しなくても、ストレージ階層間で透過的にデータにアクセスして結合します。SQL を使用して Timestream for Live Analytics でデータをクエリし、1 つ以上のテーブルから時系列データを取得できます。データベースとテーブルのメタデータ情報にアクセスできます。Timestream for Live Analytics は、時系列分析用の組み込み関数SQLもサポートしています。詳細については、 クエリ言語リファレンス リファレンスを参照してください。

Timestream for Live Analytics は、各コンポーネントが他のコンポーネントとは独立してスケールできる (アプリケーションのニーズに応じて実質的に無限のスケールを提供できるようにする) 完全に分離されたデータ取り込み、ストレージ、クエリアーキテクチャを持つように設計されています。つまり、アプリケーションが 1 日あたり数百テラバイトのデータを送信したり、少量または大量のデータを処理する数百万のクエリを実行したりしても、Timestream for Live Analytics は「チップオーバー」しません。データが時間の経過とともに増加するにつれて、Timestream for Live Analytics のクエリレイテンシーはほぼ変化しません。これは、Timestream for Live Analytics クエリアーキテクチャが大量の並列処理を活用してより大きなデータボリュームを処理し、アプリケーションのクエリスループットのニーズに合わせて自動的にスケーリングできるためです。

データモデル

Timestream は、フラットモデルと時系列モデルの 2 つのクエリデータモデルをサポートしています。

> **Note**
>
> Timestream のデータはフラットモデルを使用して保存され、データをクエリするためのデフォルトモデルです。時系列モデルはクエリ時間の概念であり、時系列分析に使用されます。

- フラットモデル
- 時系列モデル

フラットモデル

フラットモデルは、クエリ用の Timestream のデフォルトデータモデルです。これは時系列データを表形式で表します。ディメンション名、時間、メジャー名、メジャー値は列として表示されます。テーブルの各行は、時系列内の特定の時刻の測定に対応する原子データポイントです。Timestream データベース、テーブル、列には、いくつかの命名制約があります。これらについては、「」で説明されていますサービス制限。

次の表は、データが単一測定レコードとして送信されるときに、Timestream がEC2インスタンスのCPU使用率、メモリ使用率、ネットワークアクティビティを表すデータを保存する方法の例を示しています。この場合、ディメンションは、リージョン、アベイラビリティーゾーン、仮想プライベートクラウド、インスタンスIDsのインスタンスですEC2。測定値は、EC2インスタンスのCPU使用率、メモリ使用率、受信ネットワークデータです。列リージョン、az、vpc、および instance_id にはディメンション値が含まれます。列の時刻には、各レコードのタイムスタンプが含まれます。measure_name 列には、cpu-utilization、memory_utilization、network_bytes_in で表されるメジャーの名前が含まれます。列 measure_value::double には、2 倍として出力される測定値 (CPU使用率やメモリ使用率など) が含まれます。列 measure_value::bigint には、受信ネットワークデータなどの整数として出力される測定値が含まれます。

時間	region	az	vpc	instance_id	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:00.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	cpu_utilization	35.0	null
2019-12-04 19:00:01.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	cpu_utilization	38.2	null
2019-12-04 19:00:02.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	cpu_utilization	45.3	null

時間	region	az	vpc	instance_id	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:00.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	memory_utilization	54.9	null
2019-12-04 19:00:01.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	memory_utilization	42.6	null
2019-12-04 19:00:02.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	memory_utilization	33.3	null
2019-12-04 19:00:00.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	network_bytes	34,400	null
2019-12-04 19:00:01.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	network_bytes	1,500	null
2019-12-04 19:00:02.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	network_bytes	6,000	null

次の表は、データがマルチメジャーレコードとして送信されるときに、Timestream がEC2インスタンスのCPU使用率、メモリ使用率、ネットワークアクティビティを表すデータを保存する方法の例を示しています。

時間	region	az	vpc	instance_id	measure_name	cpu_utilization	memory_utilization	network_bytes
2019-12-04 19:00:00.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcde0	メトリクス	35.0	54.9	34,400
2019-12-04 19:00:01.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcde0	メトリクス	38.2	42.6	1,500
2019-12-04 19:00:02.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcde0	メトリクス	45.3	33.3	6,600

時系列モデル

時系列モデルは、時系列分析に使用されるクエリ時間構造です。これは、(時間、測定値) ペアの順序付けられたシーケンスとしてデータを表します。Timestream は、データのギャップを埋めることができるように補間などの時系列関数をサポートしています。これらの関数を使用するには、create_time_series などの関数を使用してデータを時系列モデルに変換する必要があります。詳細については、クエリ言語リファレンス「」を参照してください。

EC2 インスタンスの前の例を使用して、ここに時系列で表されるCPU使用率データを示します。

region	az	vpc	instance_id	cpu_utilization
us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	[{time: 2019-12-04 19:00:00.000000000、値: 35}、{time: 2019-12-04 19:00:01.000000000、

region	az	vpc	instance_id	cpu_utilization
				値: 38.2}、{time: 2019-12-04 19:00:02.000000000、値: 45.3}]

スケジュールされたクエリ

Amazon Timestream for Live Analytics のスケジュールされたクエリ機能は、運用ダッシュボード、ビジネスレポート、アドホック分析、その他のアプリケーションに通常使用される、集計、ロールアップ、その他の形式の前処理済みデータを計算して保存するための、フルマネージド型のサーバーレスでスケーラブルなソリューションです。スケジュールされたクエリにより、リアルタイム分析のパフォーマンスとコスト効率が向上し、データから追加のインサイトを導き出し、より良いビジネス上の意思決定を引き続き行うことができます。

スケジュールされたクエリの詳細については、「」を参照してくださいの Timestream でスケジュールされたクエリを使用する LiveAnalytics。

Timestream コンピューティングユニット (TCU)

Amazon Timestream for Live Analytics は、Timestream コンピューティングユニット () のクエリニーズに割り当てられたコンピューティングキャパシティを測定しますTCU。1 つは 4 GB vCPUs と 16 GB のメモリTCUで構成されます。Timestream for Live Analytics でクエリを実行すると、サービスはクエリの複雑さと処理されるデータ量に基づいてTCUsオンデマンドを割り当てます。クエリが消費TCUsする の数によって、関連するコストが決まります。

> **Note**
>
> 2024 年 4 月 29 日以降にサービスにオンボード AWS アカウント されるすべての は、デフォルトでクエリ料金TCUsに を使用します。

このトピックの内容

- MaxQuery TCU
- TCU の請求

- の設定 TCU
- 必要なコンピューティングユニットの見積もり
- いつ増やすか MaxQueryTCU
- いつ減らすか MaxQueryTCU
- CloudWatch メトリクスを使用した使用状況のモニタリング
- コンピューティングユニットの使用状況のバリエーションを理解する

MaxQuery TCU

この設定は、サービスがクエリを処理するために任意の時点で使用するコンピューティングユニットの最大数を指定します。クエリを実行するには、最小容量を 4 に設定する必要がありますTCUs。の最大数は、4、8、16、32 などの 4 倍数TCUsで設定できます。ワークロードに使用するコンピューティングリソースに対してのみ課金されます。例えば、最大値を 128 TCUs に設定しても、一貫して使用する は 8 のみですTCUs。8 を使用した期間についてのみ課金されますTCUs。`MaxQueryTCU` アカウントのデフォルトは 200 に設定されています。または UpdateAccountSettingsAPIオペレーションを AWS Management Console または で AWS SDK使用して、4 `MaxQueryTCU`から 1000 に調整できます AWS CLI。

`MaxQueryTCU` アカウントの を設定することをお勧めします。最大TCU制限を設定すると、サービスがクエリワークロードに使用できるコンピューティングユニットの数を制限することで、コストを制御できます。これにより、クエリの支出をより正確に予測および管理できます。

TCU の請求

各 TCUは、1 秒あたりの粒度で、最低 30 秒間、時間単位で請求されます。これらのコンピューティングユニットの使用単位は TCU- 時間です。

クエリを実行すると、クエリの実行時にTCUs使用された に対して課金され、 TCU時間単位で測定されます。例:

- ワークロードは 20 TCUsを 3 時間使用します。60 TCU時間 (20 TCUs x 3 時間) の料金が請求されます。
- ワークロードは 30 分間TCUsに 10 を使用し、次の 30 分間TCUsに 20 を使用します。15 TCU時間 (10 TCUs x 0.5 時間 + 20 TCUs x 0.5 時間) の料金が請求されます。

TCU時間あたりの料金は、 によって異なります AWS リージョン。詳細については、Amazon Timestream の料金を参照してください。ワークロードが増加すると、サービスは、一貫したパ

フォーマンスを維持するために、指定された最大TCU制限 (MaxQueryTCU) までコンピューティング容量を自動的にスケーリングします。MaxQueryTCU この設定は、サービスがスケーリングできるコンピューティング容量の上限として機能します。この設定は、コンピューティングリソースの数とその結果のコストを制御するのに役立ちます。

の設定 TCU

サービスのオンボード時、それぞれデフォルトのMaxQueryTCU制限 AWS アカウント は 200 です。必要に応じて、 AWS Management Console または UpdateAccountSettingsAPIオペレーションを または で AWS SDK使用して、この制限を更新できます AWS CLI。

設定する値に不明な点がある場合は、アカウントのQueryTCUメトリクスをモニタリングします。このメトリクスは、 AWS Management Console および Amazon で使用できます CloudWatch。このメトリクスは、1 分間の粒度TCUsで使用される最大数に関するインサイトを提供します。履歴データと将来の成長の推定に基づいて、使用量の急増に対応するMaxQueryTCUように を設定します。ピーク使用量TCUsよりも 4 ~ 16 以上のヘッドルームを用意することをお勧めします。例えば、QueryTCU過去 30 日間のピークが 128 の場合は、132 MaxQueryTCU ~ 144 に設定することをお勧めします。

必要なコンピューティングユニットの見積もり

コンピューティングユニットはクエリを同時に処理できます。必要なコンピューティングユニットの数を決定するには、次の表の一般的なガイドラインを考慮してください。

同時クエリ	TCUs
7	4
14	8
21	12

> **Note**
>
> - これらは一般的なガイドラインであり、必要なコンピューティングユニットの実際の数は、次のようないくつかの要因によって異なります。
>   - クエリの有効な同時実行数。

  - クエリパターン。
  - スキャンされたパーティションの数。
  - その他のワークロード固有の特性。
- このガイドラインは、過去数分間から 1 時間のデータをスキャンし、Timestream クエリのベストプラクティスとデータモデリングガイドライン に準拠するクエリに関するものです。 ???
- 必要に応じて、アプリケーションのパフォーマンスとQueryTCUメトリクスをモニタリングしてコンピューティングユニットを調整します。

いつ増やすか MaxQueryTCU

次のシナリオMaxQueryTCUでは、 を増やすことを検討する必要があります。

- クエリのピーク消費量が、現在設定されている最大クエリ に近づいているか、到達しています TCU。最大クエリは、ピーク消費量よりも TCU 4 ~ 16 以上TCUs高く設定することをお勧めします。
- クエリは、メッセージ MaxQueryTCUを超えた 4xx エラーを返しています。ワークロードの計画的な増加が予想される場合は、それTCUに応じて設定された最大クエリを再検討して調整します。

いつ減らすか MaxQueryTCU

以下のシナリオMaxQueryTCUでは、 を減らすことを検討する必要があります。

- ワークロードには予測可能で安定した使用パターンがあり、コンピューティングの使用要件をよく理解しています。最大クエリをピーク消費量TCUの 4 ~ 16 TCU以内に減らすと、意図しない使用量とコストを防ぐことができます。UpdateAccountSettings API オペレーションを使用して値を変更できます。
- アプリケーションまたはユーザーの動作パターンの変化により、ワークロードのピーク使用量は時間の経過とともに減少しています。最大値を下げるTCUと、意図しないコストを削減できます。

> **Note**
>
> 現在の使用状況によっては、最大TCU制限の変更を減らすのに最大 24 時間かかる場合があります。クエリが実際に消費TCUsする に対してのみ請求されます。最大クエリTCU制限を高くしても、ワークロードで使用されていない限りTCUs、コストには影響しません。

CloudWatch メトリクスを使用した使用状況のモニタリング

TCU 使用状況をモニタリングするために、Timestream for Live Analytics には という CloudWatch メトリクスが用意されていますQueryTCU。このメトリクスは、1 分間に使用されるコンピューティングユニットの数を指定し、1 分ごとに出力されます。1 分間に最大および最小TCUs使用量をモニタリングすることを選択できます。このメトリクスにアラームを設定して、クエリコストをリアルタイムで追跡することもできます。

コンピューティングユニットの使用状況のバリエーションを理解する

クエリに必要なコンピューティングリソースの数は、いくつかのパラメータに基づいて増減できます。例えば、リアルタイムクエリと分析クエリを使用するデータボリューム、データ取り込みパターン、クエリレイテンシー、クエリ形状、クエリ効率、クエリの組み合わせなどです。これらのパラメータは、ワークロードに必要なTCUユニット数の増加または減少につながる可能性があります。これらのパラメータが変更されない定常状態では、ワークロードに必要なコンピューティングユニットの数が減少することがあります。その結果、月額コストを削減できます。

さらに、ワークロードまたはデータ内のこれらのパラメータのいずれかが変更されると、必要なコンピューティングユニットの数が増加する可能性があります。Timestream がクエリを受信すると、クエリがアクセスするデータパーティションに応じて、Timestream はクエリに適切に対処するコンピューティングリソースの数を決定します。

Timestream は、取り込みとクエリのアクセスパターンに基づいて定期的にデータレイアウトを最適化します。Timestream は、アクセス頻度の低いパーティションを 1 つのパーティションにまとめるか、ホットパーティションを複数のパーティションに分割してパフォーマンスを向上させることで、最適化を実行します。したがって、同じクエリで使用されるコンピューティング容量は、異なる時点でわずかに異なる場合があります。

> **クエリのTCU料金を使用するためのオプトイン**
>
> 既存のユーザーとして、1 回限りのオプトインを実行して、よりTCUs適切なコスト管理と、計測されたクエリあたりの最小バイト数の削除を行うことができます。 AWS Management

Console または UpdateAccountSettingsAPIオペレーションを使用して、 または で AWS SDKオプトインできます AWS CLI。API オペレーションで、 QueryPricingModelパラメータを に設定しますCOMPUTE_UNITS。
コンピューティングベースの料金モデルへのオプトインは、取り返しのつかない変更です。

の Timestream へのアクセス LiveAnalytics

コンソールCLIまたは LiveAnalytics を使用して Timestream にアクセスできますAPI。の Timestream へのアクセスについては LiveAnalytics、以下を参照してください。

トピック

- にサインアップする AWS アカウント
- 管理アクセスを持つユーザーを作成する
- LiveAnalytics アクセスに Timestream を提供する
- プログラマチックアクセス権を付与する

にサインアップする AWS アカウント

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

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

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

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

AWS は、サインアッププロセスが完了した後に確認 E メールを送信します。/ に移動し、マイアカウント を選択して、いつでも現在のアカウントアクティビティを表示https://aws.amazon.comし、アカウントを管理できます。

管理アクセスを持つユーザーを作成する

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

のセキュリティ保護 AWS アカウントのルートユーザー

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

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

   手順については、「 ユーザーガイド」の AWS アカウント 「ルートユーザー (コンソール) の仮想MFAデバイスの有効化」を参照してください。 IAM

管理アクセスを持つユーザーを作成する

1. IAM Identity Center を有効にします。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Centerの有効化」を参照してください。
2. IAM Identity Center で、ユーザーに管理アクセスを許可します。

   をアイデンティティソース IAM アイデンティティセンターディレクトリ として使用する方法のチュートリアルについては、 AWS IAM Identity Center ユーザーガイドの「デフォルトを使用してユーザーアクセスを設定する IAM アイデンティティセンターディレクトリ」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

- IAM Identity Center ユーザーでサインインするには、IAMIdentity Center ユーザーの作成時に E メールアドレスにURL送信されたサインインを使用します。

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

追加のユーザーにアクセス権を割り当てる

1. IAM Identity Center で、最小権限のアクセス許可を適用するベストプラクティスに従うアクセス許可セットを作成します。

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

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

LiveAnalytics アクセスに Timestream を提供する

の Timestream にアクセスするために必要なアクセス許可 LiveAnalytics は、管理者に既に付与されています。他のユーザーには、次のポリシーを使用して Timestream LiveAnalytics にアクセスを許可する必要があります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "timestream:*",
        "kms:DescribeKey",
        "kms:CreateGrant",
        "kms:Decrypt",
        "dbqms:CreateFavoriteQuery",
        "dbqms:DescribeFavoriteQueries",
        "dbqms:UpdateFavoriteQuery",
        "dbqms:DeleteFavoriteQueries",
        "dbqms:GetQueryString",
        "dbqms:CreateQueryHistory",
        "dbqms:UpdateQueryHistory",
        "dbqms:DeleteQueryHistory",
        "dbqms:DescribeQueryHistory",
        "s3:ListAllMyBuckets"
      ],
      "Resource": "*"
```

```
    }
  ]
}
```

> **Note**
>
> の詳細についてはdbqms、「 Database Query Metadata Service のアクション、リソース、および条件キー」を参照してください。詳細については、AWS「 Key Management Service のアクション、リソース、および条件キーkms」を参照してください。

プログラマチックアクセス権を付与する

ユーザーが の AWS 外部とやり取りする場合は、プログラムによるアクセスが必要です AWS Management Console。プログラムによるアクセスを許可する方法は、 にアクセスするユーザーのタイプによって異なります AWS。

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

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

プログラマチックアクセス権を必要とするユーザー	目的	方法
		を参照してください。 AWS SDKs
IAM	一時的な認証情報を使用して AWS CLI、 AWS SDKs、または へのプログラムによるリクエストに署名します AWS APIs。	IAM 「 ユーザーガイド」の「AWS リソースで一時的な認証情報を使用する」の手順に従います。
IAM	(非推奨) 長期認証情報を使用して、AWS CLI、 AWS SDKsまたは へのプログラムによるリクエストに署名します AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、AWS Command Line Interface 「 ユーザーガイド」の IAM「ユーザー認証情報を使用した認証」を参照してください。 • および ツールについては AWS SDKs、AWS SDKs「 ツールリファレンスガイド」の「長期認証情報を使用した認証」を参照してください。 • については AWS APIs、「 ユーザーガイド」の IAM「ユーザーのアクセスキーの管理」を参照してください。 IAM

コンソールを使用する場合

Timestream Live Analytics の AWS マネジメントコンソールを使用して、データベースとテーブルを作成、編集、削除、説明、一覧表示できます。コンソールを使用してクエリを実行することもできます。

トピック

チュートリアル

このチュートリアルでは、サンプルデータセットが入力されたデータベースを作成し、サンプルクエリを実行する方法について説明します。このチュートリアルで使用されるサンプルデータセットは、IoT や DevOps シナリオで頻繁に見られます。IoT データセットには、トラックの速度、場所、負荷などの時系列データが含まれており、フリート管理を合理化し、最適化の機会を特定します。 DevOps データセットには、アプリケーションのパフォーマンスと可用性を向上させるためにCPU、、ネットワーク、メモリ使用率などのEC2インスタンスメトリクスが含まれています。このセクションで説明する手順のビデオチュートリアルを以下に示します。

以下の手順に従って、サンプルデータセットが入力されたデータベースを作成し、 AWS コンソールを使用してサンプルクエリを実行します。

1. AWS コンソール を開きます。
2. ナビゲーションペインで、データベースを選択します。
3. データベースの作成 をクリックします。

4. データベースの作成ページで、次のように入力します。

   - 設定の選択 — サンプルデータベース を選択します。
   - 名前 — 選択したデータベース名を入力します。
   - サンプルデータセットの選択 — IoT と を選択しますDevOps。
   - データベースの作成をクリックして、IoT とサンプルデータ DevOps の入力という 2 つのテーブルを含むデータベースを作成します。
5. ナビゲーションペインで、クエリエディタを選択します。
6. トップメニューからサンプルクエリを選択します。
7. サンプルクエリのいずれかをクリックします。これにより、サンプルクエリが入力されたエディタでクエリエディタに戻ります。
8. Run をクリックしてクエリを実行し、クエリ結果を表示します。

データベースを作成する

AWS コンソールを使用してデータベースを作成するには、次の手順に従います。

1. AWS コンソール を開きます。
2. ナビゲーションペインで、データベースを選択します。
3. データベースの作成 をクリックします。
4. データベースの作成ページで、次のように入力します。

   - 設定の選択 — 標準データベース を選択します。
   - 名前 — 選択したデータベース名を入力します。
   - 暗号化 — KMSキーを選択するか、デフォルトのオプションを使用します。このオプションでは、Timestream Live Analytics がキーをアカウント内に作成していない場合は、そのKMSキーを作成します。
5. データベースを作成するには、データベースの作成をクリックします。

テーブルを作成する

AWS コンソールを使用してテーブルを作成するには、次の手順に従います。

1. AWS コンソール を開きます。
2. ナビゲーションペインで、テーブルを選択します。

3. テーブルの作成 をクリックします。
4. テーブルの作成ページで、次のように入力します。
   - データベース名 — で作成されたデータベースの名前を選択します データベースを作成する。
   - テーブル名 — 任意のテーブル名を入力します。
   - メモリストアの保持 — メモリストアにデータを保持する期間を指定します。メモリストアは、到着遅延データ (タイムスタンプが現在の時刻より前のデータ) を含む受信データを処理し、高速 point-in-timeクエリ用に最適化されています。
   - マグネティックストアの保持 — マグネティックストアにデータを保持する期間を指定します。マグネティックストアは長期ストレージ用であり、高速分析クエリ用に最適化されています。
5. テーブルの作成 をクリックします。

クエリを実行する

AWS コンソールを使用してクエリを実行するには、次の手順に従います。

1. AWS コンソール を開きます。
2. ナビゲーションペインで、クエリエディタを選択します。
3. 左側のペインで、 で作成されたデータベースを選択します データベースを作成する。
4. 左側のペインで、 で作成されたデータベースを選択しますテーブルを作成する。
5. クエリエディタでは、クエリを実行できます。テーブル内の最新の 10 行を表示するには、以下を実行します。

```
SELECT * FROM <database_name>.<table_name> ORDER BY time DESC LIMIT 10
```

6. (オプション) Enable Insights をオンにして、クエリの効率に関するインサイトを取得します。

スケジュールされたクエリを作成する

AWS コンソールを使用してスケジュールされたクエリを作成するには、次の手順に従います。

1. AWS コンソール を開きます。
2. ナビゲーションペインで、スケジュールされたクエリ を選択します。
3. スケジュールされたクエリの作成 をクリックします。

4. クエリ名と宛先テーブルセクションに、以下を入力します。

   - 名前 — クエリ名を入力します。
   - データベース名 — で作成されたデータベースの名前を選択します データベースを作成する。
   - テーブル名 — で作成されたテーブルの名前を選択しますテーブルを作成する。
5. クエリステートメント セクションで、有効なクエリステートメントを入力します。次に、クエリの検証 をクリックします。
6. 送信先テーブルモデル から、未定義の属性のモデルを定義します。Visual Builder または を使用できますJSON。
7. スケジュールの実行 セクションで、固定レートまたは Chron 式 を選択します。時系列については、スケジュール式の詳細については、スケジュールされたクエリのスケジュール式を参照してください。
8. SNS トピックセクションで、通知に使用するSNSトピックを入力します。
9. エラーログレポートセクションに、エラーの報告に使用される S3 の場所を入力します。

   [Encryption key type] (暗号化キーの種類) を選択します。
10. AWS KMS キー のセキュリティ設定セクションで、キーのタイプ AWS KMSを選択します。

    Timestream for がスケジュールされたクエリの実行 LiveAnalytics に使用するIAMロールを入力します。ロールに必要なアクセス許可と信頼関係の詳細については、IAMスケジュールされたクエリのポリシー例を参照してください。
11. スケジュールされたクエリの作成 をクリックします。

スケジュールされたクエリを削除する

コンソールを使用して AWS スケジュールされたクエリを削除または無効にするには、次の手順に従います。

1. AWS コンソール を開きます。
2. ナビゲーションペインで、スケジュールされたクエリを選択します。
3. で作成されたスケジュールされたクエリを選択しますスケジュールされたクエリを作成する。
4. アクション を選択します。
5. の無効化または削除を選択します。
6. 削除を選択した場合は、アクションを確認し、「削除」を選択します。

テーブルを削除する

AWS コンソールを使用してデータベースを削除するには、次の手順に従います。

1. AWS コンソール を開きます。
2. ナビゲーションペインで、テーブルを選択します。
3. で作成したテーブルを選択しますテーブルを作成する。
4. [Delete (削除)] をクリックします。
5. 確認ボックスに delete と入力します。

データベースを削除する

AWS コンソールを使用してデータベースを削除するには、次の手順に従います。

1. AWS コンソール を開きます。
2. ナビゲーションペインで、データベースを選択します。
3. データベースの作成 で作成したデータベースを選択します。
4. [Delete (削除)] をクリックします。
5. 確認ボックスに delete と入力します。

テーブルを編集する

AWS コンソールを使用してテーブルを編集するには、次の手順に従います。

1. AWS コンソール を開きます。
2. ナビゲーションペインで、テーブルを選択します。
3. で作成したテーブルを選択しますテーブルを作成する。
4. 編集 をクリックします。
5. テーブルの詳細を編集して保存します。

    - メモリストアの保持 — メモリストアにデータを保持する期間を指定します。メモリストアは、到着遅延データ (タイムスタンプが現在の時刻より前のデータ) を含む受信データを処理し、高速 point-in-timeクエリ用に最適化されています。

- マグネティックストアの保持 — マグネティックストアにデータを保持する期間を指定します。マグネティックストアは長期ストレージ用であり、高速分析クエリ用に最適化されています。

データベースを編集する

AWS コンソールを使用してデータベースを編集するには、次の手順に従います。

1. AWS コンソール を開きます。
2. ナビゲーションペインで、データベースを選択します。
3. データベースの作成 で作成したデータベースを選択します。
4. 編集 をクリックします。
5. データベースの詳細を編集して保存します。

LiveAnalytics を使用するための Amazon Timestream へのアクセス AWS CLI

AWS Command Line Interface （ AWS CLI) を使用して、コマンドラインから複数の AWS サービスを制御し、スクリプトを使用して自動化できます。アドホックオペレーション AWS CLI には を使用できます。これを使用して、ユーティリティスクリプト内の LiveAnalytics オペレーションに Amazon Timestream を埋め込むこともできます。

Timestream for AWS CLI で を使用する前に LiveAnalytics、プログラムによるアクセスを設定する必要があります。詳細については、「プログラマチックアクセス権を付与する」を参照してください。

API で Timestream for LiveAnalytics Query で使用できるすべてのコマンドの完全なリストについては AWS CLI、AWS CLI 「 コマンドリファレンス」を参照してください。

API で Timestream for LiveAnalytics Write で使用できるすべてのコマンドの完全なリストについては AWS CLI、AWS CLI 「 コマンドリファレンス」を参照してください。

トピック

- AWS CLIのダウンロードと設定
- での Timestream AWS CLI での の使用 LiveAnalytics

AWS CLIのダウンロードと設定

は Windows、macOSまたは Linux で AWS CLI 実行されます。ダウンロード、インストールおよび設定するには次の手順に従います。

1. http://aws.amazon.com/cli AWS CLI で をダウンロードします。
2. AWS Command Line Interface ユーザーガイドの「 のインストール AWS CLI」およびAWS「 の設定CLI」の手順に従います。

での Timestream AWS CLI での の使用 LiveAnalytics

コマンドライン形式は、 LiveAnalytics オペレーション名の Amazon Timestream と、そのオペレーションのパラメータで構成されます。は、 に加えて、 パラメータ値の省略構文 AWS CLI をサポートしていますJSON。

を使用してhelp、 の Timestream で使用可能なすべてのコマンドを一覧表示します LiveAnalytics。例:

```
aws timestream-write help
```

```
aws timestream-query help
```

また help を使用して、特定コマンドを記述したり、その用法の詳細を確認したりすることもできます。

```
aws timestream-write create-database help
```

例えば、データベースを作成するには：

```
aws timestream-write create-database --database-name myFirstDatabase
```

マグネティックストアの書き込みが有効になっているテーブルを作成するには：

```
aws timestream-write create-table \
--database-name metricsdb \
--table-name metrics \
--magnetic-store-write-properties "{\"EnableMagneticStoreWrites\": true}"
```

単一測定レコードを使用してデータを書き込むには：

```
aws timestream-write write-records \
--database-name metricsdb \
--table-name metrics \
--common-attributes "{\"Dimensions\":[{\"Name\":\"asset_id\", \"Value\":\"100\"}],
 \"Time\":\"1631051324000\",\"TimeUnit\":\"MILLISECONDS\"}" \
--records "[{\"MeasureName\":\"temperature\", \"MeasureValueType\":\"DOUBLE\",
\"MeasureValue\":\"30\"},{\"MeasureName\":\"windspeed\", \"MeasureValueType\":\"DOUBLE
\",\"MeasureValue\":\"7\"},{\"MeasureName\":\"humidity\", \"MeasureValueType\":\"DOUBLE
\",\"MeasureValue\":\"15\"},{\"MeasureName\":\"brightness\", \"MeasureValueType\":
\"DOUBLE\",\"MeasureValue\":\"17\"}]"
```

マルチメジャーレコードを使用してデータを書き込むには：

```
# wide model helper method to create Multi-measure records
function ingest_multi_measure_records {
  epoch=`date +%s`
  epoch+=$i

  # multi-measure records
  aws timestream-write write-records \
  --database-name $src_db_wide \
  --table-name $src_tbl_wide \
  --common-attributes "{\"Dimensions\":[{\"Name\":\"device_id\", \
              \"Value\":\"12345678\"},\
            {\"Name\":\"device_type\", \"Value\":\"iPhone\"}, \
            {\"Name\":\"os_version\", \"Value\":\"14.8\"}, \
            {\"Name\":\"region\", \"Value\":\"us-east-1\"} ], \
            \"Time\":\"$epoch\",\"TimeUnit\":\"MILLISECONDS\"}" \
--records "[{\"MeasureName\":\"video_metrics\", \"MeasureValueType\":\"MULTI\", \
  \"MeasureValues\": \
  [{\"Name\":\"video_startup_time\",\"Value\":\"0\",\"Type\":\"BIGINT\"}, \
  {\"Name\":\"rebuffering_ratio\",\"Value\":\"0.5\",\"Type\":\"DOUBLE\"}, \
  {\"Name\":\"video_playback_failures\",\"Value\":\"0\",\"Type\":\"BIGINT\"}, \
  {\"Name\":\"average_frame_rate\",\"Value\":\"0.5\",\"Type\":\"DOUBLE\"}]}]" \
--endpoint-url $ingest_endpoint \
  --region  $region
}

# create 5 records
for i in {100..105};
  do ingest_multi_measure_records $i;
```

```
done
```

テーブルに対してクエリを実行するには:

```
aws timestream-query query \
--query-string "SELECT time, device_id, device_type, os_version,
region, video_startup_time, rebuffering_ratio, video_playback_failures, \
average_frame_rate \
FROM metricsdb.metrics \
where time >= ago (15m)"
```

スケジュールされたクエリを作成するには：

```
aws timestream-query create-scheduled-query \
  --name scheduled_query_name \
  --query-string "select bin(time, 1m) as time, \
          avg(measure_value::double) as avg_cpu, min(measure_value::double) as min_cpu,
 region \
          from $src_db.$src_tbl where measure_name = 'cpu' \
          and time BETWEEN @scheduled_runtime - (interval '5' minute)  AND
 @scheduled_runtime \
          group by region, bin(time, 1m)" \
  --schedule-configuration "{\"ScheduleExpression\":\"$cron_exp\"}" \
  --notification-configuration "{\"SnsConfiguration\":{\"TopicArn\":\"$sns_topic_arn
\"}}" \
  --scheduled-query-execution-role-arn "arn:aws:iam::452360119086:role/
TimestreamSQExecutionRole" \
  --target-configuration "{\"TimestreamConfiguration\":{\
          \"DatabaseName\": \"$dest_db\",\
          \"TableName\": \"$dest_tbl\",\
          \"TimeColumn\":\"time\",\
          \"DimensionMappings\":[{\
            \"Name\": \"region\", \"DimensionValueType\": \"VARCHAR\"
          }],\
          \"MultiMeasureMappings\":{\
            \"TargetMultiMeasureName\": \"mma_name\",
            \"MultiMeasureAttributeMappings\":[{\
              \"SourceColumn\": \"avg_cpu\", \"MeasureValueType\": \"DOUBLE\",
 \"TargetMultiMeasureAttributeName\": \"target_avg_cpu\"
            },\
            { \
              \"SourceColumn\": \"min_cpu\", \"MeasureValueType\": \"DOUBLE\",
 \"TargetMultiMeasureAttributeName\": \"target_min_cpu\"
```

```
            }] \
          }\
          }}" \
  --error-report-configuration "{\"S3Configuration\": {\
        \"BucketName\": \"$s3_err_bucket\",\
        \"ObjectKeyPrefix\": \"scherrors\",\
        \"EncryptionOption\": \"SSE_S3\"\
        }\
      }"
```

API の使用

に加えてSDKs、 の Amazon Timestream LiveAnalytics では、エンドポイント検出パターン を介して直接RESTAPIアクセスできます。エンドポイント検出パターンとそのユースケースを以下に示します。

エンドポイント検出パターン

Timestream Live Analytics の SDKsは、サービスエンドポイントの管理やマッピングなど、サービスのアーキテクチャと透過的に連携するように設計されているため、ほとんどのアプリケーションSDKsでは を使用することをお勧めします。ただし、APIエンドポイント検出パターンに LiveAnalytics REST Timestream を使用する必要があるインスタンスがいくつかあります。

- VPC の Timestream でエンドポイント (AWS PrivateLink) LiveAnalytics を使用している
- アプリケーションが、まだSDKサポートされていないプログラミング言語を使用している
- クライアント側の実装をより適切に制御する必要がある

このセクションでは、エンドポイント検出パターンの仕組み、エンドポイント検出パターンの実装方法、および使用上の注意について説明します。詳細については、以下のトピックを選択してください。

トピック

- エンドポイント検出パターンの仕組み
- エンドポイント検出パターンの実装

エンドポイント検出パターンの仕組み

Timestream は、セルラーアーキテクチャを使用して構築されており、スケーリングとトラフィック分離のプロパティが向上します。各カスタマーアカウントはリージョン内の特定のセルにマッピングされるため、アプリケーションはアカウントがマッピングされた正しいセル固有のエンドポイントを使用する必要があります。を使用する場合SDKs、このマッピングは透過的に処理されるため、セル固有のエンドポイントを管理する必要はありません。ただし、 に直接アクセスする場合はRESTAPI、正しいエンドポイントを自分で管理してマッピングする必要があります。エンドポイント検出パターン であるこのプロセスを以下に示します。

1. エンドポイント検出パターンは、 `DescribeEndpoints`アクション (`DescribeEndpoints`セクションで説明) の呼び出しから始まります。
2. エンドポイントはキャッシュされ、返 time-to-liveされた (TTL) 値 ( ) で指定された時間だけ再利用されます`CachePeriodInMinutes`。Timestream Live Analytics への呼び出しは、 の期間中 API実行できますTTL。
3. TTL の有効期限が切れたら、 に新しい呼び出しを実行してエンドポイントを更新 DescribeEndpoints する必要があります (つまり、ステップ 1 からやり直す) 。

> **Note**
>
> `DescribeEndpoints` アクションの構文、パラメータ、およびその他の使用状況については、 APIリファレンス を参照してください。`DescribeEndpoints` アクションは の両方で使用可能でありSDKs、それぞれ同じであることに注意してください。

エンドポイント検出パターンの実装については、「」を参照してくださいエンドポイント検出パターンの実装。

エンドポイント検出パターンの実装

エンドポイント検出パターンを実装するには、 API (書き込みまたはクエリ) を選択し、DescribeEndpointsリクエストを作成し、返されたTTL値 (複数可) の期間中、返されたエンドポイント (複数可) を使用します。実装手順を以下に示します。

> **Note**
>
> 使用上の注意 に精通していることを確認してください。

実装手順

1. DescribeEndpoints リクエストを使用して、 (書き込みまたはクエリ) に対して呼び出しAPIを行う のエンドポイントを取得します。

   a. 以下に説明する 2 つのエンドポイントのいずれかを使用して、API対象の (書き込みまたはクエリ) DescribeEndpointsに対応する リクエストを作成します。リクエストの入力パラメータはありません。以下のメモを必ずお読みください。

      書き込みSDK :

      ```
      ingest.timestream.<region>.amazonaws.com
      ```

      クエリSDK :

      ```
      query.timestream.<region>.amazonaws.com
      ```

      リージョンのCLI呼び出しの例us-east-1を次に示します。

      ```
      REGION_ENDPOINT="https://query.timestream.us-east-1.amazonaws.com"
      REGION=us-east-1
      aws timestream-write describe-endpoints \
      --endpoint-url $REGION_ENDPOINT \
      --region $REGION
      ```

      > **Note**
      >
      > HTTP 「ホスト」ヘッダーにはAPIエンドポイントも含める必要があります。ヘッダーが入力されていない場合、リクエストは失敗します。これは、すべてのHTTP/1.1 リクエストの標準要件です。1.1 以降をサポートするHTTPライブラリを使用する場合、HTTPライブラリはヘッダーを自動的に入力する必要があります。

      > **Note**
      >
      > 置換 *<region>* リクエストが行われるリージョンのリージョン識別子。例: us-east-1

   b. レスポンスを解析して、エンドポイント (複数可) とキャッシュTTL値 (複数可) を抽出します。レスポンスは、1 Endpoint つ以上のオブジェクト の配列です。各Endpointオブジェクトには、エンドポイントアドレス (Address) とそのエンドポイント () TTLの が含まれますCachePeriodInMinutes。

2. エンドポイントを指定された までキャッシュしますTTL。
3. TTL の有効期限が切れたら、実装のステップ 1 で最初からやり直すことで、新しいエンドポイントを取得します。

エンドポイント検出パターンの使用上の注意

- このDescribeEndpointsアクションは、Timestream Live Analytics リージョンエンドポイントが認識する唯一のアクションです。
- レスポンスには、Timestream Live Analytics をAPI呼び出すエンドポイントのリストが含まれています。
- 応答が成功すると、リストに少なくとも 1 つのエンドポイントが存在する必要があります。リストに複数のエンドポイントがある場合、そのいずれかがAPI呼び出しに等しく使用可能であり、発信者はランダムに使用するエンドポイントを選択できます。
- エンドポイントのDNSアドレスに加えて、リスト内の各エンドポイントは、数分で指定されたエンドポイントの使用が許可される有効期間 (TTL) を指定します。
- エンドポイントはキャッシュされ、返されたTTL値で指定された時間 (分単位) 再利用される必要があります。TTL の有効期限が切れると、エンドポイントTTLは有効期限が切れた後は機能しなくなるため、 への新しい呼び出しを実行して、使用するエンドポイントを更新DescribeEndpointsする必要があります。

の使用 AWS SDKs

を使用して Amazon Timestream にアクセスできます AWS SDKs。Timestream は、言語SDKsごとに 2 つ、つまり書き込み SDKとクエリ をサポートしていますSDK。書き込みSDKは、CRUDオペレーションを実行し、時系列データを Timestream に挿入するために使用されます。クエリSDKは、Timestream に保存されている既存の時系列データをクエリするために使用されます。

選択した に必要な前提条件が完了したらSDK、 を開始できますコードサンプル。

トピック

- Java

- Java v2
- Go
- Python
- Node.js
- .NET

Java

Java 1.0 SDKと Amazon Timestream の使用を開始するには、以下に説明する前提条件を完了します。

Java に必要な前提条件が完了したらSDK、 の使用を開始できますコードサンプル。

前提条件

Java の使用を開始する前に、以下を実行する必要があります。

1. AWS の設定手順に従ってくださいの Timestream へのアクセス LiveAnalytics。
2. 以下をダウンロードおよびインストールして、Java 開発環境を設定します。
   - Java SE 開発キット 8 (Amazon Corretto 8 など)。
   - Java IDE (Eclipse や IntelliJなど)。

   詳細については、「 の開始方法」を参照してください。 AWS SDK for Java
3. 開発用に AWS 認証情報とリージョンを設定します。
   - で使用するセキュリティ認証情報を設定します AWS AWS SDK for Java。
   - リージョンを設定 AWS して、 LiveAnalytics エンドポイントのデフォルトのタイムストリームを決定します。

Apache Maven の使用

Apache Maven を使用して、 AWS SDK for Java プロジェクトを設定および構築できます。

> **Note**
>
> Apache Maven を使用するには、Java SDKとランタイムが 1.8 以上であることを確認します。

AWS SDK 「Apache Maven SDKでの の使用」の説明に従って、 を Maven の依存関係として設定できます。

次のコマンドを使用して、コンパイルを実行し、ソースコードを実行できます。

```
mvn clean compile
mvn exec:java -Dexec.mainClass=<your source code Main class>
```

> **Note**
>
> <your source code Main class> は、Java ソースコードのメインクラスへのパスです。

認証情報の設定 AWS

AWS SDK for Java では、実行時にアプリケーションに AWS 認証情報を指定する必要があります。このガイドのコード例は、 AWS SDK for Java デベロッパーガイドの「開発のための AWS 認証情報とリージョンの設定」で説明されているように、認証情報ファイルを使用している AWS ことを前提としています。

以下は、 という名前の AWS 認証情報ファイルの例~/.aws/credentialsです。チルド文字 (~) はホームディレクトリを表します。

```
[default]
aws_access_key_id = AWS access key ID goes here
aws_secret_access_key = Secret key goes here
```

Java v2

Java 2.0 SDKと Amazon Timestream の使用を開始するには、以下に説明する前提条件を完了します。

Java 2.0 に必要な前提条件が完了したらSDK、 を開始できますコードサンプル。

前提条件

Java の使用を開始する前に、以下を実行する必要があります。

1. AWS の設定手順に従ってくださいの Timestream へのアクセス LiveAnalytics。

2. AWS SDK 「Apache Maven SDKでの の使用」の説明に従って、 を Maven の依存関係として設定できます。
3. 以下をダウンロードおよびインストールして、Java 開発環境を設定します。

   - Java SE 開発キット 8 (Amazon Corretto 8 など)。
   - Java IDE (Eclipse や IntelliJなど)。

     詳細については、「 の開始方法」を参照してください。 AWS SDK for Java

Apache Maven の使用

Apache Maven を使用して AWS SDK for Java プロジェクトを設定および構築できます。

> **Note**
>
> Apache Maven を使用するには、Java SDKとランタイムが 1.8 以上であることを確認します。

AWS SDK 「Apache Maven SDKでの の使用」の説明に従って、 を Maven の依存関係として設定できます。pom.xml ファイルに必要な変更については、ここで説明します。

次のコマンドを使用して、コンパイルを実行し、ソースコードを実行できます。

```
mvn clean compile
mvn exec:java -Dexec.mainClass=<your source code Main class>
```

> **Note**
>
> <your source code Main class> は、Java ソースコードのメインクラスへのパスです。

Go

Go SDK と Amazon Timestream の使用を開始するには、以下に説明する前提条件を完了します。

Go に必要な前提条件が完了したらSDK、 を開始できますコードサンプル。

前提条件

1. GO SDK 1.14 をダウンロードします。
2. GO を設定しますSDK。
3. クライアント を構築します。

Python

Python SDK と Amazon Timestream の使用を開始するには、以下に説明する前提条件を完了します。

Python に必要な前提条件が完了したらSDK、 の使用を開始できますコードサンプル。

前提条件

Python を使用するには、「」の手順に従って Boto3 をインストールして設定します。 https://boto3.amazonaws.com/v1/documentation/api/latest/index.html

Node.js

Node.js SDK と Amazon Timestream の使用を開始するには、以下に説明する前提条件を完了します。

Node.js に必要な前提条件が完了したらSDK、 を開始できますコードサンプル。

前提条件

Node.js の使用を開始する前に、以下を実行する必要があります。

1. Node.js をインストールします。
2. の をインストールします AWS SDK JavaScript。

.NET

.NET SDK と Amazon Timestream の使用を開始するには、以下に説明する前提条件を完了します。

に必要な前提条件NETが完了したらSDK、 を開始できますコードサンプル。

前提条件

の使用を開始する前にNET、必要な NuGet パッケージをインストールし、次のコマンドを実行して AWSSDK.Core バージョンが 3.3.107 以降であることを確認します。

```
dotnet add package AWSSDK.Core
dotnet add package AWSSDK.TimestreamWrite
dotnet add package AWSSDK.TimestreamQuery
```

使用開始

このセクションでは、Amazon Timestream Live Analytics の使用を開始するチュートリアルと、完全に機能するサンプルアプリケーションを設定する手順について説明します。チュートリアルまたはサンプルアプリケーションの使用を開始するには、以下のリンクのいずれかを選択します。

トピック

- チュートリアル
- サンプルアプリケーション

チュートリアル

このチュートリアルでは、サンプルデータセットが入力されたデータベースを作成し、サンプルクエリを実行する方法について説明します。このチュートリアルで使用されるサンプルデータセットは、IoT や DevOps シナリオで頻繁に見られます。IoT データセットには、トラックの速度、場所、負荷などの時系列データが含まれており、フリート管理を合理化し、最適化の機会を特定します。 DevOps データセットには、アプリケーションのパフォーマンスと可用性を向上させるためにCPU、、ネットワーク、メモリ使用率などのEC2インスタンスメトリクスが含まれています。このセクションで説明する手順のビデオチュートリアルを以下に示します。

以下の手順に従って、サンプルデータセットが入力されたデータベースを作成し、 AWS コンソールを使用してサンプルクエリを実行します。

コンソールを使用する

以下の手順に従って、サンプルデータセットが入力されたデータベースを作成し、 AWS コンソールを使用してサンプルクエリを実行します。

1. AWS コンソール を開きます。

2. ナビゲーションペインで、データベースを選択します。
3. データベースの作成 をクリックします。
4. データベースの作成ページで、次のように入力します。

   - 設定を選択 — サンプルデータベース を選択します。
   - 名前 — 選択したデータベース名を入力します。

   > **Note**
   >
   > サンプルデータセットを使用してデータベースを作成した後、コンソールで使用可能なサンプルクエリを使用するには、ここに入力したデータベース名と一致するようにクエリで参照されるデータベース名を調整できます。サンプルデータセットと時系列レコードのタイプの組み合わせごとにサンプルクエリがあります。

   - サンプルデータセットの選択 — IoT と を選択しますDevOps。
   - 時系列レコードのタイプを選択する — マルチメジャーレコードを選択します。
   - データベースの作成をクリックして、サンプルデータが入力された 2 つのテーブルを含むデータベースを作成します。マルチメジャーレコードを含むサンプルデータセットのテーブル名は、 `DevOpsMulti`および ですIoTMulti。単一測定レコードを持つサンプルデータセットのテーブル名は `DevOps`および ですIoT。
5. ナビゲーションペインで、クエリエディタを選択します。
6. トップメニューからサンプルクエリを選択します。
7. サンプルデータベースの作成時に選択したデータセットのサンプルクエリの 1 つをクリックします。これにより、サンプルクエリが入力されたエディタでクエリエディタに戻ります。
8. サンプルクエリのデータベース名を調整します。
9. Run をクリックしてクエリを実行し、クエリ結果を表示します。

SDKs の使用

Timestream Live Analytics は、データベースとテーブルの作成、約 126,000 行のサンプルデータのテーブルへの入力、サンプルクエリの実行方法を示す、完全に機能するサンプルアプリケーションを提供します。サンプルアプリケーションは、 GitHub for Java、Python、Node.js、Go、および で使用できますNET。

1. の手順に従って GitHub 、リポジトリ Timestream Live Analytics サンプルアプリケーションをクローンします GitHub。

2. の説明に従って Amazon Timestream Live Analytics の使用 AWS SDKs に接続するように AWS SDKを設定します。
3. 以下の手順に従って、サンプルアプリケーションをコンパイルして実行します。

   - Java サンプルアプリケーション の手順。
   - Java v2 サンプルアプリケーション の手順。
   - Go サンプルアプリケーション の手順。
   - Python サンプルアプリケーション の手順。
   - Node.js サンプルアプリケーション の手順。
   - . サンプルアプリケーション の手順NET。

サンプルアプリケーション

Timestream には、データベースとテーブルの作成、約 126,000 行のサンプルデータのテーブルへの入力、サンプルクエリの実行方法を示す、完全に機能するサンプルアプリケーションが付属しています。サポートされているいずれかの言語でサンプルアプリケーションの使用を開始するには、次の手順に従います。

Java

1. の指示に従って、サンプルアプリケーションの GitHub リポジトリ Timestream をクローンしますGitHub。 LiveAnalytics
2. 「 の開始方法」で説明されている LiveAnalytics 手順に従って Timestream に接続する AWSSDKように を設定しますJava。
3. Java サンプルアプリケーションは、ここで説明する手順に従って実行します。

Java v2

1. の指示に従って、サンプルアプリケーションの GitHub リポジトリ Timestream をクローンしますGitHub。 LiveAnalytics
2. 「 の開始方法」で説明されている LiveAnalytics 手順に従って Amazon Timestream に接続するように を設定します AWS SDKJava v2。
3. Java 2.0 サンプルアプリケーションを、ここで説明する手順に従って実行します。

Go

1. の指示に従って、サンプルアプリケーションの GitHub リポジトリ Timestream をクローンしますGitHub。 LiveAnalytics
2. 「 の開始方法」で説明されている LiveAnalytics 手順に従って Amazon Timestream に接続するように を設定します AWS SDKGo。
3. ここに記載されている手順に従って Go サンプルアプリケーションを実行します。

Python

1. の指示に従って、サンプルアプリケーションの GitHub リポジトリ Timestream をクローンしますGitHub。 LiveAnalytics
2. で説明されている LiveAnalytics 手順に従って Amazon Timestream に接続するように AWS SDKを設定しますPython。
3. Python サンプルアプリケーションは、ここで説明されている手順に従って実行します。

Node.js

1. の指示に従って、サンプルアプリケーションの GitHub リポジトリ Timestream をクローンしますGitHub。 LiveAnalytics
2. 「 の開始方法」で説明されている LiveAnalytics 手順に従って Amazon Timestream に接続するように を設定します AWS SDKNode.js。
3. ここで説明する手順に従って Node.js サンプルアプリケーションを実行する https://github.com/awslabs/amazon-timestream-tools/blob/master/sample_apps/js/README.md

.NET

1. の指示に従って、サンプルアプリケーションの GitHub リポジトリ Timestream をクローンしますGitHub。 LiveAnalytics
2. 「 の開始方法」で説明されている LiveAnalytics 手順に従って Amazon Timestream に接続するように を設定します AWS SDK.NET。
3. https://github.com/awslabs/amazon-timestream-tools/blob/master/sample_apps/dotnet/README.md「」で説明されている手順に従って . サンプルアプリケーションを実行します。NET

コードサンプル

を使用して Amazon Timestream にアクセスできます AWS SDKs。Timestream は、言語SDKsごとに 2 つ、つまり書き込み SDKとクエリ をサポートしますSDK。書き込みSDKは、CRUDオペレーションを実行し、時系列データを Timestream に挿入するために使用されます。クエリSDKは、Timestream に保存されている既存の時系列データをクエリするために使用されます。サポートされている各 のコードサンプルなど、詳細については、以下のリストからトピックを選択してくださいSDKs。

トピック

- SDK クライアントを書き込む
- クエリSDKクライアント
- データベースを作成
- データベースの説明
- データベースの更新
- データベースの削除
- データベースを一覧表示する
- テーブルの作成
- テーブルの説明
- テーブルを更新する
- テーブルを削除する
- テーブルの一覧表示
- 書き込みデータ (挿入とアップサート)
- クエリの実行
- UNLOAD クエリの実行
- クエリをキャンセルする
- バッチロードタスクの作成
- バッチロードタスクの説明
- バッチロードタスクを一覧表示する
- バッチロードタスクを再開する
- スケジュールされたクエリを作成する

- スケジュールされたクエリを一覧表示する
- スケジュールされたクエリを記述する
- スケジュールされたクエリを実行する
- スケジュールされたクエリを更新する
- スケジュールされたクエリを削除する

SDK クライアントを書き込む

次のコードスニペットを使用して、書き込み の Timestream クライアントを作成できますSDK。書き込みSDKは、CRUDオペレーションを実行し、時系列データを Timestream に挿入するために使用されます。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいていますGitHub。サンプルアプリケーションの開始方法の詳細については、「」を参照してくださいサンプルアプリケーション。

Java

```
    private static AmazonTimestreamWrite buildWriteClient() {
        final ClientConfiguration clientConfiguration = new ClientConfiguration()
                .withMaxConnections(5000)
                .withRequestTimeout(20 * 1000)
                .withMaxErrorRetry(10);

        return AmazonTimestreamWriteClientBuilder
                .standard()
                .withRegion("us-east-1")
                .withClientConfiguration(clientConfiguration)
                .build();
    }
```

Java v2

```
    private static TimestreamWriteClient buildWriteClient() {
        ApacheHttpClient.Builder httpClientBuilder =
```

```
                ApacheHttpClient.builder();
        httpClientBuilder.maxConnections(5000);

        RetryPolicy.Builder retryPolicy =
                RetryPolicy.builder();
        retryPolicy.numRetries(10);

        ClientOverrideConfiguration.Builder overrideConfig =
                ClientOverrideConfiguration.builder();
        overrideConfig.apiCallAttemptTimeout(Duration.ofSeconds(20));
        overrideConfig.retryPolicy(retryPolicy.build());

        return TimestreamWriteClient.builder()
                .httpClientBuilder(httpClientBuilder)
                .overrideConfiguration(overrideConfig.build())
                .region(Region.US_EAST_1)
                .build();
    }
```

Go

```
tr := &http.Transport{
        ResponseHeaderTimeout: 20 * time.Second,
        // Using DefaultTransport values for other parameters: https://golang.org/
pkg/net/http/#RoundTripper
        Proxy: http.ProxyFromEnvironment,
        DialContext: (&net.Dialer{
            KeepAlive: 30 * time.Second,
            DualStack: true,
            Timeout:   30 * time.Second,
        }).DialContext,
        MaxIdleConns:          100,
        IdleConnTimeout:       90 * time.Second,
        TLSHandshakeTimeout:   10 * time.Second,
        ExpectContinueTimeout: 1 * time.Second,
    }

    // So client makes HTTP/2 requests
    http2.ConfigureTransport(tr)

    sess, err := session.NewSession(&aws.Config{ Region: aws.String("us-east-1"),
 MaxRetries: aws.Int(10), HTTPClient: &http.Client{ Transport: tr }})
    writeSvc := timestreamwrite.New(sess)
```

Python

```
write_client = session.client('timestream-write', config=Config(read_timeout=20,
 max_pool_connections = 5000, retries={'max_attempts': 10}))
```

Node.js

次のスニペットは JavaScript v3 AWSSDKに を使用します。クライアントのインストール方法と使用状況の詳細については、 JavaScript 「v3 AWSSDKの Timestream Write Client -」を参照してください。

追加のコマンドインポートがここに表示されます。クライアントの作成にはCreateDatabaseCommandインポートは必要ありません。

```
import { TimestreamWriteClient, CreateDatabaseCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });
```

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
var https = require('https');
var agent = new https.Agent({
    maxSockets: 5000
});
writeClient = new AWS.TimestreamWrite({
        maxRetries: 10,
        httpOptions: {
            timeout: 20000,
            agent: agent
        }
    });
```

.NET

```
var writeClientConfig = new AmazonTimestreamWriteConfig
{
    RegionEndpoint = RegionEndpoint.USEast1,
    Timeout = TimeSpan.FromSeconds(20),
```

```
    MaxErrorRetry = 10
};

var writeClient = new AmazonTimestreamWriteClient(writeClientConfig);
```

次の設定を使用することをお勧めします。

- SDK 再試行回数を に設定します10。
- SDK DEFAULT_BACKOFF_STRATEGY を使用します。
- RequestTimeout 20 秒に設定します。
- 最大接続数を 5000 以上に設定してください。

クエリSDKクライアント

次のコードスニペットを使用して、クエリ の Timestream クライアントを作成できますSDK。クエリSDKは、Timestream に保存されている既存の時系列データをクエリするために使用されます。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいていますGitHub。サンプルアプリケーションの使用を開始する方法の詳細については、「」を参照してくださいサンプルアプリケーション。

Java

```
    private static AmazonTimestreamQuery buildQueryClient() {
        AmazonTimestreamQuery client =
AmazonTimestreamQueryClient.builder().withRegion("us-east-1").build();
        return client;
    }
```

Java v2

```
    private static TimestreamQueryClient buildQueryClient() {
        return TimestreamQueryClient.builder()
                .region(Region.US_EAST_1)
```

```
                .build();
    }
```

Go

```
sess, err := session.NewSession(&aws.Config{Region: aws.String("us-east-1")})
```

Python

```
query_client = session.client('timestream-query')
```

Node.js

次のスニペットは JavaScript v3 に を使用します AWS SDK。クライアントのインストール方法と使用状況の詳細については、 JavaScript 「v3 の Timestream Query Client -AWS SDK」を参照してください。

追加のコマンドインポートがここに表示されます。クライアントの作成にはQueryCommandインポートは必要ありません。

```
import { TimestreamQueryClient, QueryCommand } from "@aws-sdk/client-timestream-
query";
const queryClient = new TimestreamQueryClient({ region: "us-east-1" });
```

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
queryClient = new AWS.TimestreamQuery();
```

.NET

```
var queryClientConfig = new AmazonTimestreamQueryConfig
{
    RegionEndpoint = RegionEndpoint.USEast1
};

var queryClient = new AmazonTimestreamQueryClient(queryClientConfig);
```

データベースを作成

次のコードスニペットを使用してデータベースを作成できます。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいていますGitHub。サンプルアプリケーションの使用を開始する方法の詳細については、「」を参照してくださいサンプルアプリケーション。

Java

```
  public void createDatabase() {
       System.out.println("Creating database");
       CreateDatabaseRequest request = new CreateDatabaseRequest();
       request.setDatabaseName(DATABASE_NAME);
       try {
           amazonTimestreamWrite.createDatabase(request);
           System.out.println("Database [" + DATABASE_NAME + "] created
successfully");
       } catch (ConflictException e) {
           System.out.println("Database [" + DATABASE_NAME + "] exists. Skipping
database creation");
       }
   }
```

Java v2

```
   public void createDatabase() {
       System.out.println("Creating database");
       CreateDatabaseRequest request =
CreateDatabaseRequest.builder().databaseName(DATABASE_NAME).build();
       try {
           timestreamWriteClient.createDatabase(request);
           System.out.println("Database [" + DATABASE_NAME + "] created
successfully");
       } catch (ConflictException e) {
           System.out.println("Database [" + DATABASE_NAME + "] exists. Skipping
database creation");
       }
```

```
    }
```

Go

```
// Create database.
    createDatabaseInput := &timestreamwrite.CreateDatabaseInput{
        DatabaseName: aws.String(*databaseName),
    }

    _, err = writeSvc.CreateDatabase(createDatabaseInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Database successfully created")
    }

    fmt.Println("Describing the database, hit enter to continue")
```

Python

```
    def create_database(self):
        print("Creating Database")
        try:
            self.client.create_database(DatabaseName=Constant.DATABASE_NAME)
            print("Database [%s] created successfully." % Constant.DATABASE_NAME)
        except self.client.exceptions.ConflictException:
            print("Database [%s] exists. Skipping database creation" %
 Constant.DATABASE_NAME)
        except Exception as err:
            print("Create database failed:", err)
```

Node.js

次のスニペットは JavaScript v3 AWSSDKに を使用します。クライアントのインストール方法と使用状況の詳細については、 JavaScript 「v3 の Timestream Write Client -AWSSDK」を参照してください。

クラス CreateDatabaseCommand と も参照してくださいCreateDatabase。

```
import { TimestreamWriteClient, CreateDatabaseCommand } from "@aws-sdk/client-
timestream-write";
```

```
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode"
};

const command = new CreateDatabaseCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Database ${data.Database.DatabaseName} created successfully`);
} catch (error) {
    if (error.code === 'ConflictException') {
        console.log(`Database ${params.DatabaseName} already exists. Skipping
 creation.`);
    } else {
        console.log("Error creating database", error);
    }
}
```

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
async function createDatabase() {
    console.log("Creating Database");
    const params = {
        DatabaseName: constants.DATABASE_NAME
    };

    const promise = writeClient.createDatabase(params).promise();

    await promise.then(
        (data) => {
            console.log(`Database ${data.Database.DatabaseName} created
 successfully`);
        },
        (err) => {
            if (err.code === 'ConflictException') {
                console.log(`Database ${params.DatabaseName} already exists.
 Skipping creation.`);
            } else {
                console.log("Error creating database", err);
```

```
            }
        }
    );
}
```

.NET

```
        public async Task CreateDatabase()
        {
            Console.WriteLine("Creating Database");

            try
            {
                var createDatabaseRequest = new CreateDatabaseRequest
                {
                    DatabaseName = Constants.DATABASE_NAME
                };
                CreateDatabaseResponse response = await
 writeClient.CreateDatabaseAsync(createDatabaseRequest);
                Console.WriteLine($"Database {Constants.DATABASE_NAME} created");
            }
            catch (ConflictException)
            {
                Console.WriteLine("Database already exists.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Create database failed:" + e.ToString());
            }

        }
```

データベースの説明

次のコードスニペットを使用して、新しく作成されたデータベースの属性に関する情報を取得できます。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいていますGitHub。サンプルアプリケーションの使用を開始する方法の詳細については、「」を参照してくださいサンプルアプリケーション。

Java

```
   public void describeDatabase() {
       System.out.println("Describing database");
       final DescribeDatabaseRequest describeDatabaseRequest = new
DescribeDatabaseRequest();
       describeDatabaseRequest.setDatabaseName(DATABASE_NAME);
       try {
           DescribeDatabaseResult result =
amazonTimestreamWrite.describeDatabase(describeDatabaseRequest);
           final Database databaseRecord = result.getDatabase();
           final String databaseId = databaseRecord.getArn();
           System.out.println("Database " + DATABASE_NAME + " has id " +
databaseId);
       } catch (final Exception e) {
           System.out.println("Database doesn't exist = " + e);
           throw e;
       }
   }
```

Java v2

```
   public void describeDatabase() {
       System.out.println("Describing database");
       final DescribeDatabaseRequest describeDatabaseRequest =
DescribeDatabaseRequest.builder()
               .databaseName(DATABASE_NAME).build();
       try {
           DescribeDatabaseResponse response =
timestreamWriteClient.describeDatabase(describeDatabaseRequest);
           final Database databaseRecord = response.database();
           final String databaseId = databaseRecord.arn();
           System.out.println("Database " + DATABASE_NAME + " has id " +
databaseId);
       } catch (final Exception e) {
```

```
            System.out.println("Database doesn't exist = " + e);
            throw e;
        }
    }
```

Go

```
describeDatabaseOutput, err := writeSvc.DescribeDatabase(describeDatabaseInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Describe database is successful, below is the output:")
        fmt.Println(describeDatabaseOutput)
    }
```

Python

```
    def describe_database(self):
        print("Describing database")
        try:
            result =
 self.client.describe_database(DatabaseName=Constant.DATABASE_NAME)
            print("Database [%s] has id [%s]" % (Constant.DATABASE_NAME,
 result['Database']['Arn']))
        except self.client.exceptions.ResourceNotFoundException:
            print("Database doesn't exist")
        except Exception as err:
            print("Describe database failed:", err)
```

Node.js

次のスニペットは JavaScript v3 AWSSDKに を使用します。クライアントのインストール方法と使用状況の詳細については、 JavaScript 「v3 の Timestream Write Client -AWSSDK」を参照してください。

クラス DescribeDatabaseCommand と も参照してくださいDescribeDatabase。

```
import { TimestreamWriteClient, DescribeDatabaseCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });
```

```
const params = {
    DatabaseName: "testDbFromNode"
};

const command = new DescribeDatabaseCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Database ${data.Database.DatabaseName} has id
 ${data.Database.Arn}`);
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log("Database doesn't exist.");
    } else {
        console.log("Describe database failed.", error);
        throw error;
    }
}
```

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
async function describeDatabase () {
    console.log("Describing Database");
    const params = {
        DatabaseName: constants.DATABASE_NAME
    };

    const promise = writeClient.describeDatabase(params).promise();

    await promise.then(
        (data) => {
            console.log(`Database ${data.Database.DatabaseName} has id
 ${data.Database.Arn}`);
        },
        (err) => {
            if (err.code === 'ResourceNotFoundException') {
                console.log("Database doesn't exist.");
            } else {
                console.log("Describe database failed.", err);
                throw err;
            }
```

```
        }
    );
}
```

.NET

```
        public async Task DescribeDatabase()
        {
            Console.WriteLine("Describing Database");

            try
            {
                var describeDatabaseRequest = new DescribeDatabaseRequest
                {
                    DatabaseName = Constants.DATABASE_NAME
                };
                DescribeDatabaseResponse response = await
 writeClient.DescribeDatabaseAsync(describeDatabaseRequest);
                Console.WriteLine($"Database {Constants.DATABASE_NAME} has id:
{response.Database.Arn}");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine("Database does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Describe database failed:" + e.ToString());
            }

        }
```

データベースの更新

次のコードスニペットを使用してデータベースを更新できます。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいていますGitHub。サンプルアプリケーションの使用を開始する方法の詳細については、「」を参照してくださいサンプルアプリケーション。

Java

```
   public void updateDatabase(String kmsId) {
       System.out.println("Updating kmsId to " + kmsId);
       UpdateDatabaseRequest request = new UpdateDatabaseRequest();
       request.setDatabaseName(DATABASE_NAME);
       request.setKmsKeyId(kmsId);
       try {
           UpdateDatabaseResult result =
amazonTimestreamWrite.updateDatabase(request);
           System.out.println("Update Database complete");
       } catch (final ValidationException e) {
           System.out.println("Update database failed:");
           e.printStackTrace();
       } catch (final ResourceNotFoundException e) {
           System.out.println("Database " + DATABASE_NAME + " doesn't exist = " +
e);
       } catch (final Exception e) {
           System.out.println("Could not update Database " + DATABASE_NAME + " = "
+ e);
           throw e;
       }
   }
```

Java v2

```
   public void updateDatabase(String kmsKeyId) {

       if (kmsKeyId == null) {
           System.out.println("Skipping UpdateDatabase because KmsKeyId was not
given");
           return;
       }

       System.out.println("Updating database");
```

```
        UpdateDatabaseRequest request = UpdateDatabaseRequest.builder()
                .databaseName(DATABASE_NAME)
                .kmsKeyId(kmsKeyId)
                .build();
        try {
            timestreamWriteClient.updateDatabase(request);
            System.out.println("Database [" + DATABASE_NAME + "] updated
 successfully with kmsKeyId " + kmsKeyId);
        } catch (ResourceNotFoundException e) {
            System.out.println("Database [" + DATABASE_NAME + "] does not exist.
 Skipping UpdateDatabase");
        } catch (Exception e) {
            System.out.println("UpdateDatabase failed: " + e);
        }
    }
```

Go

```
// Update Database.
        updateDatabaseInput := &timestreamwrite.UpdateDatabaseInput {
            DatabaseName: aws.String(*databaseName),
            KmsKeyId: aws.String(*kmsKeyId),
        }

        updateDatabaseOutput, err := writeSvc.UpdateDatabase(updateDatabaseInput)

        if err != nil {
            fmt.Println("Error:")
            fmt.Println(err)
        } else {
            fmt.Println("Update database is successful, below is the output:")
            fmt.Println(updateDatabaseOutput)
        }
```

Python

```
    def update_database(self, kms_id):
        print("Updating database")
        try:
            result =
 self.client.update_database(DatabaseName=Constant.DATABASE_NAME, KmsKeyId=kms_id)
```

```
            print("Database [%s] was updated to use kms [%s] successfully" %
 (Constant.DATABASE_NAME,

 result['Database']['KmsKeyId']))
        except self.client.exceptions.ResourceNotFoundException:
            print("Database doesn't exist")
        except Exception as err:
            print("Update database failed:", err)
```

Node.js

次のスニペットは JavaScript v3 AWSSDKに を使用します。クライアントのインストール方法と使用状況の詳細については、 JavaScript 「v3 の Timestream Write Client -AWSSDK」を参照してください。

クラス UpdateDatabaseCommand と も参照してくださいUpdateDatabase。

```
import { TimestreamWriteClient, UpdateDatabaseCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });
let updatedKmsKeyId = "<updatedKmsKeyId>";

const params = {
    DatabaseName: "testDbFromNode",
    KmsKeyId: updatedKmsKeyId
};

const command = new UpdateDatabaseCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Database ${data.Database.DatabaseName} updated kmsKeyId to
 ${updatedKmsKeyId}`);
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log("Database doesn't exist.");
    } else {
        console.log("Update database failed.", error);
    }
}
```

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
async function updateDatabase(updatedKmsKeyId) {

    if (updatedKmsKeyId === undefined) {
        console.log("Skipping UpdateDatabase; KmsKeyId was not given");
        return;
    }
    console.log("Updating Database");
    const params = {
        DatabaseName: constants.DATABASE_NAME,
        KmsKeyId: updatedKmsKeyId
    }

    const promise = writeClient.updateDatabase(params).promise();

    await promise.then(
        (data) => {
            console.log(`Database ${data.Database.DatabaseName} updated kmsKeyId to
 ${updatedKmsKeyId}`);
        },
        (err) => {
            if (err.code === 'ResourceNotFoundException') {
                console.log("Database doesn't exist.");
            } else {
                console.log("Update database failed.", err);
            }
        }
    );
}
```

.NET

```
        public async Task UpdateDatabase(String updatedKmsKeyId)
        {
            Console.WriteLine("Updating Database");

            try
            {
                var updateDatabaseRequest = new UpdateDatabaseRequest
                {
```

```
                    DatabaseName = Constants.DATABASE_NAME,
                    KmsKeyId = updatedKmsKeyId
                };
                UpdateDatabaseResponse response = await
writeClient.UpdateDatabaseAsync(updateDatabaseRequest);
                Console.WriteLine($"Database {Constants.DATABASE_NAME} updated with
KmsKeyId {updatedKmsKeyId}");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine("Database does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Update database failed: " + e.ToString());
            }

        }

        private void PrintDatabases(List<Database> databases)
        {
            foreach (Database database in databases)
                Console.WriteLine($"Database:{database.DatabaseName}");
        }
```

データベースの削除

次のコードスニペットを使用してデータベースを削除できます。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいていますGitHub。サンプルアプリケーションの使用を開始する方法の詳細については、「」を参照してくださいサンプルアプリケーション。

Java

```
    public void deleteDatabase() {
        System.out.println("Deleting database");
```

```
        final DeleteDatabaseRequest deleteDatabaseRequest = new
DeleteDatabaseRequest();
        deleteDatabaseRequest.setDatabaseName(DATABASE_NAME);
        try {
            DeleteDatabaseResult result =
                    amazonTimestreamWrite.deleteDatabase(deleteDatabaseRequest);
            System.out.println("Delete database status: " +
result.getSdkHttpMetadata().getHttpStatusCode());
        } catch (final ResourceNotFoundException e) {
            System.out.println("Database " + DATABASE_NAME + " doesn't exist = " +
e);
            throw e;
        } catch (final Exception e) {
            System.out.println("Could not delete Database " + DATABASE_NAME + " = "
+ e);
            throw e;
        }
    }
```

Java v2

```
    public void deleteDatabase() {
        System.out.println("Deleting database");
        final DeleteDatabaseRequest deleteDatabaseRequest = new
DeleteDatabaseRequest();
        deleteDatabaseRequest.setDatabaseName(DATABASE_NAME);
        try {
            DeleteDatabaseResult result =
                    amazonTimestreamWrite.deleteDatabase(deleteDatabaseRequest);
            System.out.println("Delete database status: " +
result.getSdkHttpMetadata().getHttpStatusCode());
        } catch (final ResourceNotFoundException e) {
            System.out.println("Database " + DATABASE_NAME + " doesn't exist = " +
e);
            throw e;
        } catch (final Exception e) {
            System.out.println("Could not delete Database " + DATABASE_NAME + " = "
+ e);
            throw e;
        }
    }
```

Go

```
deleteDatabaseInput := &timestreamwrite.DeleteDatabaseInput{
        DatabaseName:   aws.String(*databaseName),
    }

    _, err = writeSvc.DeleteDatabase(deleteDatabaseInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Database deleted:", *databaseName)
    }
```

Python

```
    def delete_database(self):
        print("Deleting Database")
        try:
            result =
 self.client.delete_database(DatabaseName=Constant.DATABASE_NAME)
            print("Delete database status [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
        except self.client.exceptions.ResourceNotFoundException:
            print("database [%s] doesn't exist" % Constant.DATABASE_NAME)
        except Exception as err:
            print("Delete database failed:", err)
```

Node.js

次のスニペットは JavaScript v3 AWSSDKに を使用します。クライアントのインストール方法と使用状況の詳細については、 JavaScript 「v3 AWSSDKの Timestream Write Client -」を参照してください。

クラス DeleteDatabaseCommand と も参照してくださいDeleteDatabase。

```
import { TimestreamWriteClient, DeleteDatabaseCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode"
```

```
};

const command = new DeleteDatabaseCommand(params);

try {
    const data = await writeClient.send(command);
    console.log("Deleted database");
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log(`Database ${params.DatabaseName} doesn't exists.`);
    } else {
        console.log("Delete database failed.", error);
        throw error;
    }
}
```

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
async function deleteDatabase() {
    console.log("Deleting Database");
    const params = {
        DatabaseName: constants.DATABASE_NAME
    };

    const promise = writeClient.deleteDatabase(params).promise();

    await promise.then(
        function (data) {
            console.log("Deleted database");
         },
        function(err) {
            if (err.code === 'ResourceNotFoundException') {
                console.log(`Database ${params.DatabaseName} doesn't exists.`);
            } else {
                console.log("Delete database failed.", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task DeleteDatabase()
        {
            Console.WriteLine("Deleting database");
            try
            {
                var deleteDatabaseRequest = new DeleteDatabaseRequest
                {
                    DatabaseName = Constants.DATABASE_NAME
                };
                DeleteDatabaseResponse response = await
writeClient.DeleteDatabaseAsync(deleteDatabaseRequest);
                Console.WriteLine($"Database {Constants.DATABASE_NAME} delete
request status:{response.HttpStatusCode}");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine($"Database {Constants.DATABASE_NAME} does not
exists");
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception while deleting database:" +
e.ToString());
            }
        }
```

データベースを一覧表示する

次のコードスニペットを使用してデータベースを一覧表示できます。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいていますGitHub。サンプルアプリケーションの使用を開始する方法の詳細については、「」を参照してくださいサンプルアプリケーション。

Java

```
   public void listDatabases() {
       System.out.println("Listing databases");
       ListDatabasesRequest request = new ListDatabasesRequest();
       ListDatabasesResult result = amazonTimestreamWrite.listDatabases(request);
       final List<Database> databases = result.getDatabases();
       printDatabases(databases);

       String nextToken = result.getNextToken();
       while (nextToken != null && !nextToken.isEmpty()) {
           request.setNextToken(nextToken);
           ListDatabasesResult nextResult =
amazonTimestreamWrite.listDatabases(request);
           final List<Database> nextDatabases = nextResult.getDatabases();
           printDatabases(nextDatabases);
           nextToken = nextResult.getNextToken();
       }
   }

   private void printDatabases(List<Database> databases) {
       for (Database db : databases) {
           System.out.println(db.getDatabaseName());
       }
   }
```

Java v2

```
   public void listDatabases() {
       System.out.println("Listing databases");
       ListDatabasesRequest request =
ListDatabasesRequest.builder().maxResults(2).build();
       ListDatabasesIterable listDatabasesIterable =
timestreamWriteClient.listDatabasesPaginator(request);
       for(ListDatabasesResponse listDatabasesResponse : listDatabasesIterable) {
           final List<Database> databases = listDatabasesResponse.databases();
           databases.forEach(database ->
System.out.println(database.databaseName()));
       }
   }
```

Go

```
// List databases.
    listDatabasesMaxResult := int64(15)

    listDatabasesInput := &timestreamwrite.ListDatabasesInput{
        MaxResults: &listDatabasesMaxResult,
    }

    listDatabasesOutput, err := writeSvc.ListDatabases(listDatabasesInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("List databases is successful, below is the output:")
        fmt.Println(listDatabasesOutput)
    }
```

Python

```
    def list_databases(self):
        print("Listing databases")
        try:
            result = self.client.list_databases(MaxResults=5)
            self._print_databases(result['Databases'])
            next_token = result.get('NextToken', None)
            while next_token:
                result = self.client.list_databases(NextToken=next_token,
 MaxResults=5)
                self._print_databases(result['Databases'])
                next_token = result.get('NextToken', None)
        except Exception as err:
            print("List databases failed:", err)
```

Node.js

次のスニペットは JavaScript v3 AWSSDKに を使用します。クライアントのインストール方法と使用状況の詳細については、 JavaScript 「v3 の Timestream Write Client -AWSSDK」を参照してください。

クラス ListDatabasesCommand と も参照してくださいListDatabases。

```
import { TimestreamWriteClient, ListDatabasesCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    MaxResults: 15
};

const command = new ListDatabasesCommand(params);

getDatabasesList(null);

async function getDatabasesList(nextToken) {
    if (nextToken) {
        params.NextToken = nextToken;
    }

    try {
        const data = await writeClient.send(command);

        data.Databases.forEach(function (database) {
            console.log(database.DatabaseName);
        });

        if (data.NextToken) {
            return getDatabasesList(data.NextToken);
        }
    } catch (error) {
        console.log("Error while listing databases", error);
    }
}
```

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
async function listDatabases() {
    console.log("Listing databases:");
    const databases = await getDatabasesList(null);
    databases.forEach(function(database){
        console.log(database.DatabaseName);
    });
```

```
}

function getDatabasesList(nextToken, databases = []) {
    var params = {
        MaxResults: 15
    };

    if(nextToken) {
        params.NextToken = nextToken;
    }

    return writeClient.listDatabases(params).promise()
        .then(
            (data) => {
                databases.push.apply(databases, data.Databases);
                if (data.NextToken) {
                    return getDatabasesList(data.NextToken, databases);
                } else {
                    return databases;
                }
            },
            (err) => {
                console.log("Error while listing databases", err);
            });
}
```

.NET

```
        public async Task ListDatabases()
        {
            Console.WriteLine("Listing Databases");

            try
            {
                var listDatabasesRequest = new ListDatabasesRequest
                {
                    MaxResults = 5
                };
                ListDatabasesResponse response = await
 writeClient.ListDatabasesAsync(listDatabasesRequest);
                PrintDatabases(response.Databases);
                var nextToken = response.NextToken;
                while (nextToken != null)
```

```
                {
                    listDatabasesRequest.NextToken = nextToken;
                    response = await
writeClient.ListDatabasesAsync(listDatabasesRequest);
                    PrintDatabases(response.Databases);
                    nextToken = response.NextToken;
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("List database failed:" + e.ToString());
            }

        }
```

テーブルの作成

トピック

- メモリストアの書き込み
- マグネティックストアの書き込み

メモリストアの書き込み

次のコードスニペットを使用して、マグネティックストアの書き込みが無効になっているテーブルを作成できます。その結果、メモリストアの保持ウィンドウにのみデータを書き込むことができます。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいていますGitHub。サンプルアプリケーションの開始方法の詳細については、「」を参照してくださいサンプルアプリケーション。

Java

```
    public void createTable() {
        System.out.println("Creating table");
        CreateTableRequest createTableRequest = new CreateTableRequest();
        createTableRequest.setDatabaseName(DATABASE_NAME);
```

```
        createTableRequest.setTableName(TABLE_NAME);
        final RetentionProperties retentionProperties = new RetentionProperties()
                .withMemoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                .withMagneticStoreRetentionPeriodInDays(CT_TTL_DAYS);
        createTableRequest.setRetentionProperties(retentionProperties);

        try {
            amazonTimestreamWrite.createTable(createTableRequest);
            System.out.println("Table [" + TABLE_NAME + "] successfully created.");
        } catch (ConflictException e) {
            System.out.println("Table [" + TABLE_NAME + "] exists on database [" +
DATABASE_NAME + "] . Skipping database creation");
        }
    }
```

Java v2

```
    public void createTable() {
        System.out.println("Creating table");

        final RetentionProperties retentionProperties =
RetentionProperties.builder()
                .memoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                .magneticStoreRetentionPeriodInDays(CT_TTL_DAYS).build();
        final CreateTableRequest createTableRequest = CreateTableRequest.builder()

.databaseName(DATABASE_NAME).tableName(TABLE_NAME).retentionProperties(retentionProperties)

        try {
            timestreamWriteClient.createTable(createTableRequest);
            System.out.println("Table [" + TABLE_NAME + "] successfully created.");
        } catch (ConflictException e) {
            System.out.println("Table [" + TABLE_NAME + "] exists on database [" +
DATABASE_NAME + "] . Skipping database creation");
        }
    }
```

Go

```
// Create table.
    createTableInput := &timestreamwrite.CreateTableInput{
        DatabaseName: aws.String(*databaseName),
        TableName:    aws.String(*tableName),
```

```
    }
    _, err = writeSvc.CreateTable(createTableInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Create table is successful")
    }
```

Python

```
    def create_table(self):
        print("Creating table")
        retention_properties = {
            'MemoryStoreRetentionPeriodInHours': Constant.HT_TTL_HOURS,
            'MagneticStoreRetentionPeriodInDays': Constant.CT_TTL_DAYS
        }
        try:
            self.client.create_table(DatabaseName=Constant.DATABASE_NAME,
TableName=Constant.TABLE_NAME,
                                     RetentionProperties=retention_properties)
            print("Table [%s] successfully created." % Constant.TABLE_NAME)
        except self.client.exceptions.ConflictException:
            print("Table [%s] exists on database [%s]. Skipping table creation" % (
                Constant.TABLE_NAME, Constant.DATABASE_NAME))
        except Exception as err:
            print("Create table failed:", err)
```

Node.js

次のスニペットは JavaScript v3 AWSSDKに を使用します。クライアントのインストール方法と使用状況の詳細については、 JavaScript 「v3 AWSSDKの Timestream Write Client -」を参照してください。

クラス CreateTableCommand と も参照してくださいCreateTable。

```
import { TimestreamWriteClient, CreateTableCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode",
```

```
    TableName: "testTableFromNode",
    RetentionProperties: {
        MemoryStoreRetentionPeriodInHours: 24,
        MagneticStoreRetentionPeriodInDays: 365
    }
};

const command = new CreateTableCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Table ${data.Table.TableName} created successfully`);
} catch (error) {
    if (error.code === 'ConflictException') {
        console.log(`Table ${params.TableName} already exists on db
 ${params.DatabaseName}. Skipping creation.`);
    } else {
        console.log("Error creating table. ", error);
        throw error;
    }
}
```

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
async function createTable() {
    console.log("Creating Table");
    const params = {
        DatabaseName: constants.DATABASE_NAME,
        TableName: constants.TABLE_NAME,
        RetentionProperties: {
            MemoryStoreRetentionPeriodInHours: constants.HT_TTL_HOURS,
            MagneticStoreRetentionPeriodInDays: constants.CT_TTL_DAYS
        }
    };

    const promise = writeClient.createTable(params).promise();

    await promise.then(
        (data) => {
            console.log(`Table ${data.Table.TableName} created successfully`);
        },
```

```
        (err) => {
            if (err.code === 'ConflictException') {
                console.log(`Table ${params.TableName} already exists on db
 ${params.DatabaseName}. Skipping creation.`);
            } else {
                console.log("Error creating table. ", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task CreateTable()
        {
            Console.WriteLine("Creating Table");

            try
            {
                var createTableRequest = new CreateTableRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    TableName = Constants.TABLE_NAME,
                    RetentionProperties = new RetentionProperties
                    {
                        MagneticStoreRetentionPeriodInDays = Constants.CT_TTL_DAYS,
                        MemoryStoreRetentionPeriodInHours = Constants.HT_TTL_HOURS
                    }
                };
                CreateTableResponse response = await
 writeClient.CreateTableAsync(createTableRequest);
                Console.WriteLine($"Table {Constants.TABLE_NAME} created");
            }
            catch (ConflictException)
            {
                Console.WriteLine("Table already exists.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Create table failed:" + e.ToString());
            }
```

```
        }
```

マグネティックストアの書き込み

次のコードスニペットを使用して、マグネティックストア書き込みを有効にしたテーブルを作成できます。マグネティックストア書き込みを使用すると、メモリストア保持ウィンドウとマグネティックストア保持ウィンドウの両方にデータを書き込むことができます。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいていますGitHub。サンプルアプリケーションの開始方法の詳細については、「」を参照してくださいサンプルアプリケーション。

Java

```
   public void createTable(String databaseName, String tableName) {
       System.out.println("Creating table");
       CreateTableRequest createTableRequest = new CreateTableRequest();
       createTableRequest.setDatabaseName(databaseName);
       createTableRequest.setTableName(tableName);
       final RetentionProperties retentionProperties = new RetentionProperties()
               .withMemoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
               .withMagneticStoreRetentionPeriodInDays(CT_TTL_DAYS);
       createTableRequest.setRetentionProperties(retentionProperties);
       // Enable MagneticStoreWrite
       final MagneticStoreWriteProperties magneticStoreWriteProperties = new
MagneticStoreWriteProperties()
               .withEnableMagneticStoreWrites(true);

createTableRequest.setMagneticStoreWriteProperties(magneticStoreWriteProperties);
       try {
           amazonTimestreamWrite.createTable(createTableRequest);
           System.out.println("Table [" + tableName + "] successfully created.");
       } catch (ConflictException e) {
           System.out.println("Table [" + tableName + "] exists on database [" +
databaseName + "] . Skipping table creation");
           //We do not throw exception here, we use the existing table instead
       }
```

```
    }
```

Java v2

```
    public void createTable(String databaseName, String tableName) {
        System.out.println("Creating table");

        // Enable MagneticStoreWrite
        final MagneticStoreWriteProperties magneticStoreWriteProperties =
                MagneticStoreWriteProperties.builder()
                        .enableMagneticStoreWrites(true)
                        .build();

        CreateTableRequest createTableRequest =
                CreateTableRequest.builder()
                        .databaseName(databaseName)
                        .tableName(tableName)
                        .retentionProperties(RetentionProperties.builder()
                                .memoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                                .magneticStoreRetentionPeriodInDays(CT_TTL_DAYS)
                                .build())
                        .magneticStoreWriteProperties(magneticStoreWriteProperties)
                        .build();
        try {
            timestreamWriteClient.createTable(createTableRequest);
            System.out.println("Table [" + tableName + "] successfully created.");
        } catch (ConflictException e) {
            System.out.println("Table [" + tableName + "] exists in database [" +
 databaseName + "] . Skipping table creation");
        }
    }
```

Go

```
// Create table.
    createTableInput := &timestreamwrite.CreateTableInput{
        DatabaseName: aws.String(*databaseName),
        TableName:    aws.String(*tableName),
    // Enable MagneticStoreWrite
        MagneticStoreWriteProperties: &timestreamwrite.MagneticStoreWriteProperties{
            EnableMagneticStoreWrites: aws.Bool(true),
             },
      }
```

```
    _, err = writeSvc.CreateTable(createTableInput)
```

Python

```
    def create_table(self):
        print("Creating table")
        retention_properties = {
            'MemoryStoreRetentionPeriodInHours': Constant.HT_TTL_HOURS,
            'MagneticStoreRetentionPeriodInDays': Constant.CT_TTL_DAYS
        }
        magnetic_store_write_properties = {
            'EnableMagneticStoreWrites': True
        }
        try:
            self.client.create_table(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                                     RetentionProperties=retention_properties,

 MagneticStoreWriteProperties=magnetic_store_write_properties)
            print("Table [%s] successfully created." % Constant.TABLE_NAME)
        except self.client.exceptions.ConflictException:
            print("Table [%s] exists on database [%s]. Skipping table creation" % (
                Constant.TABLE_NAME, Constant.DATABASE_NAME))
        except Exception as err:
            print("Create table failed:", err)
```

Node.js

```
async function createTable() {
    console.log("Creating Table");

    const params = {
        DatabaseName: constants.DATABASE_NAME,
        TableName: constants.TABLE_NAME,
        RetentionProperties: {
            MemoryStoreRetentionPeriodInHours: constants.HT_TTL_HOURS,
            MagneticStoreRetentionPeriodInDays: constants.CT_TTL_DAYS
        },
        MagneticStoreWriteProperties: {
            EnableMagneticStoreWrites: true
        }
    };
```

```
    const promise = writeClient.createTable(params).promise();

    await promise.then(
        (data) => {
            console.log(`Table ${data.Table.TableName} created successfully`);
        },
        (err) => {
            if (err.code === 'ConflictException') {
                console.log(`Table ${params.TableName} already exists on db
 ${params.DatabaseName}. Skipping creation.`);
            } else {
                console.log("Error creating table. ", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task CreateTable()
        {
            Console.WriteLine("Creating Table");

            try
            {
                var createTableRequest = new CreateTableRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    TableName = Constants.TABLE_NAME,
                    RetentionProperties = new RetentionProperties
                    {
                        MagneticStoreRetentionPeriodInDays = Constants.CT_TTL_DAYS,
                        MemoryStoreRetentionPeriodInHours = Constants.HT_TTL_HOURS
                    },
                    // Enable MagneticStoreWrite
                    MagneticStoreWriteProperties = new MagneticStoreWriteProperties
                    {
                        EnableMagneticStoreWrites = true,
                    }
                };
                CreateTableResponse response = await
 writeClient.CreateTableAsync(createTableRequest);
```

```
                Console.WriteLine($"Table {Constants.TABLE_NAME} created");
            }
            catch (ConflictException)
            {
                Console.WriteLine("Table already exists.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Create table failed:" + e.ToString());
            }

        }
```

テーブルの説明

次のコードスニペットを使用して、テーブルの属性に関する情報を取得できます。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいていますGitHub。サンプルアプリケーションの使用を開始する方法の詳細については、「」を参照してくださいサンプルアプリケーション。

Java

```
   public void describeTable() {
        System.out.println("Describing table");
        final DescribeTableRequest describeTableRequest = new
DescribeTableRequest();
        describeTableRequest.setDatabaseName(DATABASE_NAME);
        describeTableRequest.setTableName(TABLE_NAME);
        try {
            DescribeTableResult result =
amazonTimestreamWrite.describeTable(describeTableRequest);
            String tableId = result.getTable().getArn();
            System.out.println("Table " + TABLE_NAME + " has id " + tableId);
        } catch (final Exception e) {
            System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
            throw e;
        }
```

```
    }
```

Java v2

```
    public void describeTable() {
        System.out.println("Describing table");
        final DescribeTableRequest describeTableRequest =
 DescribeTableRequest.builder()
                .databaseName(DATABASE_NAME).tableName(TABLE_NAME).build();
        try {
            DescribeTableResponse response =
 timestreamWriteClient.describeTable(describeTableRequest);
            String tableId = response.table().arn();
            System.out.println("Table " + TABLE_NAME + " has id " + tableId);
        } catch (final Exception e) {
            System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
            throw e;
        }
    }
```

Go

```
// Describe table.
    describeTableInput := &timestreamwrite.DescribeTableInput{
        DatabaseName: aws.String(*databaseName),
        TableName:    aws.String(*tableName),
    }
    describeTableOutput, err := writeSvc.DescribeTable(describeTableInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Describe table is successful, below is the output:")
        fmt.Println(describeTableOutput)
    }
```

Python

```
    def describe_table(self):
        print("Describing table")
        try:
```

```
            result = self.client.describe_table(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME)
            print("Table [%s] has id [%s]" % (Constant.TABLE_NAME, result['Table']
['Arn']))
        except self.client.exceptions.ResourceNotFoundException:
            print("Table doesn't exist")
        except Exception as err:
            print("Describe table failed:", err)
```

Node.js

次のスニペットは JavaScript v3 AWSSDKに を使用します。クライアントのインストール方法と使用状況の詳細については、 JavaScript 「v3 AWSSDKの Timestream Write Client -」を参照してください。

クラス DescribeTableCommand と も参照してくださいDescribeTable。

```
import { TimestreamWriteClient, DescribeTableCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode",
    TableName: "testTableFromNode"
};

const command = new DescribeTableCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Table ${data.Table.TableName} has id ${data.Table.Arn}`);
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log("Table or Database doesn't exist.");
    } else {
        console.log("Describe table failed.", error);
        throw error;
    }
}
```

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
async function describeTable() {
    console.log("Describing Table");
    const params = {
        DatabaseName: constants.DATABASE_NAME,
        TableName: constants.TABLE_NAME
    };

    const promise = writeClient.describeTable(params).promise();

    await promise.then(
        (data) => {
            console.log(`Table ${data.Table.TableName} has id ${data.Table.Arn}`);
        },
        (err) => {
            if (err.code === 'ResourceNotFoundException') {
                console.log("Table or Database doesn't exists.");
            } else {
                console.log("Describe table failed.", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task DescribeTable()
        {
            Console.WriteLine("Describing Table");

            try
            {
                var describeTableRequest = new DescribeTableRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    TableName = Constants.TABLE_NAME
                };
                DescribeTableResponse response = await
 writeClient.DescribeTableAsync(describeTableRequest);
                Console.WriteLine($"Table {Constants.TABLE_NAME} has id:
{response.Table.Arn}");
            }
            catch (ResourceNotFoundException)
```

```
            {
                Console.WriteLine("Table does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Describe table failed:" + e.ToString());
            }

        }
```

テーブルを更新する

次のコードスニペットを使用してテーブルを更新できます。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいていますGitHub。サンプルアプリケーションの開始方法の詳細については、「」を参照してくださいサンプルアプリケーション。

Java

```
    public void updateTable() {
        System.out.println("Updating table");
        UpdateTableRequest updateTableRequest = new UpdateTableRequest();
        updateTableRequest.setDatabaseName(DATABASE_NAME);
        updateTableRequest.setTableName(TABLE_NAME);

        final RetentionProperties retentionProperties = new RetentionProperties()
                .withMemoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                .withMagneticStoreRetentionPeriodInDays(CT_TTL_DAYS);

        updateTableRequest.setRetentionProperties(retentionProperties);

        amazonTimestreamWrite.updateTable(updateTableRequest);
        System.out.println("Table updated");
    }
```

Java v2

```
    public void updateTable() {
        System.out.println("Updating table");

        final RetentionProperties retentionProperties =
 RetentionProperties.builder()
                .memoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                .magneticStoreRetentionPeriodInDays(CT_TTL_DAYS).build();
        final UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()

 .databaseName(DATABASE_NAME).tableName(TABLE_NAME).retentionProperties(retentionProperties)

        timestreamWriteClient.updateTable(updateTableRequest);
        System.out.println("Table updated");
    }
```

Go

```
// Update table.
    magneticStoreRetentionPeriodInDays := int64(7 * 365)
    memoryStoreRetentionPeriodInHours := int64(24)

    updateTableInput := &timestreamwrite.UpdateTableInput{
        DatabaseName: aws.String(*databaseName),
        TableName:    aws.String(*tableName),
        RetentionProperties: &timestreamwrite.RetentionProperties{
            MagneticStoreRetentionPeriodInDays: &magneticStoreRetentionPeriodInDays,
            MemoryStoreRetentionPeriodInHours:  &memoryStoreRetentionPeriodInHours,
        },
    }
    updateTableOutput, err := writeSvc.UpdateTable(updateTableInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Update table is successful, below is the output:")
        fmt.Println(updateTableOutput)
    }
```

Python

```
    def update_table(self):
        print("Updating table")
        retention_properties = {
            'MemoryStoreRetentionPeriodInHours': Constant.HT_TTL_HOURS,
            'MagneticStoreRetentionPeriodInDays': Constant.CT_TTL_DAYS
        }
        try:
            self.client.update_table(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                                     RetentionProperties=retention_properties)
            print("Table updated.")
        except Exception as err:
            print("Update table failed:", err)
```

Node.js

次のスニペットは JavaScript v3 AWSSDKに を使用します。クライアントのインストール方法と使用状況の詳細については、 JavaScript 「v3 AWSSDKの Timestream Write Client -」を参照してください。

クラス UpdateTableCommand と も参照してくださいUpdateTable。

```
import { TimestreamWriteClient, UpdateTableCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode",
    TableName: "testTableFromNode",
    RetentionProperties: {
        MemoryStoreRetentionPeriodInHours: 24,
        MagneticStoreRetentionPeriodInDays: 180
    }
};

const command = new UpdateTableCommand(params);

try {
    const data = await writeClient.send(command);
    console.log("Table updated")
} catch (error) {
```

```
    console.log("Error updating table. ", error);
}
```

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
async function updateTable() {
    console.log("Updating Table");
    const params = {
        DatabaseName: constants.DATABASE_NAME,
        TableName: constants.TABLE_NAME,
        RetentionProperties: {
            MemoryStoreRetentionPeriodInHours: constants.HT_TTL_HOURS,
            MagneticStoreRetentionPeriodInDays: constants.CT_TTL_DAYS
        }
    };

    const promise = writeClient.updateTable(params).promise();

    await promise.then(
        (data) => {
            console.log("Table updated")
        },
        (err) => {
            console.log("Error updating table. ", err);
            throw err;
        }
    );
}
```

.NET

```
        public async Task UpdateTable()
        {
            Console.WriteLine("Updating Table");

            try
            {
                var updateTableRequest = new UpdateTableRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    TableName = Constants.TABLE_NAME,
```

```
                    RetentionProperties = new RetentionProperties
                    {
                        MagneticStoreRetentionPeriodInDays = Constants.CT_TTL_DAYS,
                        MemoryStoreRetentionPeriodInHours = Constants.HT_TTL_HOURS
                    }
                };
                UpdateTableResponse response = await
writeClient.UpdateTableAsync(updateTableRequest);
                Console.WriteLine($"Table {Constants.TABLE_NAME} updated");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine("Table does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Update table failed:" + e.ToString());
            }

        }
```

テーブルを削除する

次のコードスニペットを使用してテーブルを削除できます。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいていますGitHub。サンプルアプリケーションの使用を開始する方法の詳細については、「」を参照してくださいサンプルアプリケーション。

Java

```
    public void deleteTable() {
        System.out.println("Deleting table");
        final DeleteTableRequest deleteTableRequest = new DeleteTableRequest();
        deleteTableRequest.setDatabaseName(DATABASE_NAME);
        deleteTableRequest.setTableName(TABLE_NAME);
        try {
            DeleteTableResult result =
```

```
                    amazonTimestreamWrite.deleteTable(deleteTableRequest);
            System.out.println("Delete table status: " +
 result.getSdkHttpMetadata().getHttpStatusCode());
        } catch (final ResourceNotFoundException e) {
            System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
            throw e;
        } catch (final Exception e) {
            System.out.println("Could not delete table " + TABLE_NAME + " = " + e);
            throw e;
        }
    }
```

Java v2

```
    public void deleteTable() {
        System.out.println("Deleting table");
        final DeleteTableRequest deleteTableRequest = DeleteTableRequest.builder()
                .databaseName(DATABASE_NAME).tableName(TABLE_NAME).build();
        try {
            DeleteTableResponse response =
                    timestreamWriteClient.deleteTable(deleteTableRequest);
            System.out.println("Delete table status: " +
 response.sdkHttpResponse().statusCode());
        } catch (final ResourceNotFoundException e) {
            System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
            throw e;
        } catch (final Exception e) {
            System.out.println("Could not delete table " + TABLE_NAME + " = " + e);
            throw e;
        }
    }
```

Go

```
deleteTableInput := &timestreamwrite.DeleteTableInput{
        DatabaseName:   aws.String(*databaseName),
        TableName:    aws.String(*tableName),
    }
    _, err = writeSvc.DeleteTable(deleteTableInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
```

```
    } else {
        fmt.Println("Table deleted", *tableName)
    }
```

Python

```
    def delete_table(self):
        print("Deleting Table")
        try:
            result = self.client.delete_table(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME)
            print("Delete table status [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
        except self.client.exceptions.ResourceNotFoundException:
            print("Table [%s] doesn't exist" % Constant.TABLE_NAME)
        except Exception as err:
            print("Delete table failed:", err)
```

Node.js

次のスニペットは JavaScript v3 AWSSDKに を使用します。クライアントのインストール方法と使用状況の詳細については、 JavaScript 「v3 の Timestream Write Client -AWSSDK」を参照してください。

クラス DeleteTableCommand と も参照してくださいDeleteTable。

```
import { TimestreamWriteClient, DeleteTableCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode",
    TableName: "testTableFromNode"
};

const command = new DeleteTableCommand(params);

try {
    const data = await writeClient.send(command);
    console.log("Deleted table");
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
```

```
        console.log(`Table ${params.TableName} or Database ${params.DatabaseName}
 doesn't exist.`);
    } else {
        console.log("Delete table failed.", error);
        throw error;
    }
}
```

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
async function deleteTable() {
    console.log("Deleting Table");
    const params = {
        DatabaseName: constants.DATABASE_NAME,
        TableName: constants.TABLE_NAME
    };

    const promise = writeClient.deleteTable(params).promise();

    await promise.then(
        function (data) {
            console.log("Deleted table");
        },
        function(err) {
            if (err.code === 'ResourceNotFoundException') {
                console.log(`Table ${params.TableName} or Database
 ${params.DatabaseName} doesn't exists.`);
            } else {
                console.log("Delete table failed.", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task DeleteTable()
        {
            Console.WriteLine("Deleting table");
            try
```

```
            {
                var deleteTableRequest = new DeleteTableRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    TableName = Constants.TABLE_NAME
                };
                DeleteTableResponse response = await
writeClient.DeleteTableAsync(deleteTableRequest);
                Console.WriteLine($"Table {Constants.TABLE_NAME} delete request
status: {response.HttpStatusCode}");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine($"Table {Constants.TABLE_NAME} does not exists");
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception while deleting table:" + e.ToString());
            }
        }
```

テーブルの一覧表示

次のコードスニペットを使用してテーブルを一覧表示できます。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいていますGitHub。サンプルアプリケーションの使用を開始する方法の詳細については、「」を参照してくださいサンプルアプリケーション。

Java

```
    public void listTables() {
        System.out.println("Listing tables");
        ListTablesRequest request = new ListTablesRequest();
        request.setDatabaseName(DATABASE_NAME);
        ListTablesResult result = amazonTimestreamWrite.listTables(request);
        printTables(result.getTables());
```

```
        String nextToken = result.getNextToken();
        while (nextToken != null && !nextToken.isEmpty()) {
            request.setNextToken(nextToken);
            ListTablesResult nextResult = amazonTimestreamWrite.listTables(request);

            printTables(nextResult.getTables());
            nextToken = nextResult.getNextToken();
        }
    }

     private void printTables(List<Table> tables) {
        for (Table table : tables) {
            System.out.println(table.getTableName());
        }
    }
```

Java v2

```
    public void listTables() {
        System.out.println("Listing tables");
        ListTablesRequest request =
 ListTablesRequest.builder().databaseName(DATABASE_NAME).maxResults(2).build();
        ListTablesIterable listTablesIterable =
 timestreamWriteClient.listTablesPaginator(request);
        for(ListTablesResponse listTablesResponse : listTablesIterable) {
            final List<Table> tables = listTablesResponse.tables();
            tables.forEach(table -> System.out.println(table.tableName()));
        }
    }
```

Go

```
listTablesMaxResult := int64(15)

    listTablesInput := &timestreamwrite.ListTablesInput{
        DatabaseName: aws.String(*databaseName),
        MaxResults:   &listTablesMaxResult,
    }
    listTablesOutput, err := writeSvc.ListTables(listTablesInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
```

```
    } else {
        fmt.Println("List tables is successful, below is the output:")
        fmt.Println(listTablesOutput)
    }
```

Python

```
    def list_tables(self):
        print("Listing tables")
        try:
            result = self.client.list_tables(DatabaseName=Constant.DATABASE_NAME,
MaxResults=5)
            self.__print_tables(result['Tables'])
            next_token = result.get('NextToken', None)
            while next_token:
                result =
self.client.list_tables(DatabaseName=Constant.DATABASE_NAME,
                                                    NextToken=next_token, MaxResults=5)
                self.__print_tables(result['Tables'])
                next_token = result.get('NextToken', None)
        except Exception as err:
            print("List tables failed:", err)
```

Node.js

次のスニペットは JavaScript v3 AWSSDKに を使用します。クライアントのインストール方法と使用状況の詳細については、 JavaScript 「v3 AWSSDKの Timestream Write Client -」を参照してください。

クラス ListTablesCommand と も参照してくださいListTables。

```
import { TimestreamWriteClient, ListTablesCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode",
    MaxResults: 15
};

const command = new ListTablesCommand(params);

getTablesList(null);
```

```
async function getTablesList(nextToken) {
    if (nextToken) {
        params.NextToken = nextToken;
    }

    try {
        const data = await writeClient.send(command);

        data.Tables.forEach(function (table) {
            console.log(table.TableName);
        });

        if (data.NextToken) {
            return getTablesList(data.NextToken);
        }
    } catch (error) {
        console.log("Error while listing tables", error);
    }
}
```

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
async function listTables() {
    console.log("Listing tables:");
    const tables = await getTablesList(null);
    tables.forEach(function(table){
        console.log(table.TableName);
    });
}

function getTablesList(nextToken, tables = []) {
    var params = {
        DatabaseName: constants.DATABASE_NAME,
        MaxResults: 15
    };

    if(nextToken) {
        params.NextToken = nextToken;
    }
```

```
    return writeClient.listTables(params).promise()
        .then(
            (data) => {
                tables.push.apply(tables, data.Tables);
                if (data.NextToken) {
                    return getTablesList(data.NextToken, tables);
                } else {
                    return tables;
                }
            },
            (err) => {
                console.log("Error while listing databases", err);
            });
}
```

.NET

```
        public async Task ListTables()
        {
            Console.WriteLine("Listing Tables");

            try
            {
                var listTablesRequest = new ListTablesRequest
                {
                    MaxResults = 5,
                    DatabaseName = Constants.DATABASE_NAME
                };
                ListTablesResponse response = await
 writeClient.ListTablesAsync(listTablesRequest);
                PrintTables(response.Tables);
                string nextToken = response.NextToken;
                while (nextToken != null)
                {
                    listTablesRequest.NextToken = nextToken;
                    response = await writeClient.ListTablesAsync(listTablesRequest);
                    PrintTables(response.Tables);
                    nextToken = response.NextToken;
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("List table failed:" + e.ToString());
```

```
            }

        }

        private void PrintTables(List<Table> tables)
        {
            foreach (Table table in tables)
                Console.WriteLine($"Table: {table.TableName}");
        }
```

書き込みデータ (挿入とアップサート)

トピック

- レコードのバッチの書き込み
- 共通の属性を持つレコードのバッチの書き込み
- レコードの更新
- マルチメジャー属性の例
- 書き込み失敗の処理

レコードのバッチの書き込み

次のコードスニペットを使用して、Amazon Timestream テーブルにデータを書き込むことができます。データをバッチで書き込むと、書き込みのコストを最適化できます。詳細については、「書き込み数の計算」を参照してください。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいていますGitHub。サンプルアプリケーションの使用を開始する方法の詳細については、「」を参照してくださいサンプルアプリケーション。

Java

```
public void writeRecords() {
  System.out.println("Writing records");
  // Specify repeated values for all records
  List<Record> records = new ArrayList<>();
```

```
    final long time = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = new Dimension().withName("region").withValue("us-
east-1");
    final Dimension az = new Dimension().withName("az").withValue("az1");
    final Dimension hostname = new
 Dimension().withName("hostname").withValue("host1");

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record cpuUtilization = new Record()
        .withDimensions(dimensions)
        .withMeasureName("cpu_utilization")
        .withMeasureValue("13.5")
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time));
    Record memoryUtilization = new Record()
        .withDimensions(dimensions)
        .withMeasureName("memory_utilization")
        .withMeasureValue("40")
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time));

    records.add(cpuUtilization);
    records.add(memoryUtilization);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withRecords(records);

    try {
      WriteRecordsResult writeRecordsResult =
 amazonTimestreamWrite.writeRecords(writeRecordsRequest);
      System.out.println("WriteRecords Status: " +
 writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.getRejectedRecords()) {
        System.out.println("Rejected Index " + rejectedRecord.getRecordIndex() + ":
 "
```

```
            + rejectedRecord.getReason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

Java v2

```
  public void writeRecords() {
    System.out.println("Writing records");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = Dimension.builder().name("region").value("us-
east-1").build();
    final Dimension az = Dimension.builder().name("az").value("az1").build();
    final Dimension hostname =
 Dimension.builder().name("hostname").value("host1").build();

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record cpuUtilization = Record.builder()
        .dimensions(dimensions)
        .measureValueType(MeasureValueType.DOUBLE)
        .measureName("cpu_utilization")
        .measureValue("13.5")
        .time(String.valueOf(time)).build();

    Record memoryUtilization = Record.builder()
        .dimensions(dimensions)
        .measureValueType(MeasureValueType.DOUBLE)
        .measureName("memory_utilization")
        .measureValue("40")
        .time(String.valueOf(time)).build();

    records.add(cpuUtilization);
    records.add(memoryUtilization);
```

```
    WriteRecordsRequest writeRecordsRequest = WriteRecordsRequest.builder()
        .databaseName(DATABASE_NAME).tableName(TABLE_NAME).records(records).build();

    try {
      WriteRecordsResponse writeRecordsResponse =
 timestreamWriteClient.writeRecords(writeRecordsRequest);
      System.out.println("WriteRecords Status: " +
 writeRecordsResponse.sdkHttpResponse().statusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.rejectedRecords()) {
        System.out.println("Rejected Index " + rejectedRecord.recordIndex() + ": "
            + rejectedRecord.reason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

Go

```
now := time.Now()
currentTimeInSeconds := now.Unix()
writeRecordsInput := &timestreamwrite.WriteRecordsInput{
  DatabaseName: aws.String(*databaseName),
  TableName:  aws.String(*tableName),
  Records: []*timestreamwrite.Record{
    &timestreamwrite.Record{
      Dimensions: []*timestreamwrite.Dimension{
        &timestreamwrite.Dimension{
          Name:  aws.String("region"),
          Value: aws.String("us-east-1"),
        },
        &timestreamwrite.Dimension{
          Name:  aws.String("az"),
          Value: aws.String("az1"),
        },
        &timestreamwrite.Dimension{
          Name:  aws.String("hostname"),
          Value: aws.String("host1"),
        },
```

```
      },
      MeasureName:    aws.String("cpu_utilization"),
      MeasureValue:   aws.String("13.5"),
      MeasureValueType: aws.String("DOUBLE"),
      Time:       aws.String(strconv.FormatInt(currentTimeInSeconds, 10)),
      TimeUnit:  aws.String("SECONDS"),
    },
    &timestreamwrite.Record{
      Dimensions: []*timestreamwrite.Dimension{
        &timestreamwrite.Dimension{
          Name:  aws.String("region"),
          Value: aws.String("us-east-1"),
        },
        &timestreamwrite.Dimension{
          Name:  aws.String("az"),
          Value: aws.String("az1"),
        },
        &timestreamwrite.Dimension{
          Name:  aws.String("hostname"),
          Value: aws.String("host1"),
        },
      },
      MeasureName:    aws.String("memory_utilization"),
      MeasureValue:   aws.String("40"),
      MeasureValueType: aws.String("DOUBLE"),
      Time:       aws.String(strconv.FormatInt(currentTimeInSeconds, 10)),
      TimeUnit:  aws.String("SECONDS"),
    },
  },
}

_, err = writeSvc.WriteRecords(writeRecordsInput)

if err != nil {
  fmt.Println("Error:")
  fmt.Println(err)
} else {
  fmt.Println("Write records is successful")
}
```

Python

```
  def write_records(self):
```

```
    print("Writing records")
    current_time = self._current_milli_time()

    dimensions = [
      {'Name': 'region', 'Value': 'us-east-1'},
      {'Name': 'az', 'Value': 'az1'},
      {'Name': 'hostname', 'Value': 'host1'}
    ]

    cpu_utilization = {
      'Dimensions': dimensions,
      'MeasureName': 'cpu_utilization',
      'MeasureValue': '13.5',
      'MeasureValueType': 'DOUBLE',
      'Time': current_time
    }

    memory_utilization = {
      'Dimensions': dimensions,
      'MeasureName': 'memory_utilization',
      'MeasureValue': '40',
      'MeasureValueType': 'DOUBLE',
      'Time': current_time
    }

    records = [cpu_utilization, memory_utilization]

    try:
      result = self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                         Records=records, CommonAttributes={})
      print("WriteRecords Status: [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)

  @staticmethod
  def _print_rejected_records_exceptions(err):
    print("RejectedRecords: ", err)
    for rr in err.response["RejectedRecords"]:
      print("Rejected Index " + str(rr["RecordIndex"]) + ": " + rr["Reason"])
      if "ExistingVersion" in rr:
```

```
        print("Rejected record existing version: ", rr["ExistingVersion"])

  @staticmethod
  def _current_milli_time():
    return str(int(round(time.time() * 1000)))
```

Node.js

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
async function writeRecords() {
  console.log("Writing records");
  const currentTime = Date.now().toString(); // Unix time in milliseconds

  const dimensions = [
    {'Name': 'region', 'Value': 'us-east-1'},
    {'Name': 'az', 'Value': 'az1'},
    {'Name': 'hostname', 'Value': 'host1'}
  ];

  const cpuUtilization = {
    'Dimensions': dimensions,
    'MeasureName': 'cpu_utilization',
    'MeasureValue': '13.5',
    'MeasureValueType': 'DOUBLE',
    'Time': currentTime.toString()
  };

  const memoryUtilization = {
    'Dimensions': dimensions,
    'MeasureName': 'memory_utilization',
    'MeasureValue': '40',
    'MeasureValueType': 'DOUBLE',
    'Time': currentTime.toString()
  };

  const records = [cpuUtilization, memoryUtilization];

  const params = {
    DatabaseName: constants.DATABASE_NAME,
    TableName: constants.TABLE_NAME,
```

```
    Records: records
  };

  const request = writeClient.writeRecords(params);

  await request.promise().then(
    (data) => {
      console.log("Write records successful");
    },
    (err) => {
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        const responsePayload =
 JSON.parse(request.response.httpResponse.body.toString());
        console.log("RejectedRecords: ", responsePayload.RejectedRecords);
        console.log("Other records were written successfully. ");
      }
    }
  );
}
```

.NET

```
   public async Task WriteRecords()
   {
     Console.WriteLine("Writing records");

     DateTimeOffset now = DateTimeOffset.UtcNow;
     string currentTimeString = (now.ToUnixTimeMilliseconds()).ToString();

     List<Dimension> dimensions = new List<Dimension>{
       new Dimension { Name = "region", Value = "us-east-1" },
       new Dimension { Name = "az", Value = "az1" },
       new Dimension { Name = "hostname", Value = "host1" }
     };

     var cpuUtilization = new Record
     {
       Dimensions = dimensions,
       MeasureName = "cpu_utilization",
       MeasureValue = "13.6",
       MeasureValueType = MeasureValueType.DOUBLE,
       Time = currentTimeString
```

```
    };

    var memoryUtilization = new Record
    {
      Dimensions = dimensions,
      MeasureName = "memory_utilization",
      MeasureValue = "40",
      MeasureValueType = MeasureValueType.DOUBLE,
      Time = currentTimeString
    };


    List<Record> records = new List<Record> {
      cpuUtilization,
      memoryUtilization
    };

    try
    {
      var writeRecordsRequest = new WriteRecordsRequest
      {
        DatabaseName = Constants.DATABASE_NAME,
        TableName = Constants.TABLE_NAME,
        Records = records
      };
      WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
      Console.WriteLine($"Write records status code:
{response.HttpStatusCode.ToString()}");
    }
    catch (RejectedRecordsException e) {
      Console.WriteLine("RejectedRecordsException:" + e.ToString());
      foreach (RejectedRecord rr in e.RejectedRecords) {
        Console.WriteLine("RecordIndex " + rr.RecordIndex + " : " + rr.Reason);
      }
      Console.WriteLine("Other records were written successfully. ");
    }
    catch (Exception e)
    {
      Console.WriteLine("Write records failure:" + e.ToString());
    }
  }
```

共通の属性を持つレコードのバッチの書き込み

時系列データに、多くのデータポイントで共通する測定値やディメンションがある場合は、次の最適化バージョンの writeRecords APIを使用して、 の Timestream にデータを挿入することもできます LiveAnalytics。バッチ処理で一般的な属性を使用すると、 で説明されているように、書き込みのコストをさらに最適化できます書き込み数の計算。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいていますGitHub。サンプルアプリケーションの使用を開始する方法の詳細については、「」を参照してくださいサンプルアプリケーション。

Java

```
  public void writeRecordsWithCommonAttributes() {
    System.out.println("Writing records with extracting common attributes");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = new Dimension().withName("region").withValue("us-
east-1");
    final Dimension az = new Dimension().withName("az").withValue("az1");
    final Dimension hostname = new
 Dimension().withName("hostname").withValue("host1");

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = new Record()
        .withDimensions(dimensions)
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time));

    Record cpuUtilization = new Record()
        .withMeasureName("cpu_utilization")
        .withMeasureValue("13.5");
    Record memoryUtilization = new Record()
```

```
        .withMeasureName("memory_utilization")
        .withMeasureValue("40");

    records.add(cpuUtilization);
    records.add(memoryUtilization);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withCommonAttributes(commonAttributes);
    writeRecordsRequest.setRecords(records);

    try {
      WriteRecordsResult writeRecordsResult =
 amazonTimestreamWrite.writeRecords(writeRecordsRequest);
      System.out.println("writeRecordsWithCommonAttributes Status: " +
 writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.getRejectedRecords()) {
        System.out.println("Rejected Index " + rejectedRecord.getRecordIndex() + ":
 "
            + rejectedRecord.getReason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

Java v2

```
  public void writeRecordsWithCommonAttributes() {
    System.out.println("Writing records with extracting common attributes");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = Dimension.builder().name("region").value("us-
east-1").build();
    final Dimension az = Dimension.builder().name("az").value("az1").build();
```

```
   final Dimension hostname =
Dimension.builder().name("hostname").value("host1").build();

   dimensions.add(region);
   dimensions.add(az);
   dimensions.add(hostname);

   Record commonAttributes = Record.builder()
       .dimensions(dimensions)
       .measureValueType(MeasureValueType.DOUBLE)
       .time(String.valueOf(time)).build();

   Record cpuUtilization = Record.builder()
       .measureName("cpu_utilization")
       .measureValue("13.5").build();
   Record memoryUtilization = Record.builder()
       .measureName("memory_utilization")
       .measureValue("40").build();

   records.add(cpuUtilization);
   records.add(memoryUtilization);

   WriteRecordsRequest writeRecordsRequest = WriteRecordsRequest.builder()
       .databaseName(DATABASE_NAME)
       .tableName(TABLE_NAME)
       .commonAttributes(commonAttributes)
       .records(records).build();

   try {
     WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsRequest);
     System.out.println("writeRecordsWithCommonAttributes Status: " +
writeRecordsResponse.sdkHttpResponse().statusCode());
   } catch (RejectedRecordsException e) {
     System.out.println("RejectedRecords: " + e);
     for (RejectedRecord rejectedRecord : e.rejectedRecords()) {
       System.out.println("Rejected Index " + rejectedRecord.recordIndex() + ": "
           + rejectedRecord.reason());
     }
     System.out.println("Other records were written successfully. ");
   } catch (Exception e) {
     System.out.println("Error: " + e);
   }
```

```
  }
```

Go

```
now = time.Now()
currentTimeInSeconds = now.Unix()
writeRecordsCommonAttributesInput := &timestreamwrite.WriteRecordsInput{
 DatabaseName: aws.String(*databaseName),
 TableName:  aws.String(*tableName),
 CommonAttributes: &timestreamwrite.Record{
  Dimensions: []*timestreamwrite.Dimension{
   &timestreamwrite.Dimension{
    Name:  aws.String("region"),
    Value: aws.String("us-east-1"),
   },
   &timestreamwrite.Dimension{
    Name:  aws.String("az"),
    Value: aws.String("az1"),
   },
   &timestreamwrite.Dimension{
    Name:  aws.String("hostname"),
    Value: aws.String("host1"),
   },
  },
  MeasureValueType: aws.String("DOUBLE"),
  Time:       aws.String(strconv.FormatInt(currentTimeInSeconds, 10)),
  TimeUnit:     aws.String("SECONDS"),
 },
 Records: []*timestreamwrite.Record{
  &timestreamwrite.Record{
   MeasureName:  aws.String("cpu_utilization"),
   MeasureValue: aws.String("13.5"),
  },
  &timestreamwrite.Record{
   MeasureName:  aws.String("memory_utilization"),
   MeasureValue: aws.String("40"),
  },
 },
}

_, err = writeSvc.WriteRecords(writeRecordsCommonAttributesInput)

if err != nil {
```

```
 fmt.Println("Error:")
 fmt.Println(err)
} else {
 fmt.Println("Ingest records is successful")
}
```

Python

```
  def write_records_with_common_attributes(self):
    print("Writing records extracting common attributes")
    current_time = self._current_milli_time()

    dimensions = [
      {'Name': 'region', 'Value': 'us-east-1'},
      {'Name': 'az', 'Value': 'az1'},
      {'Name': 'hostname', 'Value': 'host1'}
    ]

    common_attributes = {
      'Dimensions': dimensions,
      'MeasureValueType': 'DOUBLE',
      'Time': current_time
    }

    cpu_utilization = {
      'MeasureName': 'cpu_utilization',
      'MeasureValue': '13.5'
    }

    memory_utilization = {
      'MeasureName': 'memory_utilization',
      'MeasureValue': '40'
    }

    records = [cpu_utilization, memory_utilization]

    try:
      result = self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                         Records=records, CommonAttributes=common_attributes)
      print("WriteRecords Status: [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
```

```
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)

  @staticmethod
  def _print_rejected_records_exceptions(err):
    print("RejectedRecords: ", err)
    for rr in err.response["RejectedRecords"]:
      print("Rejected Index " + str(rr["RecordIndex"]) + ": " + rr["Reason"])
      if "ExistingVersion" in rr:
        print("Rejected record existing version: ", rr["ExistingVersion"])

  @staticmethod
  def _current_milli_time():
    return str(int(round(time.time() * 1000)))
```

Node.js

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
async function writeRecordsWithCommonAttributes() {
  console.log("Writing records with common attributes");
  const currentTime = Date.now().toString(); // Unix time in milliseconds

  const dimensions = [
    {'Name': 'region', 'Value': 'us-east-1'},
    {'Name': 'az', 'Value': 'az1'},
    {'Name': 'hostname', 'Value': 'host1'}
  ];

  const commonAttributes = {
    'Dimensions': dimensions,
    'MeasureValueType': 'DOUBLE',
    'Time': currentTime.toString()
  };

  const cpuUtilization = {
    'MeasureName': 'cpu_utilization',
    'MeasureValue': '13.5'
  };
```

```
  const memoryUtilization = {
    'MeasureName': 'memory_utilization',
    'MeasureValue': '40'
  };

  const records = [cpuUtilization, memoryUtilization];

  const params = {
    DatabaseName: constants.DATABASE_NAME,
    TableName: constants.TABLE_NAME,
    Records: records,
    CommonAttributes: commonAttributes
  };

  const request = writeClient.writeRecords(params);

  await request.promise().then(
    (data) => {
      console.log("Write records successful");
    },
    (err) => {
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        const responsePayload =
 JSON.parse(request.response.httpResponse.body.toString());
        console.log("RejectedRecords: ", responsePayload.RejectedRecords);
        console.log("Other records were written successfully. ");
      }
    }
  );
}
```

.NET

```
  public async Task WriteRecordsWithCommonAttributes()
  {
    Console.WriteLine("Writing records with common attributes");

    DateTimeOffset now = DateTimeOffset.UtcNow;
    string currentTimeString = (now.ToUnixTimeMilliseconds()).ToString();

    List<Dimension> dimensions = new List<Dimension>{
      new Dimension { Name = "region", Value = "us-east-1" },
```

```
      new Dimension { Name = "az", Value = "az1" },
      new Dimension { Name = "hostname", Value = "host1" }
    };

    var commonAttributes = new Record
    {
      Dimensions = dimensions,
      MeasureValueType = MeasureValueType.DOUBLE,
      Time = currentTimeString
    };

    var cpuUtilization = new Record
    {
      MeasureName = "cpu_utilization",
      MeasureValue = "13.6"
    };

    var memoryUtilization = new Record
    {
      MeasureName = "memory_utilization",
      MeasureValue = "40"
    };


    List<Record> records = new List<Record>();
    records.Add(cpuUtilization);
    records.Add(memoryUtilization);

    try
    {
      var writeRecordsRequest = new WriteRecordsRequest
      {
        DatabaseName = Constants.DATABASE_NAME,
        TableName = Constants.TABLE_NAME,
        Records = records,
        CommonAttributes = commonAttributes
      };
      WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
      Console.WriteLine($"Write records status code:
{response.HttpStatusCode.ToString()}");
    }
    catch (RejectedRecordsException e) {
      Console.WriteLine("RejectedRecordsException:" + e.ToString());
```

```
      foreach (RejectedRecord rr in e.RejectedRecords) {
        Console.WriteLine("RecordIndex " + rr.RecordIndex + " : " + rr.Reason);
      }
      Console.WriteLine("Other records were written successfully. ");
    }
    catch (Exception e)
    {
      Console.WriteLine("Write records failure:" + e.ToString());
    }
  }
```

レコードの更新

Amazon Timestream のデフォルトの書き込みは、最初のライターがセマンティクスを勝ち取り、データは追加としてのみ保存され、重複したレコードは拒否されますが、最後のライターがセマンティクスを勝ち取り、最もバージョンの高いレコードがシステムに格納される Amazon Timestream にデータを書き込む必要があるアプリケーションがあります。また、既存のレコードを更新する必要があるアプリケーションもあります。これらのシナリオに対処するために、Amazon Timestream はデータをアップサートする機能を提供します。Upsert は、レコードが存在しない場合にシステムにレコードを挿入するか、存在する場合にレコードを更新するオペレーションです。

`WriteRecords` リクエストの送信中にレコード定義`Version`に を含めることで、レコードをアップサートできます。Amazon Timestream は、レコードを最も高い のレコードとともに保存しますVersion。以下のコードサンプルは、データを更新する方法を示しています。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいていますGitHub。サンプルアプリケーションの使用を開始する方法の詳細については、「」を参照してくださいサンプルアプリケーション。

Java

```
public void writeRecordsWithUpsert() {
  System.out.println("Writing records with upsert");
  // Specify repeated values for all records
  List<Record> records = new ArrayList<>();
  final long time = System.currentTimeMillis();
```

```
    // To achieve upsert (last writer wins) semantic, one example is to use current
 time as the version if you are writing directly from the data source
    long version = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = new Dimension().withName("region").withValue("us-
east-1");
    final Dimension az = new Dimension().withName("az").withValue("az1");
    final Dimension hostname = new
 Dimension().withName("hostname").withValue("host1");

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = new Record()
        .withDimensions(dimensions)
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time))
        .withVersion(version);

    Record cpuUtilization = new Record()
        .withMeasureName("cpu_utilization")
        .withMeasureValue("13.5");
    Record memoryUtilization = new Record()
        .withMeasureName("memory_utilization")
        .withMeasureValue("40");

    records.add(cpuUtilization);
    records.add(memoryUtilization);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withCommonAttributes(commonAttributes);
    writeRecordsRequest.setRecords(records);

    // write records for first time
    try {
      WriteRecordsResult writeRecordsResult =
 amazonTimestreamWrite.writeRecords(writeRecordsRequest);
      System.out.println("WriteRecords Status for first time: " +
 writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
```

```
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }

    // Successfully retry same writeRecordsRequest with same records and versions,
because writeRecords API is idempotent.
    try {
      WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
      System.out.println("WriteRecords Status for retry: " +
writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }

    // upsert with lower version, this would fail because a higher version is
required to update the measure value.
    version -= 1;
    commonAttributes.setVersion(version);

    cpuUtilization.setMeasureValue("14.5");
    memoryUtilization.setMeasureValue("50");

    List<Record> upsertedRecords = new ArrayList<>();
    upsertedRecords.add(cpuUtilization);
    upsertedRecords.add(memoryUtilization);

    WriteRecordsRequest writeRecordsUpsertRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withCommonAttributes(commonAttributes);
    writeRecordsUpsertRequest.setRecords(upsertedRecords);

    try {
      WriteRecordsResult writeRecordsUpsertResult =
amazonTimestreamWrite.writeRecords(writeRecordsUpsertRequest);
      System.out.println("WriteRecords Status for upsert with lower version: " +
writeRecordsUpsertResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("WriteRecords Status for upsert with lower version: ");
      printRejectedRecordsException(e);
```

```
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }

    // upsert with higher version as new data in generated
    version = System.currentTimeMillis();
    commonAttributes.setVersion(version);

    writeRecordsUpsertRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withCommonAttributes(commonAttributes);
    writeRecordsUpsertRequest.setRecords(upsertedRecords);

    try {
      WriteRecordsResult writeRecordsUpsertResult =
 amazonTimestreamWrite.writeRecords(writeRecordsUpsertRequest);
      System.out.println("WriteRecords Status for upsert with higher version: " +
 writeRecordsUpsertResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

Java v2

```
  public void writeRecordsWithUpsert() {
    System.out.println("Writing records with upsert");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    // To achieve upsert (last writer wins) semantic, one example is to use current
 time as the version if you are writing directly from the data source
    long version = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = Dimension.builder().name("region").value("us-
east-1").build();
    final Dimension az = Dimension.builder().name("az").value("az1").build();
    final Dimension hostname =
 Dimension.builder().name("hostname").value("host1").build();
```

```
   dimensions.add(region);
   dimensions.add(az);
   dimensions.add(hostname);

   Record commonAttributes = Record.builder()
       .dimensions(dimensions)
       .measureValueType(MeasureValueType.DOUBLE)
       .time(String.valueOf(time))
       .version(version)
       .build();

   Record cpuUtilization = Record.builder()
       .measureName("cpu_utilization")
       .measureValue("13.5").build();
   Record memoryUtilization = Record.builder()
       .measureName("memory_utilization")
       .measureValue("40").build();

   records.add(cpuUtilization);
   records.add(memoryUtilization);

   WriteRecordsRequest writeRecordsRequest = WriteRecordsRequest.builder()
       .databaseName(DATABASE_NAME)
       .tableName(TABLE_NAME)
       .commonAttributes(commonAttributes)
       .records(records).build();

   // write records for first time
   try {
     WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsRequest);
     System.out.println("WriteRecords Status for first time: " +
writeRecordsResponse.sdkHttpResponse().statusCode());
   } catch (RejectedRecordsException e) {
     printRejectedRecordsException(e);
   } catch (Exception e) {
     System.out.println("Error: " + e);
   }

   // Successfully retry same writeRecordsRequest with same records and versions,
because writeRecords API is idempotent.
   try {
```

```
     WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsRequest);
     System.out.println("WriteRecords Status for retry: " +
writeRecordsResponse.sdkHttpResponse().statusCode());
   } catch (RejectedRecordsException e) {
     printRejectedRecordsException(e);
   } catch (Exception e) {
     System.out.println("Error: " + e);
   }

   // upsert with lower version, this would fail because a higher version is
required to update the measure value.
   version -= 1;
   commonAttributes = Record.builder()
       .dimensions(dimensions)
       .measureValueType(MeasureValueType.DOUBLE)
       .time(String.valueOf(time))
       .version(version)
       .build();

   cpuUtilization = Record.builder()
       .measureName("cpu_utilization")
       .measureValue("14.5").build();
   memoryUtilization = Record.builder()
       .measureName("memory_utilization")
       .measureValue("50").build();

   List<Record> upsertedRecords = new ArrayList<>();
   upsertedRecords.add(cpuUtilization);
   upsertedRecords.add(memoryUtilization);

   WriteRecordsRequest writeRecordsUpsertRequest = WriteRecordsRequest.builder()
       .databaseName(DATABASE_NAME)
       .tableName(TABLE_NAME)
       .commonAttributes(commonAttributes)
       .records(upsertedRecords).build();

   try {
     WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsUpsertRequest);
     System.out.println("WriteRecords Status for upsert with lower version: " +
writeRecordsResponse.sdkHttpResponse().statusCode());
   } catch (RejectedRecordsException e) {
     System.out.println("WriteRecords Status for upsert with lower version: ");
```

```
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }

    // upsert with higher version as new data in generated
    version = System.currentTimeMillis();
    commonAttributes = Record.builder()
        .dimensions(dimensions)
        .measureValueType(MeasureValueType.DOUBLE)
        .time(String.valueOf(time))
        .version(version)
        .build();

    writeRecordsUpsertRequest = WriteRecordsRequest.builder()
        .databaseName(DATABASE_NAME)
        .tableName(TABLE_NAME)
        .commonAttributes(commonAttributes)
        .records(upsertedRecords).build();

    try {
      WriteRecordsResponse writeRecordsUpsertResponse =
 timestreamWriteClient.writeRecords(writeRecordsUpsertRequest);
      System.out.println("WriteRecords Status for upsert with higher version: " +
 writeRecordsUpsertResponse.sdkHttpResponse().statusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

Go

```
// Below code will ingest and upsert cpu_utilization and memory_utilization metric
 for a host on
// region=us-east-1, az=az1, and hostname=host1
fmt.Println("Ingesting records and set version as currentTimeInMills, hit enter to
 continue")
reader.ReadString('\n')

// Get current time in seconds.
now = time.Now()
```

```
currentTimeInSeconds = now.Unix()
// To achieve upsert (last writer wins) semantic, one example is to use current time
 as the version if you are writing directly from the data source
version := time.Now().Round(time.Millisecond).UnixNano() / 1e6   // set version as
 currentTimeInMills

writeRecordsCommonAttributesUpsertInput := &timestreamwrite.WriteRecordsInput{
 DatabaseName: aws.String(*databaseName),
 TableName:  aws.String(*tableName),
 CommonAttributes: &timestreamwrite.Record{
  Dimensions: []*timestreamwrite.Dimension{
   &timestreamwrite.Dimension{
    Name:  aws.String("region"),
    Value: aws.String("us-east-1"),
   },
   &timestreamwrite.Dimension{
    Name:  aws.String("az"),
    Value: aws.String("az1"),
   },
   &timestreamwrite.Dimension{
    Name:  aws.String("hostname"),
    Value: aws.String("host1"),
   },
  },
  MeasureValueType: aws.String("DOUBLE"),
  Time:       aws.String(strconv.FormatInt(currentTimeInSeconds, 10)),
  TimeUnit:  aws.String("SECONDS"),
  Version:      &version,
 },
 Records: []*timestreamwrite.Record{
  &timestreamwrite.Record{
   MeasureName:  aws.String("cpu_utilization"),
   MeasureValue: aws.String("13.5"),
  },
  &timestreamwrite.Record{
   MeasureName:  aws.String("memory_utilization"),
   MeasureValue: aws.String("40"),
  },
 },
}

// write records for first time
_, err = writeSvc.WriteRecords(writeRecordsCommonAttributesUpsertInput)
```

```
if err != nil {
 fmt.Println("Error:")
 fmt.Println(err)
} else {
 fmt.Println("Frist-time write records is successful")
}

fmt.Println("Retry same writeRecordsRequest with same records and versions. Because
 writeRecords API is idempotent, this will success. hit enter to continue")
reader.ReadString('\n')

_, err = writeSvc.WriteRecords(writeRecordsCommonAttributesUpsertInput)

if err != nil {
 fmt.Println("Error:")
 fmt.Println(err)
} else {
 fmt.Println("Retry write records for same request is successful")
}

fmt.Println("Upsert with lower version, this would fail because a higher version is
 required to update the measure value. hit enter to continue")
reader.ReadString('\n')
version -= 1
writeRecordsCommonAttributesUpsertInput.CommonAttributes.Version = &version

updated_cpu_utilization := &timestreamwrite.Record{
 MeasureName:    aws.String("cpu_utilization"),
 MeasureValue:   aws.String("14.5"),
}
updated_memory_utilization := &timestreamwrite.Record{
 MeasureName:    aws.String("memory_utilization"),
 MeasureValue:   aws.String("50"),
}


writeRecordsCommonAttributesUpsertInput.Records = []*timestreamwrite.Record{
 updated_cpu_utilization,
 updated_memory_utilization,
}

_, err = writeSvc.WriteRecords(writeRecordsCommonAttributesUpsertInput)

if err != nil {
```

```
 fmt.Println("Error:")
 fmt.Println(err)
} else {
 fmt.Println("Write records with lower version is successful")
}

fmt.Println("Upsert with higher version as new data in generated, this would
 success. hit enter to continue")
reader.ReadString('\n')

version = time.Now().Round(time.Millisecond).UnixNano() / 1e6  // set version as
 currentTimeInMills
writeRecordsCommonAttributesUpsertInput.CommonAttributes.Version = &version

_, err = writeSvc.WriteRecords(writeRecordsCommonAttributesUpsertInput)

if err != nil {
 fmt.Println("Error:")
 fmt.Println(err)
} else {
 fmt.Println("Write records with higher version is successful")
}
```

Python

```
  def write_records_with_upsert(self):
    print("Writing records with upsert")
    current_time = self._current_milli_time()
    # To achieve upsert (last writer wins) semantic, one example is to use current
 time as the version if you are writing directly from the data source
    version = int(self._current_milli_time())

    dimensions = [
          {'Name': 'region', 'Value': 'us-east-1'},
          {'Name': 'az', 'Value': 'az1'},
          {'Name': 'hostname', 'Value': 'host1'}
        ]

    common_attributes = {
      'Dimensions': dimensions,
      'MeasureValueType': 'DOUBLE',
      'Time': current_time,
      'Version': version
```

```
    }

    cpu_utilization = {
      'MeasureName': 'cpu_utilization',
      'MeasureValue': '13.5'
    }

    memory_utilization = {
      'MeasureName': 'memory_utilization',
      'MeasureValue': '40'
    }

    records = [cpu_utilization, memory_utilization]

    # write records for first time
    try:
      result = self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                         Records=records, CommonAttributes=common_attributes)
      print("WriteRecords Status for first time: [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)

    # Successfully retry same writeRecordsRequest with same records and versions,
 because writeRecords API is idempotent.
    try:
      result = self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                         Records=records, CommonAttributes=common_attributes)
      print("WriteRecords Status for retry: [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)

    # upsert with lower version, this would fail because a higher version is
 required to update the measure value.
    version -= 1
    common_attributes["Version"] = version
```

```
    cpu_utilization["MeasureValue"] = '14.5'
    memory_utilization["MeasureValue"] = '50'

    upsertedRecords = [cpu_utilization, memory_utilization]

    try:
      upsertedResult =
 self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                           Records=upsertedRecords,
 CommonAttributes=common_attributes)
      print("WriteRecords Status for upsert with lower version: [%s]" %
 upsertedResult['ResponseMetadata']['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)


    # upsert with higher version as new data is generated
    version = int(self._current_milli_time())
    common_attributes["Version"] = version

    try:
      upsertedResult =
 self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                           Records=upsertedRecords,
 CommonAttributes=common_attributes)
      print("WriteRecords Upsert Status: [%s]" % upsertedResult['ResponseMetadata']
['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)

  @staticmethod
  def _current_milli_time():
    return str(int(round(time.time() * 1000)))
```

Node.js

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
async function writeRecordsWithUpsert() {
  console.log("Writing records with upsert");
  const currentTime = Date.now().toString(); // Unix time in milliseconds
  // To achieve upsert (last writer wins) semantic, one example is to use current
 time as the version if you are writing directly from the data source
  let version = Date.now();

  const dimensions = [
    {'Name': 'region', 'Value': 'us-east-1'},
    {'Name': 'az', 'Value': 'az1'},
    {'Name': 'hostname', 'Value': 'host1'}
  ];

  const commonAttributes = {
    'Dimensions': dimensions,
    'MeasureValueType': 'DOUBLE',
    'Time': currentTime.toString(),
    'Version': version
  };

  const cpuUtilization = {
    'MeasureName': 'cpu_utilization',
    'MeasureValue': '13.5'
  };

  const memoryUtilization = {
    'MeasureName': 'memory_utilization',
    'MeasureValue': '40'
  };

  const records = [cpuUtilization, memoryUtilization];

  const params = {
    DatabaseName: constants.DATABASE_NAME,
    TableName: constants.TABLE_NAME,
    Records: records,
    CommonAttributes: commonAttributes
```

```
 };

 const request = writeClient.writeRecords(params);

 // write records for first time
 await request.promise().then(
   (data) => {
     console.log("Write records successful for first time.");
   },
   (err) => {
     console.log("Error writing records:", err);
     if (err.code === 'RejectedRecordsException') {
       printRejectedRecordsException(request);
     }
   }
 );

 // Successfully retry same writeRecordsRequest with same records and versions,
because writeRecords API is idempotent.
 await request.promise().then(
   (data) => {
     console.log("Write records successful for retry.");
   },
   (err) => {
     console.log("Error writing records:", err);
     if (err.code === 'RejectedRecordsException') {
       printRejectedRecordsException(request);
     }
   }
 );

 // upsert with lower version, this would fail because a higher version is required
to update the measure value.
 version--;

 const commonAttributesWithLowerVersion = {
   'Dimensions': dimensions,
   'MeasureValueType': 'DOUBLE',
   'Time': currentTime.toString(),
   'Version': version
 };

 const updatedCpuUtilization = {
   'MeasureName': 'cpu_utilization',
```

```
    'MeasureValue': '14.5'
  };

  const updatedMemoryUtilization = {
    'MeasureName': 'memory_utilization',
    'MeasureValue': '50'
  };

  const upsertedRecords = [updatedCpuUtilization, updatedMemoryUtilization];

  const upsertedParamsWithLowerVersion = {
    DatabaseName: constants.DATABASE_NAME,
    TableName: constants.TABLE_NAME,
    Records: upsertedRecords,
    CommonAttributes: commonAttributesWithLowerVersion
  };

  const upsertRequestWithLowerVersion =
writeClient.writeRecords(upsertedParamsWithLowerVersion);

  await upsertRequestWithLowerVersion.promise().then(
    (data) => {
      console.log("Write records for upsert with lower version successful");
    },
    (err) => {
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        printRejectedRecordsException(upsertRequestWithLowerVersion);
      }
    }
  );

  // upsert with higher version as new data in generated
  version = Date.now();

  const commonAttributesWithHigherVersion = {
    'Dimensions': dimensions,
    'MeasureValueType': 'DOUBLE',
    'Time': currentTime.toString(),
    'Version': version
  };

  const upsertedParamsWithHigherVerion = {
    DatabaseName: constants.DATABASE_NAME,
```

```
    TableName: constants.TABLE_NAME,
    Records: upsertedRecords,
    CommonAttributes: commonAttributesWithHigherVersion
  };

  const upsertRequestWithHigherVersion =
 writeClient.writeRecords(upsertedParamsWithHigherVerion);

  await upsertRequestWithHigherVersion.promise().then(
    (data) => {
      console.log("Write records upsert successful with higher version");
    },
    (err) => {
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        printRejectedRecordsException(upsertedParamsWithHigherVerion);
      }
    }
  );

}
```

.NET

```
  public async Task WriteRecordsWithUpsert()
  {
    Console.WriteLine("Writing records with upsert");

    DateTimeOffset now = DateTimeOffset.UtcNow;
    string currentTimeString = (now.ToUnixTimeMilliseconds()).ToString();
    // To achieve upsert (last writer wins) semantic, one example is to use current
 time as the version if you are writing directly from the data source
    long version = now.ToUnixTimeMilliseconds();

    List<Dimension> dimensions = new List<Dimension>{
      new Dimension { Name = "region", Value = "us-east-1" },
      new Dimension { Name = "az", Value = "az1" },
      new Dimension { Name = "hostname", Value = "host1" }
    };

    var commonAttributes = new Record
    {
      Dimensions = dimensions,
```

```
      MeasureValueType = MeasureValueType.DOUBLE,
      Time = currentTimeString,
      Version = version
    };

    var cpuUtilization = new Record
    {
      MeasureName = "cpu_utilization",
      MeasureValue = "13.6"
    };

    var memoryUtilization = new Record
    {
      MeasureName = "memory_utilization",
      MeasureValue = "40"
    };


    List<Record> records = new List<Record>();
    records.Add(cpuUtilization);
    records.Add(memoryUtilization);

    // write records for first time
    try
    {
      var writeRecordsRequest = new WriteRecordsRequest
      {
        DatabaseName = Constants.DATABASE_NAME,
        TableName = Constants.TABLE_NAME,
        Records = records,
        CommonAttributes = commonAttributes
      };
      WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
      Console.WriteLine($"WriteRecords Status for first time:
{response.HttpStatusCode.ToString()}");
    }
    catch (RejectedRecordsException e) {
      PrintRejectedRecordsException(e);
    }
    catch (Exception e)
    {
      Console.WriteLine("Write records failure:" + e.ToString());
    }
```

```
   // Successfully retry same writeRecordsRequest with same records and versions,
because writeRecords API is idempotent.
   try
   {
     var writeRecordsRequest = new WriteRecordsRequest
     {
       DatabaseName = Constants.DATABASE_NAME,
       TableName = Constants.TABLE_NAME,
       Records = records,
       CommonAttributes = commonAttributes
     };
     WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
     Console.WriteLine($"WriteRecords Status for retry:
{response.HttpStatusCode.ToString()}");
   }
   catch (RejectedRecordsException e) {
     PrintRejectedRecordsException(e);
   }
   catch (Exception e)
   {
     Console.WriteLine("Write records failure:" + e.ToString());
   }

   // upsert with lower version, this would fail because a higher version is
required to update the measure value.
   version--;
   Type recordType = typeof(Record);
   recordType.GetProperty("Version").SetValue(commonAttributes, version);
   recordType.GetProperty("MeasureValue").SetValue(cpuUtilization, "14.6");
   recordType.GetProperty("MeasureValue").SetValue(memoryUtilization, "50");

   List<Record> upsertedRecords = new List<Record> {
     cpuUtilization,
     memoryUtilization
   };

   try
   {
     var writeRecordsUpsertRequest = new WriteRecordsRequest
     {
       DatabaseName = Constants.DATABASE_NAME,
       TableName = Constants.TABLE_NAME,
```

```
        Records = upsertedRecords,
        CommonAttributes = commonAttributes
      };
      WriteRecordsResponse upsertResponse = await
writeClient.WriteRecordsAsync(writeRecordsUpsertRequest);
      Console.WriteLine($"WriteRecords Status for upsert with lower version:
{upsertResponse.HttpStatusCode.ToString()}");
    }
    catch (RejectedRecordsException e) {
      PrintRejectedRecordsException(e);
    }
    catch (Exception e)
    {
      Console.WriteLine("Write records failure:" + e.ToString());
    }

    // upsert with higher version as new data in generated
    now = DateTimeOffset.UtcNow;
    version = now.ToUnixTimeMilliseconds();
    recordType.GetProperty("Version").SetValue(commonAttributes, version);

    try
    {
      var writeRecordsUpsertRequest = new WriteRecordsRequest
      {
        DatabaseName = Constants.DATABASE_NAME,
        TableName = Constants.TABLE_NAME,
        Records = upsertedRecords,
        CommonAttributes = commonAttributes
      };
      WriteRecordsResponse upsertResponse = await
writeClient.WriteRecordsAsync(writeRecordsUpsertRequest);
      Console.WriteLine($"WriteRecords Status for upsert with higher version:
{upsertResponse.HttpStatusCode.ToString()}");
    }
    catch (RejectedRecordsException e) {
      PrintRejectedRecordsException(e);
    }
    catch (Exception e)
    {
      Console.WriteLine("Write records failure:" + e.ToString());
    }
  }
```

マルチメジャー属性の例

この例では、マルチムーア属性の書き込みを示しています。マルチメジャー属性は、追跡しているデバイスまたはアプリケーションが複数のメトリクスまたはイベントを同じタイムスタンプで出力する場合に役立ちます。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいていますGitHub。サンプルアプリケーションの使用を開始する方法の詳細については、「」を参照してくださいサンプルアプリケーション。

Java

```
package com.amazonaws.services.timestream;

import static com.amazonaws.services.timestream.Main.DATABASE_NAME;
import static com.amazonaws.services.timestream.Main.REGION;
import static com.amazonaws.services.timestream.Main.TABLE_NAME;

import java.util.ArrayList;
import java.util.List;

import com.amazonaws.services.timestreamwrite.AmazonTimestreamWrite;
import com.amazonaws.services.timestreamwrite.model.Dimension;
import com.amazonaws.services.timestreamwrite.model.MeasureValue;
import com.amazonaws.services.timestreamwrite.model.MeasureValueType;
import com.amazonaws.services.timestreamwrite.model.Record;
import com.amazonaws.services.timestreamwrite.model.RejectedRecordsException;
import com.amazonaws.services.timestreamwrite.model.WriteRecordsRequest;
import com.amazonaws.services.timestreamwrite.model.WriteRecordsResult;


public class multimeasureAttributeExample {
  AmazonTimestreamWrite timestreamWriteClient;

  public multimeasureAttributeExample(AmazonTimestreamWrite client) {
    this.timestreamWriteClient = client;
  }

  public void writeRecordsMultiMeasureValueSingleRecord() {
```

```
    System.out.println("Writing records with multi value attributes");

    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    long version = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = new Dimension().withName("region").withValue(REGION);
    final Dimension az = new Dimension().withName("az").withValue("az1");
    final Dimension hostname = new
Dimension().withName("hostname").withValue("host1");

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = new Record()
        .withDimensions(dimensions)
        .withTime(String.valueOf(time))
        .withVersion(version);

    MeasureValue cpuUtilization = new MeasureValue()
        .withName("cpu_utilization")
        .withType(MeasureValueType.DOUBLE)
        .withValue("13.5");
    MeasureValue memoryUtilization = new MeasureValue()
        .withName("memory_utilization")
        .withType(MeasureValueType.DOUBLE)
        .withValue("40");
    Record computationalResources = new Record()
        .withMeasureName("cpu_memory")
        .withMeasureValues(cpuUtilization, memoryUtilization)
        .withMeasureValueType(MeasureValueType.MULTI);

    records.add(computationalResources);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withCommonAttributes(commonAttributes)
        .withRecords(records);

    // write records for first time
    try {
```

```
      WriteRecordsResult writeRecordResult =
timestreamWriteClient.writeRecords(writeRecordsRequest);
      System.out.println(
          "WriteRecords Status for multi value attributes: " + writeRecordResult
              .getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }

  public void writeRecordsMultiMeasureValueMultipleRecords() {
    System.out.println(
        "Writing records with multi value attributes mixture type");

    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    long version = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = new Dimension().withName("region").withValue(REGION);
    final Dimension az = new Dimension().withName("az").withValue("az1");
    final Dimension hostname = new
Dimension().withName("hostname").withValue("host1");

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = new Record()
        .withDimensions(dimensions)
        .withTime(String.valueOf(time))
        .withVersion(version);

    MeasureValue cpuUtilization = new MeasureValue()
        .withName("cpu_utilization")
        .withType(MeasureValueType.DOUBLE)
        .withValue("13");
    MeasureValue memoryUtilization =new MeasureValue()
        .withName("memory_utilization")
        .withType(MeasureValueType.DOUBLE)
        .withValue("40");
    MeasureValue activeCores = new MeasureValue()
```

```
            .withName("active_cores")
            .withType(MeasureValueType.BIGINT)
            .withValue("4");


        Record computationalResources = new Record()
            .withMeasureName("computational_utilization")
            .withMeasureValues(cpuUtilization, memoryUtilization, activeCores)
            .withMeasureValueType(MeasureValueType.MULTI);

        records.add(computationalResources);

        WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
            .withDatabaseName(DATABASE_NAME)
            .withTableName(TABLE_NAME)
            .withCommonAttributes(commonAttributes)
            .withRecords(records);

        // write records for first time
        try {
          WriteRecordsResult writeRecordResult =
     timestreamWriteClient.writeRecords(writeRecordsRequest);
          System.out.println(
              "WriteRecords Status for multi value attributes: " + writeRecordResult
                  .getSdkHttpMetadata().getHttpStatusCode());
        } catch (RejectedRecordsException e) {
          printRejectedRecordsException(e);
        } catch (Exception e) {
          System.out.println("Error: " + e);
        }
      }

      private void printRejectedRecordsException(RejectedRecordsException e) {
        System.out.println("RejectedRecords: " + e);
        e.getRejectedRecords().forEach(System.out::println);
      }
    }
```

Java v2

```
package com.amazonaws.services.timestream;

import java.util.ArrayList;
```

```
import java.util.List;

import software.amazon.awssdk.services.timestreamwrite.TimestreamWriteClient;
import software.amazon.awssdk.services.timestreamwrite.model.Dimension;
import software.amazon.awssdk.services.timestreamwrite.model.MeasureValue;
import software.amazon.awssdk.services.timestreamwrite.model.MeasureValueType;
import software.amazon.awssdk.services.timestreamwrite.model.Record;
import
 software.amazon.awssdk.services.timestreamwrite.model.RejectedRecordsException;
import software.amazon.awssdk.services.timestreamwrite.model.WriteRecordsRequest;
import software.amazon.awssdk.services.timestreamwrite.model.WriteRecordsResponse;

import static com.amazonaws.services.timestream.Main.DATABASE_NAME;
import static com.amazonaws.services.timestream.Main.TABLE_NAME;


public class multimeasureAttributeExample {

  TimestreamWriteClient timestreamWriteClient;

  public multimeasureAttributeExample(TimestreamWriteClient client) {
    this.timestreamWriteClient = client;
  }

  public void writeRecordsMultiMeasureValueSingleRecord() {
    System.out.println("Writing records with multi value attributes");

    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    long version = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region =
        Dimension.builder().name("region").value("us-east-1").build();
    final Dimension az = Dimension.builder().name("az").value("az1").build();
    final Dimension hostname =
        Dimension.builder().name("hostname").value("host1").build();

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = Record.builder()
        .dimensions(dimensions)
```

```
        .time(String.valueOf(time))
        .version(version)
        .build();

    MeasureValue cpuUtilization = MeasureValue.builder()
        .name("cpu_utilization")
        .type(MeasureValueType.DOUBLE)
        .value("13.5").build();
    MeasureValue memoryUtilization = MeasureValue.builder()
        .name("memory_utilization")
        .type(MeasureValueType.DOUBLE)
        .value("40").build();
    Record computationalResources = Record
        .builder()
        .measureName("cpu_memory")
        .measureValues(cpuUtilization, memoryUtilization)
        .measureValueType(MeasureValueType.MULTI)
        .build();

    records.add(computationalResources);

    WriteRecordsRequest writeRecordsRequest = WriteRecordsRequest.builder()
        .databaseName(DATABASE_NAME)
        .tableName(TABLE_NAME)
        .commonAttributes(commonAttributes)
        .records(records).build();

    // write records for first time
    try {
      WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsRequest);
      System.out.println(
          "WriteRecords Status for multi value attributes: " + writeRecordsResponse
              .sdkHttpResponse()
              .statusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }

  public void writeRecordsMultiMeasureValueMultipleRecords() {
    System.out.println(
```

```
        "Writing records with multi value attributes mixture type");

    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    long version = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region =
        Dimension.builder().name("region").value("us-east-1").build();
    final Dimension az = Dimension.builder().name("az").value("az1").build();
    final Dimension hostname =
        Dimension.builder().name("hostname").value("host1").build();

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = Record.builder()
        .dimensions(dimensions)
        .time(String.valueOf(time))
        .version(version)
        .build();

    MeasureValue cpuUtilization = MeasureValue.builder()
        .name("cpu_utilization")
        .type(MeasureValueType.DOUBLE)
        .value("13.5").build();
    MeasureValue memoryUtilization = MeasureValue.builder()
        .name("memory_utilization")
        .type(MeasureValueType.DOUBLE)
        .value("40").build();
    MeasureValue activeCores = MeasureValue.builder()
        .name("active_cores")
        .type(MeasureValueType.BIGINT)
        .value("4").build();


    Record computationalResources = Record
        .builder()
        .measureName("computational_utilization")
        .measureValues(cpuUtilization, memoryUtilization, activeCores)
        .measureValueType(MeasureValueType.MULTI)
        .build();
```

```
    records.add(computationalResources);

    WriteRecordsRequest writeRecordsRequest = WriteRecordsRequest.builder()
        .databaseName(DATABASE_NAME)
        .tableName(TABLE_NAME)
        .commonAttributes(commonAttributes)
        .records(records).build();

    // write records for first time
    try {
      WriteRecordsResponse writeRecordsResponse =
 timestreamWriteClient.writeRecords(writeRecordsRequest);
      System.out.println(
          "WriteRecords Status for multi value attributes: " + writeRecordsResponse
              .sdkHttpResponse()
              .statusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }

  private void printRejectedRecordsException(RejectedRecordsException e) {
    System.out.println("RejectedRecords: " + e);
    e.rejectedRecords().forEach(System.out::println);
  }
}
```

Go

```
  now := time.Now()
  currentTimeInSeconds := now.Unix()
  writeRecordsInput := &timestreamwrite.WriteRecordsInput{
    DatabaseName: aws.String(*databaseName),
    TableName:  aws.String(*tableName),
    Records: []*timestreamwrite.Record{
    &timestreamwrite.Record{
      Dimensions: []*timestreamwrite.Dimension{
      &timestreamwrite.Dimension{
        Name:  aws.String("region"),
        Value: aws.String("us-east-1"),
      },
```

```
      &timestreamwrite.Dimension{
        Name:  aws.String("az"),
        Value: aws.String("az1"),
      },
      &timestreamwrite.Dimension{
        Name:  aws.String("hostname"),
        Value: aws.String("host1"),
      },
      },
      MeasureName:  aws.String("metrics"),
      MeasureValueType: aws.String("MULTI"),
      Time:     aws.String(strconv.FormatInt(currentTimeInSeconds, 10)),
      TimeUnit:  aws.String("SECONDS"),
      MeasureValues: []*timestreamwrite.MeasureValue{
      &timestreamwrite.MeasureValue{
        Name:  aws.String("cpu_utilization"),
        Value: aws.String("13.5"),
        Type:  aws.String("DOUBLE"),
      },
      &timestreamwrite.MeasureValue{
        Name:  aws.String("memory_utilization"),
        Value: aws.String("40"),
        Type:  aws.String("DOUBLE"),
      },
      },
    },
    },
  }

  _, err = writeSvc.WriteRecords(writeRecordsInput)

  if err != nil {
    fmt.Println("Error:")
    fmt.Println(err)
  } else {
    fmt.Println("Write records is successful")
  }
```

Python

```
import time
import boto3
import psutil
```

```
import os

from botocore.config import Config

DATABASE_NAME = os.environ['DATABASE_NAME']
TABLE_NAME = os.environ['TABLE_NAME']

COUNTRY = "UK"
CITY = "London"
HOSTNAME = "MyHostname" # You can make it dynamic using socket.gethostname()

INTERVAL = 1 # Seconds

def prepare_common_attributes():
  common_attributes = {
    'Dimensions': [
      {'Name': 'country', 'Value': COUNTRY},
      {'Name': 'city', 'Value': CITY},
      {'Name': 'hostname', 'Value': HOSTNAME}
    ],
    'MeasureName': 'utilization',
    'MeasureValueType': 'MULTI'
  }
  return common_attributes


def prepare_record(current_time):
  record = {
    'Time': str(current_time),
    'MeasureValues': []
  }
  return record


def prepare_measure(measure_name, measure_value):
  measure = {
    'Name': measure_name,
    'Value': str(measure_value),
    'Type': 'DOUBLE'
  }
  return measure


def write_records(records, common_attributes):
```

```
  try:
    result = write_client.write_records(DatabaseName=DATABASE_NAME,
                                        TableName=TABLE_NAME,
                                        CommonAttributes=common_attributes,
                                        Records=records)
    status = result['ResponseMetadata']['HTTPStatusCode']
    print("Processed %d records. WriteRecords HTTPStatusCode: %s" %
        (len(records), status))
  except Exception as err:
    print("Error:", err)


if __name__ == '__main__':

  print("writing data to database {} table {}".format(
    DATABASE_NAME, TABLE_NAME))

  session = boto3.Session()
  write_client = session.client('timestream-write', config=Config(
    read_timeout=20, max_pool_connections=5000, retries={'max_attempts': 10}))
  query_client = session.client('timestream-query') # Not used

  common_attributes = prepare_common_attributes()

  records = []

  while True:

    current_time = int(time.time() * 1000)
    cpu_utilization = psutil.cpu_percent()
    memory_utilization = psutil.virtual_memory().percent
    swap_utilization = psutil.swap_memory().percent
    disk_utilization = psutil.disk_usage('/').percent

    record = prepare_record(current_time)
    record['MeasureValues'].append(prepare_measure('cpu', cpu_utilization))
    record['MeasureValues'].append(prepare_measure('memory', memory_utilization))
    record['MeasureValues'].append(prepare_measure('swap', swap_utilization))
    record['MeasureValues'].append(prepare_measure('disk', disk_utilization))

    records.append(record)

    print("records {} - cpu {} - memory {} - swap {} - disk {}".format(
      len(records), cpu_utilization, memory_utilization,
```

```
      swap_utilization, disk_utilization))

    if len(records) == 100:
      write_records(records, common_attributes)
      records = []

    time.sleep(INTERVAL)
```

Node.js

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
async function writeRecords() {
  console.log("Writing records");
  const currentTime = Date.now().toString(); // Unix time in milliseconds

  const dimensions = [
  {'Name': 'region', 'Value': 'us-east-1'},
  {'Name': 'az', 'Value': 'az1'},
  {'Name': 'hostname', 'Value': 'host1'}
  ];

  const record = {
  'Dimensions': dimensions,
  'MeasureName': 'metrics',
  'MeasureValues': [
    {
      'Name': 'cpu_utilization',
      'Value': '40',
      'Type': 'DOUBLE',
    },
    {
      'Name': 'memory_utilization',
      'Value': '13.5',
      'Type': 'DOUBLE',
    },
    ],
    'MeasureValueType': 'MULTI',
    'Time': currentTime.toString()
  }
```

```
    const records = [record];

    const params = {
    DatabaseName: 'DatabaseName',
    TableName: 'TableName',
    Records: records
    };

    const response = await writeClient.writeRecords(params);

    console.log(response);
  }
```

.NET

```
using System;
using System.IO;
using System.Collections.Generic;
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using System.Threading.Tasks;

namespace TimestreamDotNetSample
{
  static class MultiMeasureValueConstants
  {
    public const string MultiMeasureValueSampleDb = "multiMeasureValueSampleDb";
    public const string MultiMeasureValueSampleTable =
 "multiMeasureValueSampleTable";
  }

  public class MultiValueAttributesExample
  {
    private readonly AmazonTimestreamWriteClient writeClient;

    public MultiValueAttributesExample(AmazonTimestreamWriteClient writeClient)
    {
      this.writeClient = writeClient;
    }

    public async Task WriteRecordsMultiMeasureValueSingleRecord()
    {
      Console.WriteLine("Writing records with multi value attributes");
```

```
DateTimeOffset now = DateTimeOffset.UtcNow;
string currentTimeString = (now.ToUnixTimeMilliseconds()).ToString();

List<Dimension> dimensions = new List<Dimension>{
  new Dimension { Name = "region", Value = "us-east-1" },
  new Dimension { Name = "az", Value = "az1" },
  new Dimension { Name = "hostname", Value = "host1" }
};

var commonAttributes = new Record
{
  Dimensions = dimensions,
  Time = currentTimeString
};

var cpuUtilization = new MeasureValue
{
  Name = "cpu_utilization",
  Value = "13.6",
  Type = "DOUBLE"
};

var memoryUtilization = new MeasureValue
{
  Name = "memory_utilization",
  Value = "40",
  Type = "DOUBLE"
};

var computationalRecord = new Record
{
  MeasureName = "cpu_memory",
  MeasureValues = new List<MeasureValue> {cpuUtilization, memoryUtilization},
  MeasureValueType = "MULTI"
};


List<Record> records = new List<Record>();
records.Add(computationalRecord);

try
{
  var writeRecordsRequest = new WriteRecordsRequest
```

```
        {
          DatabaseName = MultiMeasureValueConstants.MultiMeasureValueSampleDb,
          TableName = MultiMeasureValueConstants.MultiMeasureValueSampleTable,
          Records = records,
          CommonAttributes = commonAttributes
        };
        WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
        Console.WriteLine($"Write records status code:
{response.HttpStatusCode.ToString()}");
      }
      catch (Exception e)
      {
        Console.WriteLine("Write records failure:" + e.ToString());
      }
    }

    public async Task WriteRecordsMultiMeasureValueMultipleRecords()
    {
      Console.WriteLine("Writing records with multi value attributes mixture type");

      DateTimeOffset now = DateTimeOffset.UtcNow;
      string currentTimeString = (now.ToUnixTimeMilliseconds()).ToString();

      List<Dimension> dimensions = new List<Dimension>{
        new Dimension { Name = "region", Value = "us-east-1" },
        new Dimension { Name = "az", Value = "az1" },
        new Dimension { Name = "hostname", Value = "host1" }
      };

      var commonAttributes = new Record
      {
        Dimensions = dimensions,
        Time = currentTimeString
      };

      var cpuUtilization = new MeasureValue
      {
        Name = "cpu_utilization",
        Value = "13.6",
        Type = "DOUBLE"
      };

      var memoryUtilization = new MeasureValue
```

```
      {
        Name = "memory_utilization",
        Value = "40",
        Type = "DOUBLE"
      };

      var activeCores = new MeasureValue
      {
        Name = "active_cores",
        Value = "4",
        Type = "BIGINT"
      };

      var computationalRecord = new Record
      {
        MeasureName = "computational_utilization",
        MeasureValues = new List<MeasureValue> {cpuUtilization, memoryUtilization,
activeCores},
        MeasureValueType = "MULTI"
      };

      var aliveRecord = new Record
      {
        MeasureName = "is_healthy",
        MeasureValue = "true",
        MeasureValueType = "BOOLEAN"
      };

      List<Record> records = new List<Record>();
      records.Add(computationalRecord);
      records.Add(aliveRecord);

      try
      {
        var writeRecordsRequest = new WriteRecordsRequest
        {
          DatabaseName = MultiMeasureValueConstants.MultiMeasureValueSampleDb,
          TableName = MultiMeasureValueConstants.MultiMeasureValueSampleTable,
          Records = records,
          CommonAttributes = commonAttributes
        };
        WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
```

```
        Console.WriteLine($"Write records status code:
 {response.HttpStatusCode.ToString()}");
      }
      catch (Exception e)
      {
        Console.WriteLine("Write records failure:" + e.ToString());
      }
    }
  }
}
```

書き込み失敗の処理

Amazon Timestream での書き込みは、次のいずれかの理由で失敗する可能性があります。

- タイムスタンプがメモリストアの保持期間外にあるレコードがあります。
- Timestream で定義された制限を超えるディメンションやメジャーを含むレコードがあります。
- Amazon Timestream が重複レコードを検出しました。同じディメンション、タイムスタンプ、メジャー名を持つ複数のレコードがある場合、レコードは重複としてマークされますが、次のようになります。
  - 測定値は異なります。
  - バージョンがリクエストに存在しないか、新しいレコードのバージョン値が既存の値以下です。Amazon Timestream がこの理由でデータを拒否`RejectedRecords`した場合、 の `ExistingVersion` フィールドには、Amazon Timestream に保存されているレコードの最新バージョンが含まれます。更新を強制するには、レコードセットのバージョンを より大きい値に設定したリクエストを再送信できます`ExistingVersion`。

エラーと拒否されたレコードの詳細については、「エラー」と「」を参照してくださいRejectedRecord。

Timestream にレコードを書き込もう`RejectedRecordsException`としたときにアプリケーションが を受信した場合、拒否されたレコードを解析して、次のように書き込み失敗の詳細を確認できます。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいていますGitHub。サンプルアプリケーションの使用を開始する方法の詳細については、「」を参照してくださいサンプルアプリケーション。

Java

```
 try {
   WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
   System.out.println("WriteRecords Status: " +
writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
 } catch (RejectedRecordsException e) {
   System.out.println("RejectedRecords: " + e);
   for (RejectedRecord rejectedRecord : e.getRejectedRecords()) {
     System.out.println("Rejected Index " + rejectedRecord.getRecordIndex() + ": "
         + rejectedRecord.getReason());
   }
   System.out.println("Other records were written successfully. ");
 } catch (Exception e) {
   System.out.println("Error: " + e);
 }
```

Java v2

```
   try {
     WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsRequest);
     System.out.println("writeRecordsWithCommonAttributes Status: " +
writeRecordsResponse.sdkHttpResponse().statusCode());
   } catch (RejectedRecordsException e) {
     System.out.println("RejectedRecords: " + e);
     for (RejectedRecord rejectedRecord : e.rejectedRecords()) {
       System.out.println("Rejected Index " + rejectedRecord.recordIndex() + ": "
           + rejectedRecord.reason());
     }
     System.out.println("Other records were written successfully. ");
   } catch (Exception e) {
     System.out.println("Error: " + e);
```

```
    }
```

Go

```
_, err = writeSvc.WriteRecords(writeRecordsInput)

if err != nil {
  fmt.Println("Error:")
  fmt.Println(err)
} else {
  fmt.Println("Write records is successful")
}
```

Python

```
try:
  result = self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME, Records=records, CommonAttributes=common_attributes)
  print("WriteRecords Status: [%s]" % result['ResponseMetadata']['HTTPStatusCode'])
except self.client.exceptions.RejectedRecordsException as err:
  print("RejectedRecords: ", err)
  for rr in err.response["RejectedRecords"]:
    print("Rejected Index " + str(rr["RecordIndex"]) + ": " + rr["Reason"])
  print("Other records were written successfully. ")
except Exception as err:
  print("Error:", err)
```

Node.js

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
await request.promise().then(
    (data) => {
      console.log("Write records successful");
    },
    (err) => {
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        const responsePayload =
 JSON.parse(request.response.httpResponse.body.toString());
```

```
        console.log("RejectedRecords: ", responsePayload.RejectedRecords);
        console.log("Other records were written successfully. ");
      }
    }
  );
```

.NET

```
 try
 {
   var writeRecordsRequest = new WriteRecordsRequest
   {
     DatabaseName = Constants.DATABASE_NAME,
     TableName = Constants.TABLE_NAME,
     Records = records,
     CommonAttributes = commonAttributes
   };
   WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
   Console.WriteLine($"Write records status code:
{response.HttpStatusCode.ToString()}");
 }
 catch (RejectedRecordsException e) {
   Console.WriteLine("RejectedRecordsException:" + e.ToString());
   foreach (RejectedRecord rr in e.RejectedRecords) {
     Console.WriteLine("RecordIndex " + rr.RecordIndex + " : " + rr.Reason);
   }
   Console.WriteLine("Other records were written successfully. ");
 }
 catch (Exception e)
 {
   Console.WriteLine("Write records failure:" + e.ToString());
 }
```

クエリの実行

トピック

- 結果のページ分割
- 結果セットの解析
- クエリステータスへのアクセス

結果のページ分割

クエリを実行すると、Timestream は結果セットをページ分割して返し、アプリケーションの応答性を最適化します。以下のコードスニペットは、結果セットをページ分割する方法を示しています。null 値が表示されるまで、すべての結果セットページをループする必要があります。ページ分割トークンは、 の Timestream によって発行されてから 3 時間後に期限切れになります LiveAnalytics。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいていますGitHub。サンプルアプリケーションの使用を開始する方法の詳細については、「」を参照してくださいサンプルアプリケーション。

Java

```
    private void runQuery(String queryString) {
        try {
            QueryRequest queryRequest = new QueryRequest();
            queryRequest.setQueryString(queryString);
            QueryResult queryResult = queryClient.query(queryRequest);
            while (true) {
                parseQueryResult(queryResult);
                if (queryResult.getNextToken() == null) {
                    break;
                }
                queryRequest.setNextToken(queryResult.getNextToken());
                queryResult = queryClient.query(queryRequest);
            }
        } catch (Exception e) {
            // Some queries might fail with 500 if the result of a sequence function
has more than 10000 entries
            e.printStackTrace();
        }
    }
```

Java v2

```
    private void runQuery(String queryString) {
        try {
```

```
            QueryRequest queryRequest =
 QueryRequest.builder().queryString(queryString).build();
            final QueryIterable queryResponseIterator =
 timestreamQueryClient.queryPaginator(queryRequest);
            for(QueryResponse queryResponse : queryResponseIterator) {
                parseQueryResult(queryResponse);
            }
        } catch (Exception e) {
            // Some queries might fail with 500 if the result of a sequence function
 has more than 10000 entries
            e.printStackTrace();
        }
    }
```

Go

```
func runQuery(queryPtr *string, querySvc *timestreamquery.TimestreamQuery, f
 *os.File) {
    queryInput := &timestreamquery.QueryInput{
        QueryString: aws.String(*queryPtr),
    }
    fmt.Println("QueryInput:")
    fmt.Println(queryInput)
    // execute the query
    err := querySvc.QueryPages(queryInput,
        func(page *timestreamquery.QueryOutput, lastPage bool) bool {
            // process query response
            queryStatus := page.QueryStatus
            fmt.Println("Current query status:", queryStatus)
            // query response metadata
            // includes column names and types
            metadata := page.ColumnInfo
            // fmt.Println("Metadata:")
            fmt.Println(metadata)
            header := ""
            for i := 0; i < len(metadata); i++ {
                header += *metadata[i].Name
                if i != len(metadata)-1 {
                    header += ", "
                }
            }
            write(f, header)
```

```
            // query response data
            fmt.Println("Data:")
            // process rows
            rows := page.Rows
            for i := 0; i < len(rows); i++ {
                data := rows[i].Data
                value := processRowType(data, metadata)
                fmt.Println(value)
                write(f, value)
            }
            fmt.Println("Number of rows:", len(page.Rows))
            return true
        })
    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    }
}
```

Python

```
    def run_query(self, query_string):
        try:
            page_iterator = self.paginator.paginate(QueryString=query_string)
            for page in page_iterator:
                self._parse_query_result(page)
        except Exception as err:
            print("Exception while running query:", err)
```

Node.js

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
async function getAllRows(query, nextToken) {
    const params = {
        QueryString: query
    };

    if (nextToken) {
        params.NextToken = nextToken;
```

```
    }

    await queryClient.query(params).promise()
        .then(
            (response) => {
                parseQueryResult(response);
                if (response.NextToken) {
                    getAllRows(query, response.NextToken);
                }
            },
            (err) => {
                console.error("Error while querying:", err);
            });
}
```

.NET

```
        private async Task RunQueryAsync(string queryString)
        {
            try
            {
                QueryRequest queryRequest = new QueryRequest();
                queryRequest.QueryString = queryString;
                QueryResponse queryResponse = await
 queryClient.QueryAsync(queryRequest);
                while (true)
                {
                    ParseQueryResult(queryResponse);
                    if (queryResponse.NextToken == null)
                    {
                        break;
                    }
                    queryRequest.NextToken = queryResponse.NextToken;
                    queryResponse = await queryClient.QueryAsync(queryRequest);
                }
            } catch(Exception e)
            {
                // Some queries might fail with 500 if the result of a sequence
 function has more than 10000 entries
                Console.WriteLine(e.ToString());
            }
        }
```

結果セットの解析

次のコードスニペットを使用して、結果セットからデータを抽出できます。クエリ結果は、クエリ完了後最大 24 時間アクセスできます。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいていますGitHub。サンプルアプリケーションの使用を開始する方法の詳細については、「」を参照してくださいサンプルアプリケーション。

Java

```
   private static final DateTimeFormatter TIMESTAMP_FORMATTER =
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss.SSSSSSSSS");
   private static final DateTimeFormatter DATE_FORMATTER =
DateTimeFormatter.ofPattern("yyyy-MM-dd");
   private static final DateTimeFormatter TIME_FORMATTER =
DateTimeFormatter.ofPattern("HH:mm:ss.SSSSSSSSS");

   private static final long ONE_GB_IN_BYTES = 1073741824L;

   private void parseQueryResult(QueryResult response) {
       final QueryStatus currentStatusOfQuery = queryResult.getQueryStatus();

       System.out.println("Query progress so far: " +
currentStatusOfQuery.getProgressPercentage() + "%");

       double bytesScannedSoFar = ((double)
currentStatusOfQuery.getCumulativeBytesScanned() / ONE_GB_IN_BYTES);
       System.out.println("Bytes scanned so far: " + bytesScannedSoFar + " GB");

       double bytesMeteredSoFar = ((double)
currentStatusOfQuery.getCumulativeBytesMetered() / ONE_GB_IN_BYTES);
       System.out.println("Bytes metered so far: " + bytesMeteredSoFar + " GB");

       List<ColumnInfo> columnInfo = response.getColumnInfo();
       List<Row> rows = response.getRows();

       System.out.println("Metadata: " + columnInfo);
       System.out.println("Data: ");
```

```
        // iterate every row
        for (Row row : rows) {
            System.out.println(parseRow(columnInfo, row));
        }
    }

    private String parseRow(List<ColumnInfo> columnInfo, Row row) {
        List<Datum> data = row.getData();
        List<String> rowOutput = new ArrayList<>();
        // iterate every column per row
        for (int j = 0; j < data.size(); j++) {
            ColumnInfo info = columnInfo.get(j);
            Datum datum = data.get(j);
            rowOutput.add(parseDatum(info, datum));
        }
        return String.format("{%s}",
rowOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }

    private String parseDatum(ColumnInfo info, Datum datum) {
        if (datum.isNullValue() != null && datum.isNullValue()) {
            return info.getName() + "=" + "NULL";
        }
        Type columnType = info.getType();
        // If the column is of TimeSeries Type
        if (columnType.getTimeSeriesMeasureValueColumnInfo() != null) {
            return parseTimeSeries(info, datum);
        }
        // If the column is of Array Type
        else if (columnType.getArrayColumnInfo() != null) {
            List<Datum> arrayValues = datum.getArrayValue();
            return info.getName() + "=" +
parseArray(info.getType().getArrayColumnInfo(), arrayValues);
        }
        // If the column is of Row Type
        else if (columnType.getRowColumnInfo() != null) {
            List<ColumnInfo> rowColumnInfo = info.getType().getRowColumnInfo();
            Row rowValues = datum.getRowValue();
            return parseRow(rowColumnInfo, rowValues);
        }
        // If the column is of Scalar Type
        else {
            return parseScalarType(info, datum);
```

```
        }
    }

    private String parseTimeSeries(ColumnInfo info, Datum datum) {
        List<String> timeSeriesOutput = new ArrayList<>();
        for (TimeSeriesDataPoint dataPoint : datum.getTimeSeriesValue()) {
            timeSeriesOutput.add("{time=" + dataPoint.getTime() + ", value=" +
                    parseDatum(info.getType().getTimeSeriesMeasureValueColumnInfo(),
dataPoint.getValue()) + "}");
        }
        return String.format("[%s]",
timeSeriesOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }

    private String parseScalarType(ColumnInfo info, Datum datum) {
        switch (ScalarType.fromValue(info.getType().getScalarType())) {
            case VARCHAR:
                return parseColumnName(info) + datum.getScalarValue();
            case BIGINT:
                Long longValue = Long.valueOf(datum.getScalarValue());
                return parseColumnName(info) + longValue;
            case INTEGER:
                Integer intValue = Integer.valueOf(datum.getScalarValue());
                return parseColumnName(info) + intValue;
            case BOOLEAN:
                Boolean booleanValue = Boolean.valueOf(datum.getScalarValue());
                return parseColumnName(info) + booleanValue;
            case DOUBLE:
                Double doubleValue = Double.valueOf(datum.getScalarValue());
                return parseColumnName(info) + doubleValue;
            case TIMESTAMP:
                return parseColumnName(info) +
LocalDateTime.parse(datum.getScalarValue(), TIMESTAMP_FORMATTER);
            case DATE:
                return parseColumnName(info) +
LocalDate.parse(datum.getScalarValue(), DATE_FORMATTER);
            case TIME:
                return parseColumnName(info) +
LocalTime.parse(datum.getScalarValue(), TIME_FORMATTER);
            case INTERVAL_DAY_TO_SECOND:
            case INTERVAL_YEAR_TO_MONTH:
                return parseColumnName(info) + datum.getScalarValue();
            case UNKNOWN:
                return parseColumnName(info) + datum.getScalarValue();
```

```
            default:
                throw new IllegalArgumentException("Given type is not valid: " +
info.getType().getScalarType());
        }
    }

    private String parseColumnName(ColumnInfo info) {
        return info.getName() == null ? "" : info.getName() + "=";
    }

    private String parseArray(ColumnInfo arrayColumnInfo, List<Datum> arrayValues) {
        List<String> arrayOutput = new ArrayList<>();
        for (Datum datum : arrayValues) {
            arrayOutput.add(parseDatum(arrayColumnInfo, datum));
        }
        return String.format("[%s]",
arrayOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }
```

Java v2

```
    private static final long ONE_GB_IN_BYTES = 1073741824L;

    private void parseQueryResult(QueryResponse response) {
        final QueryStatus currentStatusOfQuery = response.queryStatus();

        System.out.println("Query progress so far: " +
currentStatusOfQuery.progressPercentage() + "%");

        double bytesScannedSoFar = ((double)
currentStatusOfQuery.cumulativeBytesScanned() / ONE_GB_IN_BYTES);
        System.out.println("Bytes scanned so far: " + bytesScannedSoFar + " GB");

        double bytesMeteredSoFar = ((double)
currentStatusOfQuery.cumulativeBytesMetered() / ONE_GB_IN_BYTES);
        System.out.println("Bytes metered so far: " + bytesMeteredSoFar + " GB");

        List<ColumnInfo> columnInfo = response.columnInfo();
        List<Row> rows = response.rows();

        System.out.println("Metadata: " + columnInfo);
        System.out.println("Data: ");
```

```
        // iterate every row
        for (Row row : rows) {
            System.out.println(parseRow(columnInfo, row));
        }
    }

    private String parseRow(List<ColumnInfo> columnInfo, Row row) {
        List<Datum> data = row.data();
        List<String> rowOutput = new ArrayList<>();
        // iterate every column per row
        for (int j = 0; j < data.size(); j++) {
            ColumnInfo info = columnInfo.get(j);
            Datum datum = data.get(j);
            rowOutput.add(parseDatum(info, datum));
        }
        return String.format("{%s}",
rowOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }

    private String parseDatum(ColumnInfo info, Datum datum) {
        if (datum.nullValue() != null && datum.nullValue()) {
            return info.name() + "=" + "NULL";
        }
        Type columnType = info.type();
        // If the column is of TimeSeries Type
        if (columnType.timeSeriesMeasureValueColumnInfo() != null) {
            return parseTimeSeries(info, datum);
        }
        // If the column is of Array Type
        else if (columnType.arrayColumnInfo() != null) {
            List<Datum> arrayValues = datum.arrayValue();
            return info.name() + "=" + parseArray(info.type().arrayColumnInfo(),
arrayValues);
        }
        // If the column is of Row Type
        else if (columnType.rowColumnInfo() != null &&
columnType.rowColumnInfo().size() > 0) {
            List<ColumnInfo> rowColumnInfo = info.type().rowColumnInfo();
            Row rowValues = datum.rowValue();
            return parseRow(rowColumnInfo, rowValues);
        }
        // If the column is of Scalar Type
        else {
            return parseScalarType(info, datum);
```

```
        }
    }

    private String parseTimeSeries(ColumnInfo info, Datum datum) {
        List<String> timeSeriesOutput = new ArrayList<>();
        for (TimeSeriesDataPoint dataPoint : datum.timeSeriesValue()) {
            timeSeriesOutput.add("{time=" + dataPoint.time() + ", value=" +
                    parseDatum(info.type().timeSeriesMeasureValueColumnInfo(),
 dataPoint.value()) + "}");
        }
        return String.format("[%s]",
 timeSeriesOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }

    private String parseScalarType(ColumnInfo info, Datum datum) {
        return parseColumnName(info) + datum.scalarValue();
    }

    private String parseColumnName(ColumnInfo info) {
        return info.name() == null ? "" : info.name() + "=";
    }

    private String parseArray(ColumnInfo arrayColumnInfo, List<Datum> arrayValues) {
        List<String> arrayOutput = new ArrayList<>();
        for (Datum datum : arrayValues) {
            arrayOutput.add(parseDatum(arrayColumnInfo, datum));
        }
        return String.format("[%s]",
 arrayOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }
```

Go

```
func processScalarType(data *timestreamquery.Datum) string {
    return *data.ScalarValue
}

func processTimeSeriesType(data []*timestreamquery.TimeSeriesDataPoint, columnInfo
 *timestreamquery.ColumnInfo) string {
    value := ""
    for k := 0; k < len(data); k++ {
        time := data[k].Time
        value += *time + ":"
```

```
        if columnInfo.Type.ScalarType != nil {
            value += processScalarType(data[k].Value)
        } else if columnInfo.Type.ArrayColumnInfo != nil {
            value += processArrayType(data[k].Value.ArrayValue,
 columnInfo.Type.ArrayColumnInfo)
        } else if columnInfo.Type.RowColumnInfo != nil {
            value += processRowType(data[k].Value.RowValue.Data,
 columnInfo.Type.RowColumnInfo)
        } else {
            fail("Bad data type")
        }
        if k != len(data)-1 {
            value += ", "
        }
    }
    return value
}

func processArrayType(datumList []*timestreamquery.Datum, columnInfo
 *timestreamquery.ColumnInfo) string {
    value := ""
    for k := 0; k < len(datumList); k++ {
        if columnInfo.Type.ScalarType != nil {
            value += processScalarType(datumList[k])
        } else if columnInfo.Type.TimeSeriesMeasureValueColumnInfo != nil {
            value += processTimeSeriesType(datumList[k].TimeSeriesValue,
 columnInfo.Type.TimeSeriesMeasureValueColumnInfo)
        } else if columnInfo.Type.ArrayColumnInfo != nil {
            value += "["
            value += processArrayType(datumList[k].ArrayValue,
 columnInfo.Type.ArrayColumnInfo)
            value += "]"
        } else if columnInfo.Type.RowColumnInfo != nil {
            value += "["
            value += processRowType(datumList[k].RowValue.Data,
 columnInfo.Type.RowColumnInfo)
            value += "]"
        } else {
            fail("Bad column type")
        }

        if k != len(datumList)-1 {
            value += ", "
        }
```

```
    }
    return value
}

func processRowType(data []*timestreamquery.Datum, metadata
 []*timestreamquery.ColumnInfo) string {
    value := ""
    for j := 0; j < len(data); j++ {
        if metadata[j].Type.ScalarType != nil {
            // process simple data types
            value += processScalarType(data[j])
        } else if metadata[j].Type.TimeSeriesMeasureValueColumnInfo != nil {
            // fmt.Println("Timeseries measure value column info")
            // fmt.Println(metadata[j].Type.TimeSeriesMeasureValueColumnInfo.Type)
            datapointList := data[j].TimeSeriesValue
            value += "["
            value += processTimeSeriesType(datapointList,
 metadata[j].Type.TimeSeriesMeasureValueColumnInfo)
            value += "]"
        } else if metadata[j].Type.ArrayColumnInfo != nil {
            columnInfo := metadata[j].Type.ArrayColumnInfo
            // fmt.Println("Array column info")
            // fmt.Println(columnInfo)
            datumList := data[j].ArrayValue
            value += "["
            value += processArrayType(datumList, columnInfo)
            value += "]"
        } else if metadata[j].Type.RowColumnInfo != nil {
            columnInfo := metadata[j].Type.RowColumnInfo
            datumList := data[j].RowValue.Data
            value += "["
            value += processRowType(datumList, columnInfo)
            value += "]"
        } else {
            panic("Bad column type")
        }
        // comma seperated column values
        if j != len(data)-1 {
            value += ", "
        }
    }
    return value
}
```

Python

```
   def _parse_query_result(self, query_result):
       query_status = query_result["QueryStatus"]

       progress_percentage = query_status["ProgressPercentage"]
       print(f"Query progress so far: {progress_percentage}%")

       bytes_scanned = float(query_status["CumulativeBytesScanned"]) /
ONE_GB_IN_BYTES
       print(f"Data scanned so far: {bytes_scanned} GB")

       bytes_metered = float(query_status["CumulativeBytesMetered"]) /
ONE_GB_IN_BYTES
       print(f"Data metered so far: {bytes_metered} GB")

       column_info = query_result['ColumnInfo']

       print("Metadata: %s" % column_info)
       print("Data: ")
       for row in query_result['Rows']:
           print(self._parse_row(column_info, row))

   def _parse_row(self, column_info, row):
       data = row['Data']
       row_output = []
       for j in range(len(data)):
           info = column_info[j]
           datum = data[j]
           row_output.append(self._parse_datum(info, datum))

       return "{%s}" % str(row_output)

   def _parse_datum(self, info, datum):
       if datum.get('NullValue', False):
           return "%s=NULL" % info['Name'],

       column_type = info['Type']

       # If the column is of TimeSeries Type
       if 'TimeSeriesMeasureValueColumnInfo' in column_type:
           return self._parse_time_series(info, datum)

       # If the column is of Array Type
```

```
        elif 'ArrayColumnInfo' in column_type:
            array_values = datum['ArrayValue']
            return "%s=%s" % (info['Name'], self._parse_array(info['Type']
['ArrayColumnInfo'], array_values))

        # If the column is of Row Type
        elif 'RowColumnInfo' in column_type:
            row_column_info = info['Type']['RowColumnInfo']
            row_values = datum['RowValue']
            return self._parse_row(row_column_info, row_values)

        # If the column is of Scalar Type
        else:
            return self._parse_column_name(info) + datum['ScalarValue']

    def _parse_time_series(self, info, datum):
        time_series_output = []
        for data_point in datum['TimeSeriesValue']:
            time_series_output.append("{time=%s, value=%s}"
                                      % (data_point['Time'],
                                         self._parse_datum(info['Type']
['TimeSeriesMeasureValueColumnInfo'],
                                                           data_point['Value'])))
        return "[%s]" % str(time_series_output)

    def _parse_array(self, array_column_info, array_values):
        array_output = []
        for datum in array_values:
            array_output.append(self._parse_datum(array_column_info, datum))

        return "[%s]" % str(array_output)

    @staticmethod
    def _parse_column_name(info):
        if 'Name' in info:
            return info['Name'] + "="
        else:
            return ""
```

Node.js

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
function parseQueryResult(response) {
    const queryStatus = response.QueryStatus;
    console.log("Current query status: " + JSON.stringify(queryStatus));

    const columnInfo = response.ColumnInfo;
    const rows = response.Rows;

    console.log("Metadata: " + JSON.stringify(columnInfo));
    console.log("Data: ");

    rows.forEach(function (row) {
        console.log(parseRow(columnInfo, row));
    });
}

function parseRow(columnInfo, row) {
    const data = row.Data;
    const rowOutput = [];

    var i;
    for ( i = 0; i < data.length; i++ ) {
        info = columnInfo[i];
        datum = data[i];
        rowOutput.push(parseDatum(info, datum));
    }

    return `{${rowOutput.join(", ")}}`
}

function parseDatum(info, datum) {
    if (datum.NullValue != null && datum.NullValue === true) {
        return `${info.Name}=NULL`;
    }

    const columnType = info.Type;

    // If the column is of TimeSeries Type
```

```
    if (columnType.TimeSeriesMeasureValueColumnInfo != null) {
        return parseTimeSeries(info, datum);
    }
    // If the column is of Array Type
    else if (columnType.ArrayColumnInfo != null) {
        const arrayValues = datum.ArrayValue;
        return `${info.Name}=${parseArray(info.Type.ArrayColumnInfo, arrayValues)}`;
    }
    // If the column is of Row Type
    else if (columnType.RowColumnInfo != null) {
        const rowColumnInfo = info.Type.RowColumnInfo;
        const rowValues = datum.RowValue;
        return parseRow(rowColumnInfo, rowValues);
    }
    // If the column is of Scalar Type
    else {
        return parseScalarType(info, datum);
    }
}

function parseTimeSeries(info, datum) {
    const timeSeriesOutput = [];
    datum.TimeSeriesValue.forEach(function (dataPoint) {
        timeSeriesOutput.push(`{time=${dataPoint.Time}, value=
${parseDatum(info.Type.TimeSeriesMeasureValueColumnInfo, dataPoint.Value)}}`)
    });

    return `[${timeSeriesOutput.join(", ")}]`
}

function parseScalarType(info, datum) {
    return parseColumnName(info) + datum.ScalarValue;
}

function parseColumnName(info) {
    return info.Name == null ? "" : `${info.Name}=`;
}

function parseArray(arrayColumnInfo, arrayValues) {
    const arrayOutput = [];
    arrayValues.forEach(function (datum) {
        arrayOutput.push(parseDatum(arrayColumnInfo, datum));
    });
    return `[${arrayOutput.join(", ")}]`
```

```
}
```

.NET

```
        private void ParseQueryResult(QueryResponse response)
        {
            List<ColumnInfo> columnInfo = response.ColumnInfo;
            var options = new JsonSerializerOptions
            {
                IgnoreNullValues = true
            };
            List<String> columnInfoStrings = columnInfo.ConvertAll(x =>
JsonSerializer.Serialize(x, options));
            List<Row> rows = response.Rows;

            QueryStatus queryStatus = response.QueryStatus;
            Console.WriteLine("Current Query status:" +
JsonSerializer.Serialize(queryStatus, options));

            Console.WriteLine("Metadata:" + string.Join(",", columnInfoStrings));
            Console.WriteLine("Data:");

            foreach (Row row in rows)
            {
                Console.WriteLine(ParseRow(columnInfo, row));
            }
        }

        private string ParseRow(List<ColumnInfo> columnInfo, Row row)
        {
            List<Datum> data = row.Data;
            List<string> rowOutput = new List<string>();
            for (int j = 0; j < data.Count; j++)
            {
                ColumnInfo info = columnInfo[j];
                Datum datum = data[j];
                rowOutput.Add(ParseDatum(info, datum));
            }
            return $"{{{string.Join(",", rowOutput)}}}";
        }

        private string ParseDatum(ColumnInfo info, Datum datum)
        {
```

```
            if (datum.NullValue)
            {
                return $"{info.Name}=NULL";
            }

            Amazon.TimestreamQuery.Model.Type columnType = info.Type;
            if (columnType.TimeSeriesMeasureValueColumnInfo != null)
            {
                return ParseTimeSeries(info, datum);
            }
            else if (columnType.ArrayColumnInfo != null)
            {
                List<Datum> arrayValues = datum.ArrayValue;
                return $"{info.Name}={ParseArray(info.Type.ArrayColumnInfo,
arrayValues)}";
            }
            else if (columnType.RowColumnInfo != null &&
columnType.RowColumnInfo.Count > 0)
            {
                List<ColumnInfo> rowColumnInfo = info.Type.RowColumnInfo;
                Row rowValue = datum.RowValue;
                return ParseRow(rowColumnInfo, rowValue);
            }
            else
            {
                return ParseScalarType(info, datum);
            }
        }

        private string ParseTimeSeries(ColumnInfo info, Datum datum)
        {
            var timeseriesString = datum.TimeSeriesValue
                .Select(value => $"{{time={value.Time},
value={ParseDatum(info.Type.TimeSeriesMeasureValueColumnInfo, value.Value)}}}")
                .Aggregate((current, next) => current + "," + next);

            return $"[{timeseriesString}]";
        }

        private string ParseScalarType(ColumnInfo info, Datum datum)
        {
            return ParseColumnName(info) + datum.ScalarValue;
        }
```

```
        private string ParseColumnName(ColumnInfo info)
        {
            return info.Name == null ? "" : (info.Name + "=");
        }

        private string ParseArray(ColumnInfo arrayColumnInfo, List<Datum>
arrayValues)
        {
            return $"[{arrayValues.Select(value => ParseDatum(arrayColumnInfo,
value)).Aggregate((current, next) => current + "," + next)}]";
        }
```

クエリステータスへのアクセス

クエリのステータスにはQueryResponse、クエリの進行状況、クエリによってスキャンされたバイト数、クエリによって計測されたバイト数に関する情報を含む からアクセスできます。bytesMetered と のbytesScanned値は累積的であり、ページングクエリ結果中に継続的に更新されます。この情報を使用して、個々のクエリによってスキャンされたバイト数を理解し、特定の決定を行うこともできます。例えば、クエリの料金がスキャンされた GB あたり 0.01 USD であるとすると、クエリあたり 25 USD を超えるクエリ、または GB X をキャンセルできます。以下のコードスニペットは、これを行う方法を示しています。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいていますGitHub。サンプルアプリケーションの使用を開始する方法の詳細については、「」を参照してくださいサンプルアプリケーション。

Java

```
   private static final long ONE_GB_IN_BYTES = 1073741824L;
   private static final double QUERY_COST_PER_GB_IN_DOLLARS = 0.01; // Assuming the
price of query is $0.01 per GB

   public void cancelQueryBasedOnQueryStatus() {
       System.out.println("Starting query: " + SELECT_ALL_QUERY);
       QueryRequest queryRequest = new QueryRequest();
       queryRequest.setQueryString(SELECT_ALL_QUERY);
       QueryResult queryResult = queryClient.query(queryRequest);
```

```
        while (true) {
            final QueryStatus currentStatusOfQuery = queryResult.getQueryStatus();
            System.out.println("Query progress so far: " +
currentStatusOfQuery.getProgressPercentage() + "%");
            double bytesMeteredSoFar = ((double)
currentStatusOfQuery.getCumulativeBytesMetered() / ONE_GB_IN_BYTES);
            System.out.println("Bytes metered so far: " + bytesMeteredSoFar + "
GB");
            // Cancel query if its costing more than 1 cent
            if (bytesMeteredSoFar * QUERY_COST_PER_GB_IN_DOLLARS > 0.01) {
                cancelQuery(queryResult);
                break;
            }

            if (queryResult.getNextToken() == null) {
                break;
            }
            queryRequest.setNextToken(queryResult.getNextToken());
            queryResult = queryClient.query(queryRequest);
        }
    }
```

Java v2

```
    private static final long ONE_GB_IN_BYTES = 1073741824L;
    private static final double QUERY_COST_PER_GB_IN_DOLLARS = 0.01; // Assuming the
price of query is $0.01 per GB

    public void cancelQueryBasedOnQueryStatus() {
        System.out.println("Starting query: " + SELECT_ALL_QUERY);
        QueryRequest queryRequest =
QueryRequest.builder().queryString(SELECT_ALL_QUERY).build();

        final QueryIterable queryResponseIterator =
timestreamQueryClient.queryPaginator(queryRequest);
        for(QueryResponse queryResponse : queryResponseIterator) {
            final QueryStatus currentStatusOfQuery = queryResponse.queryStatus();
            System.out.println("Query progress so far: " +
currentStatusOfQuery.progressPercentage() + "%");
            double bytesMeteredSoFar = ((double)
currentStatusOfQuery.cumulativeBytesMetered() / ONE_GB_IN_BYTES);
            System.out.println("Bytes metered so far: " + bytesMeteredSoFar + "GB");
```

```
            // Cancel query if its costing more than 1 cent
            if (bytesMeteredSoFar * QUERY_COST_PER_GB_IN_DOLLARS > 0.01) {
                cancelQuery(queryResponse);
                break;
            }
        }
    }
```

Go

```
const OneGbInBytes = 1073741824
// Assuming the price of query is $0.01 per GB
const QueryCostPerGbInDollars = 0.01

func cancelQueryBasedOnQueryStatus(queryPtr *string, querySvc
 *timestreamquery.TimestreamQuery, f *os.File) {
    queryInput := &timestreamquery.QueryInput{
        QueryString: aws.String(*queryPtr),
    }
    fmt.Println("QueryInput:")
    fmt.Println(queryInput)
    // execute the query
    err := querySvc.QueryPages(queryInput,
        func(page *timestreamquery.QueryOutput, lastPage bool) bool {
            // process query response
            queryStatus := page.QueryStatus
            fmt.Println("Current query status:", queryStatus)
            bytes_metered := float64(*queryStatus.CumulativeBytesMetered) /
 float64(ONE_GB_IN_BYTES)
            if bytes_metered * QUERY_COST_PER_GB_IN_DOLLARS > 0.01 {
                cancelQuery(page, querySvc)
                return true
            }
            // query response metadata
            // includes column names and types
            metadata := page.ColumnInfo
            // fmt.Println("Metadata:")
            fmt.Println(metadata)
            header := ""
            for i := 0; i < len(metadata); i++ {
                header += *metadata[i].Name
                if i != len(metadata)-1 {
                    header += ", "
```

```
                }
            }
            write(f, header)

            // query response data
            fmt.Println("Data:")
            // process rows
            rows := page.Rows
            for i := 0; i < len(rows); i++ {
                data := rows[i].Data
                value := processRowType(data, metadata)
                fmt.Println(value)
                write(f, value)
            }
            fmt.Println("Number of rows:", len(page.Rows))
            return true
        })
    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    }
}
```

Python

```
ONE_GB_IN_BYTES = 1073741824
# Assuming the price of query is $0.01 per GB
QUERY_COST_PER_GB_IN_DOLLARS = 0.01

    def cancel_query_based_on_query_status(self):
        try:
            print("Starting query: " + self.SELECT_ALL)
            page_iterator = self.paginator.paginate(QueryString=self.SELECT_ALL)
            for page in page_iterator:
                query_status = page["QueryStatus"]
                progress_percentage = query_status["ProgressPercentage"]
                print("Query progress so far: " + str(progress_percentage) + "%")
                bytes_metered = query_status["CumulativeBytesMetered"] /
 self.ONE_GB_IN_BYTES
                print("Bytes Metered so far: " + str(bytes_metered) + " GB")
                if bytes_metered * self.QUERY_COST_PER_GB_IN_DOLLARS > 0.01:
                    self.cancel_query_for(page)
                    break
```

```
        except Exception as err:
            print("Exception while running query:", err)
            traceback.print_exc(file=sys.stderr)
```

Node.js

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
function parseQueryResult(response) {
    const queryStatus = response.QueryStatus;
    console.log("Current query status: " + JSON.stringify(queryStatus));

    const columnInfo = response.ColumnInfo;
    const rows = response.Rows;

    console.log("Metadata: " + JSON.stringify(columnInfo));
    console.log("Data: ");

    rows.forEach(function (row) {
        console.log(parseRow(columnInfo, row));
    });
}

function parseRow(columnInfo, row) {
    const data = row.Data;
    const rowOutput = [];

    var i;
    for ( i = 0; i < data.length; i++ ) {
        info = columnInfo[i];
        datum = data[i];
        rowOutput.push(parseDatum(info, datum));
    }

    return `{${rowOutput.join(", ")}}`
}

function parseDatum(info, datum) {
    if (datum.NullValue != null && datum.NullValue === true) {
        return `${info.Name}=NULL`;
    }
```

```
    const columnType = info.Type;

    // If the column is of TimeSeries Type
    if (columnType.TimeSeriesMeasureValueColumnInfo != null) {
        return parseTimeSeries(info, datum);
    }
    // If the column is of Array Type
    else if (columnType.ArrayColumnInfo != null) {
        const arrayValues = datum.ArrayValue;
        return `${info.Name}=${parseArray(info.Type.ArrayColumnInfo, arrayValues)}`;
    }
    // If the column is of Row Type
    else if (columnType.RowColumnInfo != null) {
        const rowColumnInfo = info.Type.RowColumnInfo;
        const rowValues = datum.RowValue;
        return parseRow(rowColumnInfo, rowValues);
    }
    // If the column is of Scalar Type
    else {
        return parseScalarType(info, datum);
    }
}

function parseTimeSeries(info, datum) {
    const timeSeriesOutput = [];
    datum.TimeSeriesValue.forEach(function (dataPoint) {
        timeSeriesOutput.push(`{time=${dataPoint.Time}, value=
${parseDatum(info.Type.TimeSeriesMeasureValueColumnInfo, dataPoint.Value)}}`)
    });

    return `[${timeSeriesOutput.join(", ")}]`
}

function parseScalarType(info, datum) {
    return parseColumnName(info) + datum.ScalarValue;
}

function parseColumnName(info) {
    return info.Name == null ? "" : `${info.Name}=`;
}

function parseArray(arrayColumnInfo, arrayValues) {
    const arrayOutput = [];
```

```
    arrayValues.forEach(function (datum) {
        arrayOutput.push(parseDatum(arrayColumnInfo, datum));
    });
    return `[${arrayOutput.join(", ")}]`
}
```

.NET

```
private static readonly long ONE_GB_IN_BYTES = 1073741824L;
private static readonly double QUERY_COST_PER_GB_IN_DOLLARS = 0.01; // Assuming the
 price of query is $0.01 per GB

private async Task CancelQueryBasedOnQueryStatus(string queryString)
{
    try
    {
        QueryRequest queryRequest = new QueryRequest();
        queryRequest.QueryString = queryString;
        QueryResponse queryResponse = await queryClient.QueryAsync(queryRequest);
        while (true)
        {
            QueryStatus queryStatus = queryResponse.QueryStatus;
            double bytesMeteredSoFar = ((double)
 queryStatus.CumulativeBytesMetered / ONE_GB_IN_BYTES);
            // Cancel query if its costing more than 1 cent
            if (bytesMeteredSoFar * QUERY_COST_PER_GB_IN_DOLLARS > 0.01)
            {
                await CancelQuery(queryResponse);
                break;
            }

            ParseQueryResult(queryResponse);
            if (queryResponse.NextToken == null)
            {
                break;
            }
            queryRequest.NextToken = queryResponse.NextToken;
            queryResponse = await queryClient.QueryAsync(queryRequest);
       }
    } catch(Exception e)
    {
        // Some queries might fail with 500 if the result of a sequence function has
 more than 10000 entries
```

```
        Console.WriteLine(e.ToString());
    }
}
```

クエリをキャンセルする方法の詳細については、「」を参照してくださいクエリをキャンセルする。

UNLOAD クエリの実行

次のコード例では、UNLOADクエリを呼び出します。UNLOAD の詳細については、「UNLOAD を使用して、 の Timestream から S3 にクエリ結果をエクスポートする LiveAnalytics」を参照してください。UNLOAD クエリの例については、「」を参照してくださいの Timestream UNLOAD からの のユースケースの例 LiveAnalytics。

トピック

- UNLOAD クエリを構築して実行する
- UNLOAD レスポンスを解析し、行数、マニフェストリンク、メタデータリンクを取得する
- マニフェストコンテンツの読み取りと解析
- メタデータコンテンツの読み取りと解析

UNLOAD クエリを構築して実行する

Java

```
// When you have a SELECT like below

String QUERY_1 = "SELECT user_id, ip_address, event, session_id, measure_name, time,
 query, quantity, product_id, channel FROM "
        + DATABASE_NAME + "." + UNLOAD_TABLE_NAME
        + " WHERE time BETWEEN ago(2d) AND now()";

// You can construct UNLOAD query as follows
UnloadQuery unloadQuery = UnloadQuery.builder()
        .selectQuery(QUERY_1)
        .bucketName("timestream-sample-<region>-<accountId>")
        .resultsPrefix("without_partition")
        .format(CSV)
        .compression(UnloadQuery.Compression.GZIP)
        .build();
QueryResult unloadResult = runQuery(unloadQuery.getUnloadQuery());
```

```
// Run UNLOAD query (Similar to how you run SELECT query)
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.pagination
    private QueryResult runQuery(String queryString) {
        QueryResult queryResult = null;
        try {
            QueryRequest queryRequest = new QueryRequest();
            queryRequest.setQueryString(queryString);
            queryResult = queryClient.query(queryRequest);
            while (true) {
                parseQueryResult(queryResult);
                if (queryResult.getNextToken() == null) {
                    break;
                }
                queryRequest.setNextToken(queryResult.getNextToken());
                queryResult = queryClient.query(queryRequest);
            }
        } catch (Exception e) {
            // Some queries might fail with 500 if the result of a sequence function
 has more than 10000 entries
            e.printStackTrace();
        }
        return queryResult;
    }

// Utility that helps to construct UNLOAD query

@Builder
static class UnloadQuery {
    private String selectQuery;
    private String bucketName;
    private String resultsPrefix;
    private Format format;
    private Compression compression;
    private EncryptionType encryptionType;
    private List<String> partitionColumns;
    private String kmsKey;
    private Character csvFieldDelimiter;
    private Character csvEscapeCharacter;

    public String getUnloadQuery() {
        String destination = constructDestination();
        String withClause = constructOptionalParameters();
```

```
        return String.format("UNLOAD (%s) TO '%s' %s", selectQuery, destination,
withClause);
    }

    private String constructDestination() {
        return "s3://" + this.bucketName + "/" + this.resultsPrefix + "/";
    }

    private String constructOptionalParameters() {
        boolean isOptionalParametersPresent = Objects.nonNull(format)
                || Objects.nonNull(compression)
                || Objects.nonNull(encryptionType)
                || Objects.nonNull(partitionColumns)
                || Objects.nonNull(kmsKey)
                || Objects.nonNull(csvFieldDelimiter)
                || Objects.nonNull(csvEscapeCharacter);

        String withClause = "";
        if (isOptionalParametersPresent) {
            StringJoiner optionalParameters = new StringJoiner(",");
            if (Objects.nonNull(format)) {
                optionalParameters.add("format = '" + format + "'");
            }
            if (Objects.nonNull(compression)) {
                optionalParameters.add("compression = '" + compression + "'");
            }
            if (Objects.nonNull(encryptionType)) {
                optionalParameters.add("encryption = '" + encryptionType + "'");
            }
            if (Objects.nonNull(kmsKey)) {
                optionalParameters.add("kms_key = '" + kmsKey + "'");
            }
            if (Objects.nonNull(csvFieldDelimiter)) {
                optionalParameters.add("field_delimiter = '" + csvFieldDelimiter +
"'");
            }
            if (Objects.nonNull(csvEscapeCharacter)) {
                optionalParameters.add("escaped_by = '" + csvEscapeCharacter + "'");
            }
            if (Objects.nonNull(partitionColumns) && !partitionColumns.isEmpty()) {
                final StringJoiner partitionedByList = new StringJoiner(",");
                partitionColumns.forEach(column -> partitionedByList.add("'" +
column + "'"));
```

```
                optionalParameters.add(String.format("partitioned_by = ARRAY[%s]",
 partitionedByList));
            }
            withClause = String.format("WITH (%s)", optionalParameters);
        }
        return withClause;
    }

    public enum Format {
        CSV, PARQUET
    }

    public enum Compression {
        GZIP, NONE
    }

    public enum EncryptionType {
        SSE_S3, SSE_KMS
    }

    @Override
    public String toString() {
        return getUnloadQuery();
    }
}
```

Java v2

```
// When you have a SELECT like below

String QUERY_1 = "SELECT user_id, ip_address, event, session_id, measure_name, time,
 query, quantity, product_id, channel FROM "
        + DATABASE_NAME + "." + UNLOAD_TABLE_NAME
        + " WHERE time BETWEEN ago(2d) AND now()";

//You can construct UNLOAD query as follows
UnloadQuery unloadQuery = UnloadQuery.builder()
        .selectQuery(QUERY_1)
        .bucketName("timestream-sample-<region>-<accountId>")
        .resultsPrefix("without_partition")
        .format(CSV)
        .compression(UnloadQuery.Compression.GZIP)
        .build();
```

```
QueryResponse unloadResponse = runQuery(unloadQuery.getUnloadQuery());


// Run UNLOAD query (Similar to how you run SELECT query)
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.pagination
private QueryResponse runQuery(String queryString) {
   QueryResponse finalResponse = null;
    try {
        QueryRequest queryRequest =
 QueryRequest.builder().queryString(queryString).build();
        final QueryIterable queryResponseIterator =
 timestreamQueryClient.queryPaginator(queryRequest);
        for(QueryResponse queryResponse : queryResponseIterator) {
            parseQueryResult(queryResponse);
           finalResponse = queryResponse;
        }
    } catch (Exception e) {
        // Some queries might fail with 500 if the result of a sequence function has
 more than 10000 entries
        e.printStackTrace();
    }
   return finalResponse;
}

// Utility that helps to construct UNLOAD query
@Builder
static class UnloadQuery {
    private String selectQuery;
    private String bucketName;
    private String resultsPrefix;
    private Format format;
    private Compression compression;
    private EncryptionType encryptionType;
    private List<String> partitionColumns;
    private String kmsKey;
    private Character csvFieldDelimiter;
    private Character csvEscapeCharacter;

    public String getUnloadQuery() {
        String destination = constructDestination();
        String withClause = constructOptionalParameters();
```

```
        return String.format("UNLOAD (%s) TO '%s' %s", selectQuery, destination,
withClause);
    }

    private String constructDestination() {
        return "s3://" + this.bucketName + "/" + this.resultsPrefix + "/";
    }

    private String constructOptionalParameters() {
        boolean isOptionalParametersPresent = Objects.nonNull(format)
                || Objects.nonNull(compression)
                || Objects.nonNull(encryptionType)
                || Objects.nonNull(partitionColumns)
                || Objects.nonNull(kmsKey)
                || Objects.nonNull(csvFieldDelimiter)
                || Objects.nonNull(csvEscapeCharacter);

        String withClause = "";
        if (isOptionalParametersPresent) {
            StringJoiner optionalParameters = new StringJoiner(",");
            if (Objects.nonNull(format)) {
                optionalParameters.add("format = '" + format + "'");
            }
            if (Objects.nonNull(compression)) {
                optionalParameters.add("compression = '" + compression + "'");
            }
            if (Objects.nonNull(encryptionType)) {
                optionalParameters.add("encryption = '" + encryptionType + "'");
            }
            if (Objects.nonNull(kmsKey)) {
                optionalParameters.add("kms_key = '" + kmsKey + "'");
            }
            if (Objects.nonNull(csvFieldDelimiter)) {
                optionalParameters.add("field_delimiter = '" + csvFieldDelimiter +
"'");
            }
            if (Objects.nonNull(csvEscapeCharacter)) {
                optionalParameters.add("escaped_by = '" + csvEscapeCharacter + "'");
            }
            if (Objects.nonNull(partitionColumns) && !partitionColumns.isEmpty()) {
                final StringJoiner partitionedByList = new StringJoiner(",");
                partitionColumns.forEach(column -> partitionedByList.add("'" +
column + "'"));
```

```
                optionalParameters.add(String.format("partitioned_by = ARRAY[%s]",
 partitionedByList));
            }
            withClause = String.format("WITH (%s)", optionalParameters);
        }
        return withClause;
    }

    public enum Format {
        CSV, PARQUET
    }

    public enum Compression {
        GZIP, NONE
    }

    public enum EncryptionType {
        SSE_S3, SSE_KMS
    }

    @Override
    public String toString() {
        return getUnloadQuery();
    }
}
```

Go

```
// When you have a SELECT like below
var Query = "SELECT user_id, ip_address, event, session_id, measure_name, time,
 query, quantity, product_id, channel FROM "
+ *databaseName + "." + *tableName + " WHERE time BETWEEN ago(2d) AND now()"

// You can construct UNLOAD query as follows
var unloadQuery = UnloadQuery{
    Query: "SELECT user_id, ip_address, session_id, measure_name, time, query,
 quantity, product_id, channel, event FROM " + *databaseName + "." + *tableName +
    " WHERE time BETWEEN ago(2d) AND now()",
    Partitioned_by: []string{},
    Compression: "GZIP",
    Format: "CSV",
    S3Location: bucketName,
    ResultPrefix: "without_partition",
```

```
}


// Run UNLOAD query (Similar to how you run SELECT query)
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.pagination

queryInput := &timestreamquery.QueryInput{
    QueryString: build_query(unloadQuery),
}

err := querySvc.QueryPages(queryInput,
    func(page *timestreamquery.QueryOutput, lastPage bool) bool {
        if (lastPage) {
            var response = parseQueryResult(page)
            var unloadFiles = getManifestAndMetadataFiles(s3Svc, response)
            displayColumns(unloadFiles, unloadQuery.Partitioned_by)
            displayResults(s3Svc, unloadFiles)
        }
        return true
    })

if err != nil {
    fmt.Println("Error:")
    fmt.Println(err)
}

// Utility that helps to construct UNLOAD query
type UnloadQuery struct {
    Query string
    Partitioned_by []string
    Format string
    S3Location string
    ResultPrefix string
    Compression string
}

func build_query(unload_query UnloadQuery) *string {
    var query_results_s3_path = "'s3://" + unload_query.S3Location + "/" +
 unload_query.ResultPrefix + "/'"
    var query = "UNLOAD(" + unload_query.Query + ") TO " + query_results_s3_path + "
 WITH ( "
    if (len(unload_query.Partitioned_by) > 0) {
        query = query + "partitioned_by=ARRAY["
```

```
        for i, column := range unload_query.Partitioned_by {
            if i == 0 {
                query = query + "'" + column + "'"
            } else {
                query = query + ",'" + column + "'"
            }
        }
        query = query + "],"
    }
    query = query + " format='" + unload_query.Format + "', "
    query = query + "  compression='" + unload_query.Compression + "')"
    fmt.Println(query)
    return aws.String(query)
}
```

Python

```
# When you have a SELECT like below
QUERY_1 = "SELECT user_id, ip_address, event, session_id, measure_name, time, query,
 quantity, product_id, channel FROM "
        + database_name + "." + table_name + " WHERE time BETWEEN ago(2d) AND now()"
# You can construct UNLOAD query as follows
UNLOAD_QUERY_1 = UnloadQuery(QUERY_1, "timestream-sample-<region>-<accountId>",
 "without_partition", "CSV", "GZIP", "")

# Run UNLOAD query (Similar to how you run SELECT query)
# https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.pagination
def run_query(self, query_string):
    try:
        page_iterator = self.paginator.paginate(QueryString=UNLOAD_QUERY_1)
    except Exception as err:
        print("Exception while running query:", err)

# Utility that helps to construct UNLOAD query
class UnloadQuery:
    def __init__(self, query, s3_bucket_location, results_prefix, format,
 compression , partition_by):
        self.query = query
        self.s3_bucket_location = s3_bucket_location
        self.results_prefix = results_prefix
        self.format = format
        self.compression = compression
```

```
        self.partition_by = partition_by

    def build_query(self):
        query_results_s3_path = "'s3://" + self.s3_bucket_location + "/" +
 self.results_prefix + "/'"
        unload_query = "UNLOAD("
        unload_query = unload_query + self.query
        unload_query = unload_query + ") "
        unload_query = unload_query + " TO " + query_results_s3_path
        unload_query = unload_query + " WITH ( "

        if(len(self.partition_by) > 0) :
            unload_query = unload_query + " partitioned_by = ARRAY" +
 str(self.partition_by) + ","

        unload_query = unload_query + " format='" + self.format  + "', "
        unload_query = unload_query + "  compression='" + self.compression + "')"

        return unload_query
```

Node.js

```
// When you have a SELECT like below
QUERY_1 = "SELECT user_id, ip_address, event, session_id, measure_name, time, query,
 quantity, product_id, channel FROM "
        + database_name + "." + table_name + " WHERE time BETWEEN ago(2d) AND now()"
// You can construct UNLOAD query as follows
UNLOAD_QUERY_1 = new UnloadQuery(QUERY_1, "timestream-sample-<region>-<accountId>",
 "without_partition", "CSV", "GZIP", "")


// Run UNLOAD query (Similar to how you run SELECT query)
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.pagination

async runQuery(query = UNLOAD_QUERY_1, nextToken) {
    const params = new QueryCommand({
        QueryString: query
    });

    if (nextToken) {
        params.NextToken = nextToken;
    }
```

```
    await queryClient.send(params).then(
            (response) => {
                if (response.NextToken) {
                    runQuery(queryClient, query, response.NextToken);
                } else {
                    await parseAndDisplayResults(response);
                }
            },
            (err) => {
                console.error("Error while querying:", err);
            });
}


class UnloadQuery {
    constructor(query, s3_bucket_location, results_prefix, format, compression ,
 partition_by) {
        this.query = query;
        this.s3_bucket_location = s3_bucket_location
        this.results_prefix = results_prefix
        this.format = format
        this.compression = compression
        this.partition_by = partition_by
    }

    buildQuery() {
        const query_results_s3_path = "'s3://" + this.s3_bucket_location + "/" +
 this.results_prefix + "/'"
        let unload_query = "UNLOAD("
        unload_query = unload_query + this.query
        unload_query = unload_query + ") "
        unload_query = unload_query + " TO " + query_results_s3_path
        unload_query = unload_query + " WITH ( "

        if(this.partition_by.length > 0) {
            let partitionBy = ""
            this.partition_by.forEach((str, i) => {
                partitionBy = partitionBy + (i ? ",'" : "'") + str + "'"
            })
            unload_query = unload_query + " partitioned_by = ARRAY[" + partitionBy +
 "],"
        }
        unload_query = unload_query + " format='" + this.format  + "', "
```

```
        unload_query = unload_query + "  compression='" + this.compression + "')"

        return unload_query
    }
}
```

UNLOAD レスポンスを解析し、行数、マニフェストリンク、メタデータリンクを取得する

Java

```
// Parsing UNLOAD query response is similar to how you parse SELECT query response:
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-query.html#code-samples.run-query.parsing

// But unlike SELECT, UNLOAD only has 1 row * 3 columns outputed
// (rows, metadataFile, manifestFile) => (BIGINT, VARCHAR, VARCHAR)

public UnloadResponse parseResult(QueryResult queryResult) {
    Map<String, String> outputMap = new HashMap<>();
    for (int i = 0; i < queryResult.getColumnInfo().size(); i++) {
        outputMap.put(queryResult.getColumnInfo().get(i).getName(),
                queryResult.getRows().get(0).getData().get(i).getScalarValue());

    }
    return new UnloadResponse(outputMap);
}

@Getter
class UnloadResponse {
    private final String metadataFile;
    private final String manifestFile;
    private final int rows;

    public UnloadResponse(Map<String, String> unloadResponse) {
        this.metadataFile = unloadResponse.get("metadataFile");
        this.manifestFile = unloadResponse.get("manifestFile");
        this.rows = Integer.parseInt(unloadResponse.get("rows"));
    }
}
```

Java v2

```
// Parsing UNLOAD query response is similar to how you parse SELECT query response:
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.parsing

// But unlike SELECT, UNLOAD only has 1 row * 3 columns outputed
// (rows, metadataFile, manifestFile) => (BIGINT, VARCHAR, VARCHAR)

public UnloadResponse parseResult(QueryResponse queryResponse) {
    Map<String, String> outputMap = new HashMap<>();
    for (int i = 0; i < queryResponse.columnInfo().size(); i++) {
        outputMap.put(queryResponse.columnInfo().get(i).name(),
                queryResponse.rows().get(0).data().get(i).scalarValue());

    }
    return new UnloadResponse(outputMap);
}

@Getter
class UnloadResponse {
    private final String metadataFile;
    private final String manifestFile;
    private final int rows;

    public UnloadResponse(Map<String, String> unloadResponse) {
        this.metadataFile = unloadResponse.get("metadataFile");
        this.manifestFile = unloadResponse.get("manifestFile");
        this.rows = Integer.parseInt(unloadResponse.get("rows"));
    }
}
```

Go

```
// Parsing UNLOAD query response is similar to how you parse SELECT query response:
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.parsing

// But unlike SELECT, UNLOAD only has 1 row * 3 columns outputed
// (rows, metadataFile, manifestFile) => (BIGINT, VARCHAR, VARCHAR)

func parseQueryResult(queryOutput *timestreamquery.QueryOutput) map[string]string {
    var columnInfo = queryOutput.ColumnInfo;
```

```
    fmt.Println("ColumnInfo", columnInfo)
    fmt.Println("QueryId", queryOutput.QueryId)
    fmt.Println("QueryStatus", queryOutput.QueryStatus)
    return parseResponse(columnInfo, queryOutput.Rows[0])
}

func parseResponse(columnInfo []*timestreamquery.ColumnInfo, row
 *timestreamquery.Row) map[string]string {
    var datum = row.Data
    response := make(map[string]string)
    for i, column := range columnInfo {
        response[*column.Name] = *datum[i].ScalarValue
    }
    return response
}
```

Python

```
# Parsing UNLOAD query response is similar to how you parse SELECT query response:
# https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.parsing

# But unlike SELECT, UNLOAD only has 1 row * 3 columns outputed
# (rows, metadataFile, manifestFile) => (BIGINT, VARCHAR, VARCHAR)

for page in page_iterator:
  last_page = page
response = self._parse_unload_query_result(last_page)

def _parse_unload_query_result(self, query_result):
    column_info = query_result['ColumnInfo']

    print("ColumnInfo: %s" % column_info)
    print("QueryId: %s" % query_result['QueryId'])
    print("QueryStatus:%s" % query_result['QueryStatus'])
    return self.parse_unload_response(column_info, query_result['Rows'][0])

def parse_unload_response(self, column_info, row):
    response = {}
    data = row['Data']
    for i, column in enumerate(column_info):
        response[column['Name']] = data[i]['ScalarValue']
   print("Rows: %s" % response['rows'])
```

```
   print("Metadata File location: %s" % response['metadataFile'])
   print("Manifest File location: %s" % response['manifestFile'])
   return response
```

Node.js

```
# Parsing UNLOAD query response is similar to how you parse SELECT query response:
# https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.parsing

# But unlike SELECT, UNLOAD only has 1 row * 3 columns outputed
# (rows, metadataFile, manifestFile) => (BIGINT, VARCHAR, VARCHAR)

async parseAndDisplayResults(data, query) {
    const columnInfo = data['ColumnInfo'];
    console.log("ColumnInfo:", columnInfo)
    console.log("QueryId: %s", data['QueryId'])
    console.log("QueryStatus:", data['QueryStatus'])
    await this.parseResponse(columnInfo, data['Rows'][0], query)
}

async parseResponse(columnInfo, row, query) {
    let response = {}
    const data = row['Data']
    columnInfo.forEach((column, i) => {
        response[column['Name']] = data[i]['ScalarValue']
    })

   console.log("Manifest file", response['manifestFile']);
   console.log("Metadata file", response['metadataFile']);

   return response
}
```

マニフェストコンテンツの読み取りと解析

Java

```
// Read and parse manifest content
public UnloadManifest getUnloadManifest(UnloadResponse unloadResponse) throws
 IOException {
   AmazonS3URI s3URI = new AmazonS3URI(unloadResponse.getManifestFile());
```

```
    S3Object s3Object = s3Client.getObject(s3URI.getBucket(), s3URI.getKey());
    String manifestFileContent = new
 String(IOUtils.toByteArray(s3Object.getObjectContent()), StandardCharsets.UTF_8);
    return new Gson().fromJson(manifestFileContent, UnloadManifest.class);
}

class UnloadManifest {
    @Getter
    public class FileMetadata {
        long content_length_in_bytes;
        long row_count;
    }

    @Getter
    public class ResultFile {
        String url;
        FileMetadata file_metadata;
    }

    @Getter
    public class QueryMetadata {
        long total_content_length_in_bytes;
        long total_row_count;
        String result_format;
        String result_version;
    }

    @Getter
    public class Author {
        String name;
        String manifest_file_version;
    }

    @Getter
    private List<ResultFile> result_files;
    @Getter
    private QueryMetadata query_metadata;
    @Getter
    private Author author;
}
```

Java v2

```
// Read and parse manifest content
public UnloadManifest getUnloadManifest(UnloadResponse unloadResponse) throws
 URISyntaxException {
    // Space needs to encoded to use S3 parseUri function
    S3Uri s3Uri =
 s3Utilities.parseUri(URI.create(unloadResponse.getManifestFile().replace(" ",
 "%20")));
    ResponseBytes<GetObjectResponse> objectBytes =
 s3Client.getObjectAsBytes(GetObjectRequest.builder()
            .bucket(s3Uri.bucket().orElseThrow(() -> new
 URISyntaxException(unloadResponse.getManifestFile(), "Invalid S3 URI")))
            .key(s3Uri.key().orElseThrow(() -> new
 URISyntaxException(unloadResponse.getManifestFile(), "Invalid S3 URI")))
            .build());
    String manifestFileContent = new String(objectBytes.asByteArray(),
 StandardCharsets.UTF_8);
    return new Gson().fromJson(manifestFileContent, UnloadManifest.class);
}

class UnloadManifest {
    @Getter
    public class FileMetadata {
        long content_length_in_bytes;
        long row_count;
    }

    @Getter
    public class ResultFile {
        String url;
        FileMetadata file_metadata;
    }

    @Getter
    public class QueryMetadata {
        long total_content_length_in_bytes;
        long total_row_count;
        String result_format;
        String result_version;
    }

    @Getter
    public class Author {
```

```
        String name;
        String manifest_file_version;
    }

    @Getter
    private List<ResultFile> result_files;
    @Getter
    private QueryMetadata query_metadata;
    @Getter
    private Author author;
}
```

Go

```
// Read and parse manifest content

func getManifestFile(s3Svc *s3.S3, response map[string]string) Manifest {
    var manifestBuf = getObject(s3Svc, response["manifestFile"])
    var manifest Manifest
    json.Unmarshal(manifestBuf.Bytes(), &manifest)
    return manifest
}

func getObject(s3Svc *s3.S3, s3Uri string) *bytes.Buffer {
    u,_ := url.Parse(s3Uri)
    getObjectInput := &s3.GetObjectInput{
        Key:    aws.String(u.Path),
        Bucket: aws.String(u.Host),
    }
    getObjectOutput, err := s3Svc.GetObject(getObjectInput)
    if err != nil {
        fmt.Println("Error: %s\n", err.Error())
    }
    buf := new(bytes.Buffer)
    buf.ReadFrom(getObjectOutput.Body)
    return buf
}

// Unload's Manifest structure

type Manifest struct {
    Author interface{}
    Query_metadata map[string]any
```

```
    Result_files  []struct {
        File_metadata interface{}
        Url string
    }
}}
```

Python

```
def __get_manifest_file(self, response):
    manifest = self.get_object(response['manifestFile']).read().decode('utf-8')
    parsed_manifest = json.loads(manifest)
    print("Manifest contents: \n%s" % parsed_manifest)

def get_object(self, uri):
    try:
        bucket, key = uri.replace("s3://", "").split("/", 1)
        s3_client = boto3.client('s3', region_name=<region>)
        response = s3_client.get_object(Bucket=bucket, Key=key)
        return response['Body']
    except Exception as err:
        print("Failed to get the object for URI:", uri)
        raise err
```

Node.js

```
// Read and parse manifest content

async getManifestFile(response) {
    let manifest;
    await this.getS3Object(response['manifestFile']).then(
        (data) => {
            manifest = JSON.parse(data);
        }
    );
    return manifest;
}

async getS3Object(uri) {
    const {bucketName, key} = this.getBucketAndKey(uri);
    const params = new GetObjectCommand({
        Bucket: bucketName,
        Key: key
    })
```

```
    const response = await this.s3Client.send(params);
    return await response.Body.transformToString();
}

getBucketAndKey(uri) {
    const [bucketName] = uri.replace("s3://", "").split("/", 1);
    const key = uri.replace("s3://", "").split('/').slice(1).join('/');
    return {bucketName, key};
}
```

メタデータコンテンツの読み取りと解析

Java

```
// Read and parse metadata content
public UnloadMetadata getUnloadMetadata(UnloadResponse unloadResponse) throws
 IOException {
   AmazonS3URI s3URI = new AmazonS3URI(unloadResponse.getMetadataFile());
   S3Object s3Object = s3Client.getObject(s3URI.getBucket(), s3URI.getKey());
    String metadataFileContent = new
 String(IOUtils.toByteArray(s3Object.getObjectContent()), StandardCharsets.UTF_8);
    final Gson gson = new GsonBuilder()
            .setFieldNamingPolicy(FieldNamingPolicy.UPPER_CAMEL_CASE)
            .create();
    return gson.fromJson(metadataFileContent, UnloadMetadata.class);
}

class UnloadMetadata {
    @JsonProperty("ColumnInfo")
    List<ColumnInfo> columnInfo;
    @JsonProperty("Author")
    Author author;

    @Data
    public class Author {
        @JsonProperty("Name")
        String name;
        @JsonProperty("MetadataFileVersion")
        String metadataFileVersion;
    }
}
```

Java v2

```
// Read and parse metadata content

public UnloadMetadata getUnloadMetadata(UnloadResponse unloadResponse) throws
 URISyntaxException {
   // Space needs to encoded to use S3 parseUri function
    S3Uri s3Uri =
 s3Utilities.parseUri(URI.create(unloadResponse.getMetadataFile().replace(" ",
 "%20")));
    ResponseBytes<GetObjectResponse> objectBytes =
 s3Client.getObjectAsBytes(GetObjectRequest.builder()
            .bucket(s3Uri.bucket().orElseThrow(() -> new
 URISyntaxException(unloadResponse.getMetadataFile(), "Invalid S3 URI")))
            .key(s3Uri.key().orElseThrow(() -> new
 URISyntaxException(unloadResponse.getMetadataFile(), "Invalid S3 URI")))
            .build());
    String metadataFileContent = new String(objectBytes.asByteArray(),
 StandardCharsets.UTF_8);
    final Gson gson = new GsonBuilder()
            .setFieldNamingPolicy(FieldNamingPolicy.UPPER_CAMEL_CASE)
            .create();
    return gson.fromJson(metadataFileContent, UnloadMetadata.class);
}

class UnloadMetadata {
    @JsonProperty("ColumnInfo")
    List<ColumnInfo> columnInfo;
    @JsonProperty("Author")
    Author author;

    @Data
    public class Author {
        @JsonProperty("Name")
        String name;
        @JsonProperty("MetadataFileVersion")
        String metadataFileVersion;
    }
}
```

Go

```
// Read and parse metadata content
```

```
func getMetadataFile(s3Svc *s3.S3, response map[string]string) Metadata {
    var metadataBuf = getObject(s3Svc, response["metadataFile"])
    var metadata Metadata
    json.Unmarshal(metadataBuf.Bytes(), &metadata)
    return metadata
}

func getObject(s3Svc *s3.S3, s3Uri string) *bytes.Buffer {
    u,_ := url.Parse(s3Uri)
    getObjectInput := &s3.GetObjectInput{
        Key:    aws.String(u.Path),
        Bucket: aws.String(u.Host),
    }
    getObjectOutput, err := s3Svc.GetObject(getObjectInput)
    if err != nil {
        fmt.Println("Error: %s\n", err.Error())
    }
    buf := new(bytes.Buffer)
    buf.ReadFrom(getObjectOutput.Body)
    return buf
}

// Unload's Metadata structure

type Metadata struct {
    Author interface{}
    ColumnInfo []struct {
        Name string
        Type map[string]string
    }
}
```

Python

```
def __get_metadata_file(self, response):
    metadata = self.get_object(response['metadataFile']).read().decode('utf-8')
    parsed_metadata = json.loads(metadata)
   print("Metadata contents: \n%s" % parsed_metadata)

def get_object(self, uri):
    try:
        bucket, key = uri.replace("s3://", "").split("/", 1)
```

```
        s3_client = boto3.client('s3', region_name=<region>)
        response = s3_client.get_object(Bucket=bucket, Key=key)
        return response['Body']
    except Exception as err:
        print("Failed to get the object for URI:", uri)
        raise err
```

Node.js

```
// Read and parse metadata content
async getMetadataFile(response) {
    let metadata;
    await this.getS3Object(response['metadataFile']).then(
        (data) => {
            metadata = JSON.parse(data);
        }
    );
    return metadata;
}

async getS3Object(uri) {
    const {bucketName, key} = this.getBucketAndKey(uri);
    const params = new GetObjectCommand({
        Bucket: bucketName,
        Key: key
    })
    const response = await this.s3Client.send(params);
    return await response.Body.transformToString();
}

getBucketAndKey(uri) {
    const [bucketName] = uri.replace("s3://", "").split("/", 1);
    const key = uri.replace("s3://", "").split('/').slice(1).join('/');
    return {bucketName, key};
}
```

クエリをキャンセルする

次のコードスニペットを使用してクエリをキャンセルできます。

> **Note**
>
> これらのコードスニペットは、 上の完全なサンプルアプリケーションに基づいています[GitHub](#)。サンプルアプリケーションの開始方法の詳細については、「」を参照してください[サンプルアプリケーション](#)。

Java

```
   public void cancelQuery() {
       System.out.println("Starting query: " + SELECT_ALL_QUERY);
       QueryRequest queryRequest = new QueryRequest();
       queryRequest.setQueryString(SELECT_ALL_QUERY);
       QueryResult queryResult = queryClient.query(queryRequest);

       System.out.println("Cancelling the query: " + SELECT_ALL_QUERY);
       final CancelQueryRequest cancelQueryRequest = new CancelQueryRequest();
       cancelQueryRequest.setQueryId(queryResult.getQueryId());
       try {
           queryClient.cancelQuery(cancelQueryRequest);
           System.out.println("Query has been successfully cancelled");
       } catch (Exception e) {
           System.out.println("Could not cancel the query: " + SELECT_ALL_QUERY + "
= " + e);
       }
   }
```

Java v2

```
   public void cancelQuery() {
       System.out.println("Starting query: " + SELECT_ALL_QUERY);
       QueryRequest queryRequest =
QueryRequest.builder().queryString(SELECT_ALL_QUERY).build();
       QueryResponse queryResponse = timestreamQueryClient.query(queryRequest);

       System.out.println("Cancelling the query: " + SELECT_ALL_QUERY);
       final CancelQueryRequest cancelQueryRequest = CancelQueryRequest.builder()
               .queryId(queryResponse.queryId()).build();
       try {
           timestreamQueryClient.cancelQuery(cancelQueryRequest);
           System.out.println("Query has been successfully cancelled");
       } catch (Exception e) {
```

```
            System.out.println("Could not cancel the query: " + SELECT_ALL_QUERY + "
 = " + e);
        }
    }
```

Go

```
cancelQueryInput := &timestreamquery.CancelQueryInput{
      QueryId: aws.String(*queryOutput.QueryId),
  }

  fmt.Println("Submitting cancellation for the query")
  fmt.Println(cancelQueryInput)

  // submit the query
  cancelQueryOutput, err := querySvc.CancelQuery(cancelQueryInput)

  if err != nil {
      fmt.Println("Error:")
      fmt.Println(err)
  } else {
    fmt.Println("Query has been cancelled successfully")
    fmt.Println(cancelQueryOutput)
  }
```

Python

```
    def cancel_query(self):
        print("Starting query: " + self.SELECT_ALL)
        result = self.client.query(QueryString=self.SELECT_ALL)
        print("Cancelling query: " + self.SELECT_ALL)
        try:
            self.client.cancel_query(QueryId=result['QueryId'])
            print("Query has been successfully cancelled")
        except Exception as err:
            print("Cancelling query failed:", err)
```

Node.js

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
async function tryCancelQuery() {
    const params = {
        QueryString: SELECT_ALL_QUERY
    };
    console.log(`Running query: ${SELECT_ALL_QUERY}`);

    await queryClient.query(params).promise()
        .then(
            async (response) => {
                await cancelQuery(response.QueryId);
            },
            (err) => {
                console.error("Error while executing select all query:", err);
            });
}

async function cancelQuery(queryId) {
    const cancelParams = {
        QueryId: queryId
    };
    console.log(`Sending cancellation for query: ${SELECT_ALL_QUERY}`);
    await queryClient.cancelQuery(cancelParams).promise()
        .then(
            (response) => {
                console.log("Query has been cancelled successfully");
            },
            (err) => {
                console.error("Error while cancelling select all:", err);
            });
}
```

.NET

```
        public async Task CancelQuery()
        {
            Console.WriteLine("Starting query: " + SELECT_ALL_QUERY);
            QueryRequest queryRequest = new QueryRequest();
            queryRequest.QueryString = SELECT_ALL_QUERY;
            QueryResponse queryResponse = await
 queryClient.QueryAsync(queryRequest);

            Console.WriteLine("Cancelling query: " + SELECT_ALL_QUERY);
            CancelQueryRequest cancelQueryRequest = new CancelQueryRequest();
```

```
            cancelQueryRequest.QueryId = queryResponse.QueryId;

            try
            {
                await queryClient.CancelQueryAsync(cancelQueryRequest);
                Console.WriteLine("Query has been successfully cancelled.");
            } catch(Exception e)
            {
                Console.WriteLine("Could not cancel the query: " + SELECT_ALL_QUERY
 + " = " + e);
            }
        }
```

バッチロードタスクの作成

次のコードスニペットを使用してバッチロードタスクを作成できます。

Java

```
package com.example.tryit;

import java.util.Arrays;

import
 software.amazon.awssdk.services.timestreamwrite.model.CreateBatchLoadTaskRequest;
import
 software.amazon.awssdk.services.timestreamwrite.model.CreateBatchLoadTaskResponse;
import software.amazon.awssdk.services.timestreamwrite.model.DataModel;
import software.amazon.awssdk.services.timestreamwrite.model.DataModelConfiguration;
import
 software.amazon.awssdk.services.timestreamwrite.model.DataSourceConfiguration;
import
 software.amazon.awssdk.services.timestreamwrite.model.DataSourceS3Configuration;
import software.amazon.awssdk.services.timestreamwrite.model.DimensionMapping;
import
 software.amazon.awssdk.services.timestreamwrite.model.MultiMeasureAttributeMapping;
import software.amazon.awssdk.services.timestreamwrite.model.MultiMeasureMappings;
import software.amazon.awssdk.services.timestreamwrite.model.ReportConfiguration;
import software.amazon.awssdk.services.timestreamwrite.model.ReportS3Configuration;
import software.amazon.awssdk.services.timestreamwrite.model.ScalarMeasureValueType;
import software.amazon.awssdk.services.timestreamwrite.model.TimeUnit;
import software.amazon.awssdk.services.timestreamwrite.TimestreamWriteClient;
```

```
public class BatchLoadExample {
    public static final String DATABASE_NAME = <database name>;
    public static final String TABLE_NAME = <table name>;
    public static final String INPUT_BUCKET = <S3 location>;
    public static final String INPUT_OBJECT_KEY_PREFIX = <CSV filename>;
    public static final String REPORT_BUCKET = <S3 location>;
    public static final long HT_TTL_HOURS = 24L;
    public static final long CT_TTL_DAYS = 7L;

    TimestreamWriteClient amazonTimestreamWrite;

    public BatchLoadExample(TimestreamWriteClient client) {
        this.amazonTimestreamWrite = client;
    }

    public String createBatchLoadTask() {
        System.out.println("Creating batch load task");

        CreateBatchLoadTaskRequest request = CreateBatchLoadTaskRequest.builder()
                .dataModelConfiguration(DataModelConfiguration.builder()
                        .dataModel(DataModel.builder()
                                .timeColumn("timestamp")
                                .timeUnit(TimeUnit.SECONDS)
                                .dimensionMappings(Arrays.asList(
                                        DimensionMapping.builder()
                                                .sourceColumn("vehicle")
                                                .build(),
                                        DimensionMapping.builder()
                                                .sourceColumn("registration")
                                                .destinationColumn("license")
                                                .build()))
                                .multiMeasureMappings(MultiMeasureMappings.builder()
                                        .targetMultiMeasureName("mva_measure_name")

 .multiMeasureAttributeMappings(Arrays.asList(

 MultiMeasureAttributeMapping.builder()
                                                        .sourceColumn("wgt")

 .targetMultiMeasureAttributeName("weight")

 .measureValueType(ScalarMeasureValueType.DOUBLE)
                                                        .build(),
```

```
MultiMeasureAttributeMapping.builder()
                                                        .sourceColumn("spd")

.targetMultiMeasureAttributeName("speed")

.measureValueType(ScalarMeasureValueType.DOUBLE)
                                                        .build(),

MultiMeasureAttributeMapping.builder()
                                                        .sourceColumn("fuel")

.measureValueType(ScalarMeasureValueType.DOUBLE)
                                                        .build(),

MultiMeasureAttributeMapping.builder()
                                                        .sourceColumn("miles")

.measureValueType(ScalarMeasureValueType.DOUBLE)
                                                        .build()))
                                        .build())
                                .build())
                        .build())
                .dataSourceConfiguration(DataSourceConfiguration.builder()
                        .dataSourceS3Configuration(
                                DataSourceS3Configuration.builder()
                                        .bucketName(INPUT_BUCKET)
                                        .objectKeyPrefix(INPUT_OBJECT_KEY_PREFIX)
                                        .build())
                        .dataFormat("CSV")
                        .build())
                .reportConfiguration(ReportConfiguration.builder()
                        .reportS3Configuration(ReportS3Configuration.builder()
                                .bucketName(REPORT_BUCKET)
                                .build())
                        .build())
                .targetDatabaseName(DATABASE_NAME)
                .targetTableName(TABLE_NAME)
                .build();
        try {
            final CreateBatchLoadTaskResponse createBatchLoadTaskResponse =
amazonTimestreamWrite.createBatchLoadTask(request);
            String taskId = createBatchLoadTaskResponse.taskId();
            System.out.println("Successfully created batch load task: " + taskId);
```

```
            return taskId;
        } catch (Exception e) {
            System.out.println("Failed to create batch load task: " + e);
            throw e;
        }
    }
}
```

Go

```
package main

import (
 "fmt"
 "context"
 "log"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/timestreamwrite"
 "github.com/aws/aws-sdk-go-v2/service/timestreamwrite/types"
)

func main() {
 customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{})(aws.Endpoint, error) {
  if service == timestreamwrite.ServiceID &&  region == "us-west-2" {
      return aws.Endpoint{
          PartitionID:   "aws",
          URL:           <URL>,
          SigningRegion: "us-west-2",
      }, nil
  }
  return aws.Endpoint{}, &  aws.EndpointNotFoundError{}
 })

 cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver), config.WithRegion("us-
west-2"))

 if err != nil {
    log.Fatalf("failed to load configuration, %v", err)
 }
```

```
client := timestreamwrite.NewFromConfig(cfg)

response, err := client.CreateBatchLoadTask(context.TODO(), &
timestreamwrite.CreateBatchLoadTaskInput{
           TargetDatabaseName: aws.String("BatchLoadExampleDatabase"),
           TargetTableName: aws.String("BatchLoadExampleTable"),
           RecordVersion: aws.Int64(1),
           DataModelConfiguration: &  types.DataModelConfiguration{
               DataModel: &  types.DataModel{
                   TimeColumn: aws.String("timestamp"),
                   TimeUnit: types.TimeUnitMilliseconds,
                   DimensionMappings: []types.DimensionMapping{
                       {
                           SourceColumn: aws.String("registration"),
                           DestinationColumn: aws.String("license"),
                       },
                   },
                   MultiMeasureMappings: &  types.MultiMeasureMappings{
                       TargetMultiMeasureName: aws.String("mva_measure_name"),
                       MultiMeasureAttributeMappings:
[]types.MultiMeasureAttributeMapping{
                           {
                               SourceColumn: aws.String("wgt"),
                               TargetMultiMeasureAttributeName:
aws.String("weight"),
                               MeasureValueType:
types.ScalarMeasureValueTypeDouble,
                           },
                           {
                               SourceColumn: aws.String("spd"),
                               TargetMultiMeasureAttributeName:
aws.String("speed"),
                               MeasureValueType:
types.ScalarMeasureValueTypeDouble,
                           },
                           {
                               SourceColumn: aws.String("fuel_consumption"),
                               TargetMultiMeasureAttributeName: aws.String("fuel"),
                               MeasureValueType:
types.ScalarMeasureValueTypeDouble,
                           },
                       },
                   },
               },
```

```
            },
           DataSourceConfiguration: &  types.DataSourceConfiguration{
                DataSourceS3Configuration: &  types.DataSourceS3Configuration{
                    BucketName: aws.String("test-batch-load-west-2"),
                    ObjectKeyPrefix: aws.String("sample.csv"),
                },
                DataFormat: types.BatchLoadDataFormatCsv,
            },
           ReportConfiguration: & types.ReportConfiguration{
                ReportS3Configuration: & types.ReportS3Configuration{
                    BucketName: aws.String("test-batch-load-report-west-2"),
                    EncryptionOption: types.S3EncryptionOptionSseS3,
                },
            },
 })

 fmt.Println(aws.ToString(response.TaskId))
}
```

Python

```
import boto3
from botocore.config import Config

INGEST_ENDPOINT = "<URL>"
REGION = "us-west-2"
HT_TTL_HOURS = 24
CT_TTL_DAYS = 7
DATABASE_NAME = "<database name>"
TABLE_NAME = "<table name>"
INPUT_BUCKET_NAME = "<S3 location>"
INPUT_OBJECT_KEY_PREFIX = "<CSV file name>"
REPORT_BUCKET_NAME = "<S3 location>"


def create_batch_load_task(client, database_name, table_name, input_bucket_name,
 input_object_key_prefix, report_bucket_name):
    try:
        result = client.create_batch_load_task(TargetDatabaseName=database_name,
 TargetTableName=table_name,
                                               DataModelConfiguration={"DataModel":
 {
                                                   "TimeColumn": "timestamp",
```

```
                                                    "TimeUnit": "SECONDS",
                                                    "DimensionMappings": [
                                                        {
                                                            "SourceColumn": "vehicle"
                                                        },
                                                        {
                                                            "SourceColumn":
"registration",
                                                            "DestinationColumn":
"license"
                                                        }
                                                    ],
                                                    "MultiMeasureMappings": {
                                                        "TargetMultiMeasureName":
"metrics",

"MultiMeasureAttributeMappings": [
                                                            {
                                                                "SourceColumn":
"wgt",
                                                                "MeasureValueType":
"DOUBLE"
                                                            },
                                                            {
                                                                "SourceColumn":
"spd",
                                                                "MeasureValueType":
"DOUBLE"
                                                            },
                                                            {
                                                                "SourceColumn":
"fuel_consumption",

"TargetMultiMeasureAttributeName": "fuel",
                                                                "MeasureValueType":
"DOUBLE"
                                                            },
                                                            {
                                                                "SourceColumn":
"miles",
                                                                "MeasureValueType":
"DOUBLE"
                                                            }
                                                        ]}
```

```
                                                    }
                                                    },
                                                    DataSourceConfiguration={
                                                        "DataSourceS3Configuration": {
                                                            "BucketName":
 input_bucket_name,
                                                            "ObjectKeyPrefix":
 input_object_key_prefix
                                                        },
                                                        "DataFormat": "CSV"
                                                    },
                                                    ReportConfiguration={
                                                        "ReportS3Configuration": {
                                                            "BucketName":
 report_bucket_name,
                                                            "EncryptionOption": "SSE_S3"
                                                        }
                                                    }
                                                    )

        task_id = result["TaskId"]
        print("Successfully created batch load task: ", task_id)
        return task_id
    except Exception as err:
        print("Create batch load task job failed:", err)
        return None


if __name__ == '__main__':
    session = boto3.Session()

    write_client = session.client('timestream-write',
                                  endpoint_url=INGEST_ENDPOINT, region_name=REGION,
                                  config=Config(read_timeout=20,
 max_pool_connections=5000, retries={'max_attempts': 10}))

    task_id = create_batch_load_task(write_client, DATABASE_NAME, TABLE_NAME,
                                     INPUT_BUCKET_NAME, INPUT_OBJECT_KEY_PREFIX,
 REPORT_BUCKET_NAME)
```

Node.js

次のスニペットは JavaScript v3 AWSSDKに を使用します。クライアントのインストール方法と使用状況の詳細については、 JavaScript 「v3 AWSSDKの Timestream Write Client -」を参照してください。

API 詳細については、「クラス CreateBatchLoadCommand」と「」を参照してくださいCreateBatchLoadTask。

```
import { TimestreamWriteClient, CreateBatchLoadTaskCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-west-2", endpoint:
 "https://gamma-ingest-cell3.timestream.us-west-2.amazonaws.com" });

const params = {
    TargetDatabaseName: "BatchLoadExampleDatabase",
    TargetTableName: "BatchLoadExampleTable",
 RecordVersion: 1,
 DataModelConfiguration: {
  DataModel: {
   TimeColumn: "timestamp",
            TimeUnit: "MILLISECONDS",
            DimensionMappings: [
                {
                    SourceColumn: "registration",
                    DestinationColumn: "license"
                }
            ],
            MultiMeasureMappings: {
                TargetMultiMeasureName: "mva_measure_name",
                MultiMeasureAttributeMappings: [
                    {
                        SourceColumn: "wgt",
                        TargetMultiMeasureAttributeName: "weight",
                        MeasureValueType: "DOUBLE"
                    },
                    {
                        SourceColumn: "spd",
                        TargetMultiMeasureAttributeName: "speed",
                        MeasureValueType: "DOUBLE"
                    },
                    {
                        SourceColumn: "fuel_consumption",
```

```
                        TargetMultiMeasureAttributeName: "fuel",
                        MeasureValueType: "DOUBLE"
                    }
                ]
            }
        }
    },
 DataSourceConfiguration: {
        DataSourceS3Configuration: {
            BucketName: "test-batch-load-west-2",
            ObjectKeyPrefix: "sample.csv"
        },
        DataFormat: "CSV"
    },
    ReportConfiguration: {
        ReportS3Configuration: {
            BucketName: "test-batch-load-report-west-2",
            EncryptionOption: "SSE_S3"
        }
    }
};

const command = new CreateBatchLoadTaskCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Created batch load task ` + data.TaskId);
} catch (error) {
 console.log("Error creating table. ", error);
 throw error;
}
```

.NET

```
using System;
using System.IO;
using System.Collections.Generic;
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using System.Threading.Tasks;

namespace TimestreamDotNetSample
{
```

```
public class CreateBatchLoadTaskExample
{
    public const string DATABASE_NAME = "<database name>";
    public const string TABLE_NAME = "<table name>";
    public const string INPUT_BUCKET = "<input bucket name>";
    public const string INPUT_OBJECT_KEY_PREFIX = "<CSV file name>";
    public const string REPORT_BUCKET = "<report bucket name>";
    public const long HT_TTL_HOURS = 24L;
    public const long CT_TTL_DAYS = 7L;
    private readonly AmazonTimestreamWriteClient writeClient;

    public CreateBatchLoadTaskExample(AmazonTimestreamWriteClient writeClient)
    {
        this.writeClient = writeClient;
    }

    public async Task CreateBatchLoadTask()
    {
        try
        {
            var createBatchLoadTaskRequest = new CreateBatchLoadTaskRequest
            {
                DataModelConfiguration = new DataModelConfiguration
                {
                    DataModel = new DataModel
                    {
                        TimeColumn = "timestamp",
                        TimeUnit = TimeUnit.SECONDS,
                        DimensionMappings = new List<DimensionMapping>()
                        {
                            new()
                            {
                                    SourceColumn = "vehicle"
                            },
                            new()
                            {
                                    SourceColumn = "registration",
                                    DestinationColumn = "license"
                            }
                        },
                        MultiMeasureMappings = new MultiMeasureMappings
                        {
                            TargetMultiMeasureName = "mva_measure_name",
```

```
                                MultiMeasureAttributeMappings = new
List<MultiMeasureAttributeMapping>()
                                {
                                        new()
                                        {
                                                SourceColumn = "wgt",
                                                TargetMultiMeasureAttributeName =
"weight",
                                                MeasureValueType =
ScalarMeasureValueType.DOUBLE
                                        },
                                        new()
                                        {
                                                SourceColumn = "spd",
                                                TargetMultiMeasureAttributeName =
"speed",
                                                MeasureValueType =
ScalarMeasureValueType.DOUBLE
                                        },
                                        new()
                                        {
                                                SourceColumn = "fuel",
                                                TargetMultiMeasureAttributeName =
"fuel",
                                                MeasureValueType =
ScalarMeasureValueType.DOUBLE
                                        },
                                        new()
                                        {
                                                SourceColumn = "miles",
                                                TargetMultiMeasureAttributeName =
"miles",
                                                MeasureValueType =
ScalarMeasureValueType.DOUBLE
                                        }
                                }
                            }
                        }
                    },
                    DataSourceConfiguration = new DataSourceConfiguration
                    {
                        DataSourceS3Configuration = new DataSourceS3Configuration
                        {
                            BucketName = INPUT_BUCKET,
```

```
                            ObjectKeyPrefix = INPUT_OBJECT_KEY_PREFIX
                        },
                        DataFormat = "CSV"
                    },
                    ReportConfiguration = new ReportConfiguration
                    {
                        ReportS3Configuration = new ReportS3Configuration
                        {
                            BucketName = REPORT_BUCKET
                        }
                    },
                    TargetDatabaseName = DATABASE_NAME,
                    TargetTableName = TABLE_NAME
                };

                CreateBatchLoadTaskResponse response = await
 writeClient.CreateBatchLoadTaskAsync(createBatchLoadTaskRequest);
                Console.WriteLine($"Task created: " + response.TaskId);
            }
            catch (Exception e)
            {
                Console.WriteLine("Create batch load task failed:" + e.ToString());
            }
        }
    }
}
```

```
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using Amazon;
using Amazon.TimestreamQuery;
using System.Threading.Tasks;
using System;
using CommandLine;
static class Constants
{

}
namespace TimestreamDotNetSample
{
    class MainClass
    {
        public class Options
```

```
        {

        }
        public static void Main(string[] args)
        {
            Parser.Default.ParseArguments<Options>(args)
                .WithParsed<Options>(o => {
                    MainAsync().GetAwaiter().GetResult();
                });
        }

        static async Task MainAsync()
        {
            var writeClientConfig = new AmazonTimestreamWriteConfig
            {
                ServiceURL =  "<service URL>",
                Timeout = TimeSpan.FromSeconds(20),
                MaxErrorRetry = 10
            };

            var writeClient = new AmazonTimestreamWriteClient(writeClientConfig);
            var example = new CreateBatchLoadTaskExample(writeClient);
            await example.CreateBatchLoadTask();
        }
    }
}
```

バッチロードタスクの説明

次のコードスニペットを使用して、バッチロードタスクを記述できます。

Java

```
   public void describeBatchLoadTask(String taskId) {
           final DescribeBatchLoadTaskResponse batchLoadTaskResponse =
 amazonTimestreamWrite

 .describeBatchLoadTask(DescribeBatchLoadTaskRequest.builder()
                                            .taskId(taskId)
                                            .build());
```

```
            System.out.println("Task id: " +
 batchLoadTaskResponse.batchLoadTaskDescription().taskId());
            System.out.println("Status: " +
 batchLoadTaskResponse.batchLoadTaskDescription().taskStatusAsString());
            System.out.println("Records processed: "
                               +
 batchLoadTaskResponse.batchLoadTaskDescription().progressReport().recordsProcessed());
    }
```

Go

```
package main

import (
 "fmt"
 "context"
 "log"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/timestreamwrite"
)

func main() {
 customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{}) (aws.Endpoint, error) {
  if service == timestreamwrite.ServiceID && region == "us-west-2" {
      return aws.Endpoint{
          PartitionID:   "aws",
          URL:           <URL>,
          SigningRegion: "us-west-2",
      }, nil
  }
  return aws.Endpoint{}, &aws.EndpointNotFoundError{}
 })

 cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver), config.WithRegion("us-
west-2"))

 if err != nil {
    log.Fatalf("failed to load configuration, %v", err)
 }
```

```
 client := timestreamwrite.NewFromConfig(cfg)

 response, err := client.DescribeBatchLoadTask(context.TODO(),
 &timestreamwrite.DescribeBatchLoadTaskInput{
  TaskId: aws.String("<TaskId>"),
 })

 fmt.Println(aws.ToString(response.BatchLoadTaskDescription.TaskId))
}
```

Python

```
import boto3
from botocore.config import Config

INGEST_ENDPOINT="<url>"
REGION="us-west-2"
HT_TTL_HOURS = 24
CT_TTL_DAYS = 7
TASK_ID = "<task id>"

def describe_batch_load_task(client, task_id):
    try:
        result = client.describe_batch_load_task(TaskId=task_id)
        print("Successfully described batch load task: ", result)
    except Exception as err:
        print("Describe batch load task job failed:", err)


if __name__ == '__main__':
    session = boto3.Session()

    write_client = session.client('timestream-write', \
        endpoint_url=INGEST_ENDPOINT, region_name=REGION, \
        config=Config(read_timeout=20, max_pool_connections = 5000,
 retries={'max_attempts': 10}))

    describe_batch_load_task(write_client, TASK_ID)
```

Node.js

次のスニペットは JavaScript v3 AWSSDKに を使用します。クライアントのインストール方法と使用状況の詳細については、 JavaScript 「v3 AWSSDKの Timestream Write Client -」を参照してください。

API 詳細については、「クラス DescribeBatchLoadCommand」と「」を参照してくださいDescribeBatchLoadTask。

```
import { TimestreamWriteClient, DescribeBatchLoadTaskCommand } from "@aws-sdk/
client-timestream-write";
const writeClient = new TimestreamWriteClient({ region: "<region>", endpoint:
 "<endpoint>" });

const params = {
    TaskId: "<TaskId>"
};

const command = new DescribeBatchLoadTaskCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Batch load task has id ` + data.BatchLoadTaskDescription.TaskId);
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log("Batch load task doesn't exist.");
    } else {
        console.log("Describe batch load task failed.", error);
        throw error;
    }
}
```

.NET

```
using System;
using System.IO;
using System.Collections.Generic;
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using System.Threading.Tasks;

namespace TimestreamDotNetSample
{
```

```
    public class DescribeBatchLoadTaskExample
    {
        private readonly AmazonTimestreamWriteClient writeClient;

        public DescribeBatchLoadTaskExample(AmazonTimestreamWriteClient writeClient)
        {
            this.writeClient = writeClient;
        }

        public async Task DescribeBatchLoadTask(String taskId)
        {
            try
            {
                var describeBatchLoadTaskRequest = new DescribeBatchLoadTaskRequest
                {
                    TaskId = taskId
                };
                DescribeBatchLoadTaskResponse response = await
 writeClient.DescribeBatchLoadTaskAsync(describeBatchLoadTaskRequest);
                Console.WriteLine($"Task has id:
{response.BatchLoadTaskDescription.TaskId}");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine("Batch load task does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Describe batch load task failed:" +
 e.ToString());
            }
        }
    }
}
```

```
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using Amazon;
using Amazon.TimestreamQuery;
using System.Threading.Tasks;
using System;
using CommandLine;
static class Constants
```

```
{

}
namespace TimestreamDotNetSample
{
    class MainClass
    {
        public class Options
        {

        }
        public static void Main(string[] args)
        {
            Parser.Default.ParseArguments<Options>(args)
                .WithParsed<Options>(o => {
                    MainAsync().GetAwaiter().GetResult();
                });
        }

        static async Task MainAsync()
        {
            var writeClientConfig = new AmazonTimestreamWriteConfig
            {
                ServiceURL =  "<service URL>",
                Timeout = TimeSpan.FromSeconds(20),
                MaxErrorRetry = 10
            };

            var writeClient = new AmazonTimestreamWriteClient(writeClientConfig);
            var example = new DescribeBatchLoadTaskExample(writeClient);
            await example.DescribeBatchLoadTask("<batch load task id>");
        }
    }
}
```

バッチロードタスクを一覧表示する

次のコードスニペットを使用して、バッチロードタスクを一覧表示できます。

Java

```
    public void listBatchLoadTasks() {
```

```
            final ListBatchLoadTasksResponse listBatchLoadTasksResponse =
 amazonTimestreamWrite
                            .listBatchLoadTasks(ListBatchLoadTasksRequest.builder()
                                                .maxResults(15)
                                                .build());

            for (BatchLoadTask batchLoadTask :
 listBatchLoadTasksResponse.batchLoadTasks()) {
                    System.out.println(batchLoadTask.taskId());
            }
    }
```

Go

```
package main

import (
 "fmt"
 "context"
 "log"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/timestreamwrite"
)

func main() {
 customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{}) (aws.Endpoint, error) {
  if service == timestreamwrite.ServiceID && region == "us-west-2" {
      return aws.Endpoint{
          PartitionID:   "aws",
          URL:           <URL>,
          SigningRegion: "us-west-2",
      }, nil
  }
  return aws.Endpoint{}, &aws.EndpointNotFoundError{}
 })

 cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver), config.WithRegion("us-
west-2"))

 if err != nil {
```

```
    log.Fatalf("failed to load configuration, %v", err)
 }

 client := timestreamwrite.NewFromConfig(cfg)
 listBatchLoadTasksMaxResult := int32(15)

 response, err := client.ListBatchLoadTasks(context.TODO(),
 &timestreamwrite.ListBatchLoadTasksInput{
  MaxResults: &listBatchLoadTasksMaxResult,
 })

 for i, task := range response.BatchLoadTasks {
  fmt.Println(i, aws.ToString(task.TaskId))
 }
}
```

Python

```
import boto3
from botocore.config import Config

INGEST_ENDPOINT = "<url>"
REGION = "us-west-2"
HT_TTL_HOURS = 24
CT_TTL_DAYS = 7


def print_batch_load_tasks(batch_load_tasks):
    for batch_load_task in batch_load_tasks:
        print(batch_load_task['TaskId'])


def list_batch_load_tasks(client):
    print("\nListing batch load tasks")
    try:
        response = client.list_batch_load_tasks(MaxResults=10)
        print_batch_load_tasks(response['BatchLoadTasks'])
        next_token = response.get('NextToken', None)
        while next_token:
            response = client.list_batch_load_tasks(
                NextToken=next_token, MaxResults=10)
            print_batch_load_tasks(response['BatchLoadTasks'])
            next_token = response.get('NextToken', None)
```

```
    except Exception as err:
        print("List batch load tasks failed:", err)
        raise err


if __name__ == '__main__':
    session = boto3.Session()

    write_client = session.client('timestream-write',
                                  endpoint_url=INGEST_ENDPOINT, region_name=REGION,
                                  config=Config(read_timeout=20,
 max_pool_connections=5000, retries={'max_attempts': 10}))

    list_batch_load_tasks(write_client)
```

Node.js

次のスニペットは JavaScript v3 AWSSDKに を使用します。クライアントのインストール方法と使用状況の詳細については、 JavaScript 「v3 AWSSDKの Timestream Write Client -」を参照してください。

API 詳細については、「クラス DescribeBatchLoadCommand」と「」を参照してくださいDescribeBatchLoadTask。

```
import { TimestreamWriteClient, ListBatchLoadTasksCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "<region>", endpoint:
 "<endpoint>" });

const params = {
    MaxResults: <15>
};

const command = new ListBatchLoadTasksCommand(params);

getBatchLoadTasksList(null);

async function getBatchLoadTasksList(nextToken) {
    if (nextToken) {
        params.NextToken = nextToken;
    }

    try {
```

```
        const data = await writeClient.send(command);

        data.BatchLoadTasks.forEach(function (task) {
            console.log(task.TaskId);
        });

        if (data.NextToken) {
            return getBatchLoadTasksList(data.NextToken);
        }
    } catch (error) {
        console.log("Error while listing batch load tasks", error);
    }
}
```

.NET

```
using System;
using System.IO;
using System.Collections.Generic;
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using System.Threading.Tasks;

namespace TimestreamDotNetSample
{
    public class ListBatchLoadTasksExample
    {
        private readonly AmazonTimestreamWriteClient writeClient;

        public ListBatchLoadTasksExample(AmazonTimestreamWriteClient writeClient)
        {
            this.writeClient = writeClient;
        }

        public async Task ListBatchLoadTasks()
        {
            Console.WriteLine("Listing batch load tasks");

            try
            {
                var listBatchLoadTasksRequest = new ListBatchLoadTasksRequest
                {
                    MaxResults = 15
```

```
                };

                ListBatchLoadTasksResponse response = await
 writeClient.ListBatchLoadTasksAsync(listBatchLoadTasksRequest);

                PrintBatchLoadTasks(response.BatchLoadTasks);
                var nextToken = response.NextToken;

                while (nextToken != null)
                {
                    listBatchLoadTasksRequest.NextToken = nextToken;
                    response = await
 writeClient.ListBatchLoadTasksAsync(listBatchLoadTasksRequest);
                    PrintBatchLoadTasks(response.BatchLoadTasks);
                    nextToken = response.NextToken;
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("List batch load tasks failed:" + e.ToString());
            }
        }

        private void PrintBatchLoadTasks(List<BatchLoadTask> tasks)
        {
            foreach (BatchLoadTask task in tasks)
                Console.WriteLine($"Task:{task.TaskId}");
        }
    }
}
```

```
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using Amazon;
using Amazon.TimestreamQuery;
using System.Threading.Tasks;
using System;
using CommandLine;
static class Constants
{

}
namespace TimestreamDotNetSample
```

```
{
    class MainClass
    {
        public class Options
        {

        }
        public static void Main(string[] args)
        {
            Parser.Default.ParseArguments<Options>(args)
                .WithParsed<Options>(o => {
                    MainAsync().GetAwaiter().GetResult();
                });
        }

        static async Task MainAsync()
        {
            var writeClientConfig = new AmazonTimestreamWriteConfig
            {
                ServiceURL =  "<service URL>",
                Timeout = TimeSpan.FromSeconds(20),
                MaxErrorRetry = 10
            };

            var writeClient = new AmazonTimestreamWriteClient(writeClientConfig);
            var example = new ListBatchLoadTasksExample(writeClient);
            await example.ListBatchLoadTasks();
        }
    }
}
```

バッチロードタスクを再開する

次のコードスニペットを使用して、バッチロードタスクを再開できます。

Java

```
    public void resumeBatchLoadTask(String taskId) {
            try {
                    amazonTimestreamWrite

 .resumeBatchLoadTask(ResumeBatchLoadTaskRequest.builder()
```

```
                                                    .taskId(taskId)
                                                    .build());

                    System.out.println("Successfully resumed batch load task.");
            } catch (ValidationException validationException) {
                    System.out.println(validationException.getMessage());
            }
    }
```

Go

```
package main

import (
 "fmt"
 "context"
 "log"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/timestreamwrite"
)

func main() {
 customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{}) (aws.Endpoint, error) {
  if service == timestreamwrite.ServiceID && region == "us-west-2" {
      return aws.Endpoint{
          PartitionID:   "aws",
          URL:           <URL>,
          SigningRegion: "us-west-2",
      }, nil
  }
  return aws.Endpoint{}, &aws.EndpointNotFoundError{}
 })

 cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver), config.WithRegion("us-
west-2"))

 if err != nil {
    log.Fatalf("failed to load configuration, %v", err)
 }
```

```
 client := timestreamwrite.NewFromConfig(cfg)

 response, err := client.ResumeBatchLoadTask(context.TODO(),
 &timestreamwrite.ResumeBatchLoadTaskInput{
  TaskId: aws.String("TaskId"),
 })

 if err != nil {
  fmt.Println("Error:")
  fmt.Println(err)
 } else {
  fmt.Println("Resume batch load task is successful")
  fmt.Println(response)
 }
}
```

Python

```
import boto3
from botocore.config import Config

INGEST_ENDPOINT="<url>"
REGION="us-west-2"
HT_TTL_HOURS = 24
CT_TTL_DAYS = 7
TASK_ID = "<TaskId>"

def resume_batch_load_task(client, task_id):
    try:
        result = client.resume_batch_load_task(TaskId=task_id)
        print("Successfully resumed batch load task: ", result)
    except Exception as err:
        print("Resume batch load task failed:", err)


if __name__ == '__main__':
    session = boto3.Session()

    write_client = session.client('timestream-write', \
        endpoint_url=INGEST_ENDPOINT, region_name=REGION, \
        config=Config(read_timeout=20, max_pool_connections = 5000,
 retries={'max_attempts': 10}))
```

```
    resume_batch_load_task(write_client, TASK_ID)
```

Node.js

次のスニペットは JavaScript v3 AWSSDKに を使用します。クライアントのインストール方法と使用状況の詳細については、JavaScript「v3 AWSSDKの Timestream Write Client -」を参照してください。

API 詳細については、「クラス CreateBatchLoadCommand」と「」を参照してくださいCreateBatchLoadTask。

```
import { TimestreamWriteClient, ResumeBatchLoadTaskCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "<region>", endpoint:
 "<endpoint>" });

const params = {
    TaskId: "<TaskId>"
};

const command = new ResumeBatchLoadTaskCommand(params);

try {
    const data = await writeClient.send(command);
    console.log("Resumed batch load task");
} catch (error) {
    console.log("Resume batch load task failed.", error);
 throw error;
}
```

.NET

```
using System;
using System.IO;
using System.Collections.Generic;
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using System.Threading.Tasks;

namespace TimestreamDotNetSample
{
    public class ResumeBatchLoadTaskExample
    {
```

```
        private readonly AmazonTimestreamWriteClient writeClient;

        public ResumeBatchLoadTaskExample(AmazonTimestreamWriteClient writeClient)
        {
            this.writeClient = writeClient;
        }

        public async Task ResumeBatchLoadTask(String taskId)
        {
            try
            {
                var resumeBatchLoadTaskRequest = new ResumeBatchLoadTaskRequest
                {
                    TaskId = taskId
                };
                ResumeBatchLoadTaskResponse response = await
 writeClient.ResumeBatchLoadTaskAsync(resumeBatchLoadTaskRequest);
                Console.WriteLine("Successfully resumed batch load task.");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine("Batch load task does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Resume batch load task failed: " + e.ToString());
            }
        }
    }
}
```

スケジュールされたクエリを作成する

次のコードスニペットを使用して、マルチメジャーマッピングでスケジュールされたクエリを作成できます。

Java

```
public static String DATABASE_NAME = "devops_sample_application";
public static String TABLE_NAME = "host_metrics_sample_application";
public static String HOSTNAME = "host-24Gju";
public static String SQ_NAME = "daily-sample";
```

```
public static String SCHEDULE_EXPRESSION = "cron(0/2 * * * ? *)";

// Find the average, p90, p95, and p99 CPU utilization for a specific EC2 host over
 the past 2 hours.
public static String QUERY = "SELECT region, az, hostname, BIN(time, 15s) AS
 binned_timestamp, " +
"ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization " +
"FROM " +  DATABASE_NAME + "." +  TABLE_NAME + " " +
"WHERE measure_name = 'metrics' " +
"AND hostname = '" + HOSTNAME + "' " +
"AND time > ago(2h) " +
"GROUP BY region, hostname, az, BIN(time, 15s) " +
"ORDER BY binned_timestamp ASC " +
"LIMIT 5";


public String createScheduledQuery(String topic_arn,
    String role_arn,
    String database_name,
    String table_name) {
    System.out.println("Creating Scheduled Query");

    List<Pair<String, MeasureValueType>> sourceColToMeasureValueTypes =
 Arrays.asList(
        Pair.of("avg_cpu_utilization", DOUBLE),
        Pair.of("p90_cpu_utilization", DOUBLE),
        Pair.of("p95_cpu_utilization", DOUBLE),
        Pair.of("p99_cpu_utilization", DOUBLE));

    CreateScheduledQueryRequest createScheduledQueryRequest = new
 CreateScheduledQueryRequest()
            .withName(SQ_NAME)
            .withQueryString(QUERY)
            .withScheduleConfiguration(new ScheduleConfiguration()
                    .withScheduleExpression(SCHEDULE_EXPRESSION))
            .withNotificationConfiguration(new NotificationConfiguration()
                    .withSnsConfiguration(new SnsConfiguration()
                            .withTopicArn(topic_arn)))
            .withTargetConfiguration(new
 TargetConfiguration().withTimestreamConfiguration(new TimestreamConfiguration()
                    .withDatabaseName(database_name)
```

```
                    .withTableName(table_name)
                    .withTimeColumn("binned_timestamp")
                    .withDimensionMappings(Arrays.asList(
                            new DimensionMapping()
                                    .withName("region")
                                    .withDimensionValueType("VARCHAR"),
                            new DimensionMapping()
                                    .withName("az")
                                    .withDimensionValueType("VARCHAR"),
                            new DimensionMapping()
                                    .withName("hostname")
                                    .withDimensionValueType("VARCHAR")
                    ))
                    .withMultiMeasureMappings(new MultiMeasureMappings()
                        .withTargetMultiMeasureName("multi-metrics")
                        .withMultiMeasureAttributeMappings(
                            sourceColToMeasureValueTypes.stream()
                            .map(pair -> new MultiMeasureAttributeMapping()
                                .withMeasureValueType(pair.getValue().name())
                                .withSourceColumn(pair.getKey()))
                            .collect(Collectors.toList()))))
            .withErrorReportConfiguration(new ErrorReportConfiguration()
                    .withS3Configuration(new S3Configuration()

 .withBucketName(timestreamDependencyHelper.getS3ErrorReportBucketName())))
            .withScheduledQueryExecutionRoleArn(role_arn);

    try {
        final CreateScheduledQueryResult createScheduledQueryResult =
 queryClient.createScheduledQuery(createScheduledQueryRequest);
        final String scheduledQueryArn = createScheduledQueryResult.getArn();
        System.out.println("Successfully created scheduled query : " +
 scheduledQueryArn);
        return scheduledQueryArn;
    }
    catch (Exception e) {
        System.out.println("Scheduled Query creation failed: " + e);
        throw e;
    }
}
```

Java v2

```
public static String DATABASE_NAME = "testJavaV2DB";
public static String TABLE_NAME = "testJavaV2Table";
public static String HOSTNAME = "host-24Gju";
public static String SQ_NAME = "daily-sample";
public static String SCHEDULE_EXPRESSION = "cron(0/2 * * * ? *)";

// Find the average, p90, p95, and p99 CPU utilization for a specific EC2 host over
 the past 2 hours.
public static String VALID_QUERY = "SELECT region, az, hostname, BIN(time, 15s) AS
 binned_timestamp, " +
"ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization " +
"FROM " +  DATABASE_NAME + "." +  TABLE_NAME + " " +
"WHERE measure_name = 'metrics' " +
"AND hostname = '" + HOSTNAME + "' " +
"AND time > ago(2h) " +
"GROUP BY region, hostname, az, BIN(time, 15s) " +
"ORDER BY binned_timestamp ASC " +
"LIMIT 5";


private String createScheduledQueryHelper(String topicArn, String roleArn,
        String s3ErrorReportBucketName, String query,
        TargetConfiguration targetConfiguration) {
    System.out.println("Creating Scheduled Query");

    CreateScheduledQueryRequest createScheduledQueryRequest =
 CreateScheduledQueryRequest.builder()
            .name(SQ_NAME)
            .queryString(query)
            .scheduleConfiguration(ScheduleConfiguration.builder()
                    .scheduleExpression(SCHEDULE_EXPRESSION)
                    .build())
            .notificationConfiguration(NotificationConfiguration.builder()
                    .snsConfiguration(SnsConfiguration.builder()
                            .topicArn(topicArn)
                            .build())
                    .build())
            .targetConfiguration(targetConfiguration)
            .errorReportConfiguration(ErrorReportConfiguration.builder()
```

```
                    .s3Configuration(S3Configuration.builder()
                            .bucketName(s3ErrorReportBucketName)
                            .objectKeyPrefix(SCHEDULED_QUERY_EXAMPLE)
                            .build())
                    .build())
            .scheduledQueryExecutionRoleArn(roleArn)
            .build();

    try {
        final CreateScheduledQueryResponse response =
 queryClient.createScheduledQuery(createScheduledQueryRequest);
        final String scheduledQueryArn = response.arn();
        System.out.println("Successfully created scheduled query : " +
 scheduledQueryArn);
        return scheduledQueryArn;
    }
    catch (Exception e) {
        System.out.println("Scheduled Query creation failed: " + e);
        throw e;
    }
}

public String createScheduledQuery(String topicArn, String roleArn,
        String databaseName, String tableName, String s3ErrorReportBucketName) {
    List<Pair<String, MeasureValueType>> sourceColToMeasureValueTypes =
 Arrays.asList(
            Pair.of("avg_cpu_utilization", DOUBLE),
            Pair.of("p90_cpu_utilization", DOUBLE),
            Pair.of("p95_cpu_utilization", DOUBLE),
            Pair.of("p99_cpu_utilization", DOUBLE));

    TargetConfiguration targetConfiguration = TargetConfiguration.builder()
            .timestreamConfiguration(TimestreamConfiguration.builder()
            .databaseName(databaseName)
            .tableName(tableName)
            .timeColumn("binned_timestamp")
            .dimensionMappings(Arrays.asList(
                    DimensionMapping.builder()
                            .name("region")
                            .dimensionValueType("VARCHAR")
                            .build(),
                    DimensionMapping.builder()
                            .name("az")
                            .dimensionValueType("VARCHAR")
```

```
                            .build(),
                    DimensionMapping.builder()
                            .name("hostname")
                            .dimensionValueType("VARCHAR")
                            .build()
            ))
            .multiMeasureMappings(MultiMeasureMappings.builder()
                    .targetMultiMeasureName("multi-metrics")
                    .multiMeasureAttributeMappings(
                            sourceColToMeasureValueTypes.stream()
                                    .map(pair ->
 MultiMeasureAttributeMapping.builder()

 .measureValueType(pair.getValue().name())
                                            .sourceColumn(pair.getKey())
                                            .build())
                                    .collect(Collectors.toList()))
                    .build())
            .build())
            .build();

    return createScheduledQueryHelper(topicArn, roleArn, s3ErrorReportBucketName,
 VALID_QUERY, targetConfiguration);
}}
```

Go

```
SQ_ERROR_CONFIGURATION_S3_BUCKET_NAME_PREFIX = "sq-error-configuration-sample-s3-
bucket-"
HOSTNAME            = "host-24Gju"
SQ_NAME             = "daily-sample"
SCHEDULE_EXPRESSION = "cron(0/1 * * * ? *)"
QUERY               = "SELECT region, az, hostname, BIN(time, 15s) AS
 binned_timestamp, " +
    "ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " +
    "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, " +
    "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization, " +
    "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization " +
    "FROM %s.%s " +
    "WHERE measure_name = 'metrics' " +
    "AND hostname = '" + HOSTNAME + "' " +
    "AND time > ago(2h) " +
    "GROUP BY region, hostname, az, BIN(time, 15s) " +
```

```
    "ORDER BY binned_timestamp ASC " +
    "LIMIT 5"
s3BucketName = utils.SQ_ERROR_CONFIGURATION_S3_BUCKET_NAME_PREFIX +
 generateRandomStringWithSize(5)

func generateRandomStringWithSize(size int) string {
     rand.Seed(time.Now().UnixNano())
     alphaNumericList := []rune("abcdefghijklmnopqrstuvwxyz0123456789")
     randomPrefix := make([]rune, size)
     for i := range randomPrefix {
         randomPrefix[i] = alphaNumericList[rand.Intn(len(alphaNumericList))]
     }
     return string(randomPrefix)
 }

func (timestreamBuilder TimestreamBuilder) createScheduledQuery(topicArn string,
 roleArn string, s3ErrorReportBucketName string,
query string, targetConfiguration timestreamquery.TargetConfiguration) (string,
 error) {

createScheduledQueryInput := &timestreamquery.CreateScheduledQueryInput{
    Name:        aws.String(SQ_NAME),
    QueryString: aws.String(query),
    ScheduleConfiguration: &timestreamquery.ScheduleConfiguration{
        ScheduleExpression: aws.String(SCHEDULE_EXPRESSION),
    },
    NotificationConfiguration: &timestreamquery.NotificationConfiguration{
        SnsConfiguration: &timestreamquery.SnsConfiguration{
            TopicArn: aws.String(topicArn),
        },
    },
    TargetConfiguration: &targetConfiguration,
    ErrorReportConfiguration: &timestreamquery.ErrorReportConfiguration{
        S3Configuration: &timestreamquery.S3Configuration{
            BucketName: aws.String(s3ErrorReportBucketName),
        },
    },
    ScheduledQueryExecutionRoleArn: aws.String(roleArn),
}

createScheduledQueryOutput, err :=
 timestreamBuilder.QuerySvc.CreateScheduledQuery(createScheduledQueryInput)

if err != nil {
```

```
    fmt.Printf("Error: %s", err.Error())
} else {
    fmt.Println("createScheduledQueryResult is successful")
    return *createScheduledQueryOutput.Arn, nil
}
return "", err
}

 func (timestreamBuilder TimestreamBuilder) CreateValidScheduledQuery(topicArn
 string, roleArn string, s3ErrorReportBucketName string,
     sqDatabaseName string, sqTableName string, databaseName string, tableName
 string) (string, error) {

     targetConfiguration := timestreamquery.TargetConfiguration{
         TimestreamConfiguration: &timestreamquery.TimestreamConfiguration{
             DatabaseName: aws.String(sqDatabaseName),
             TableName:    aws.String(sqTableName),
             TimeColumn:   aws.String("binned_timestamp"),
             DimensionMappings: []*timestreamquery.DimensionMapping{
                 {
                     Name:               aws.String("region"),
                     DimensionValueType: aws.String("VARCHAR"),
                 },
                 {
                     Name:               aws.String("az"),
                     DimensionValueType: aws.String("VARCHAR"),
                 },
                 {
                     Name:               aws.String("hostname"),
                     DimensionValueType: aws.String("VARCHAR"),
                 },
             },
             MultiMeasureMappings: &timestreamquery.MultiMeasureMappings{
                 TargetMultiMeasureName: aws.String("multi-metrics"),
                 MultiMeasureAttributeMappings:
 []*timestreamquery.MultiMeasureAttributeMapping{
                     {
                         SourceColumn:     aws.String("avg_cpu_utilization"),
                         MeasureValueType:
 aws.String(timestreamquery.MeasureValueTypeDouble),
                     },
                     {
                         SourceColumn:     aws.String("p90_cpu_utilization"),
```

```
                        MeasureValueType:
aws.String(timestreamquery.MeasureValueTypeDouble),
                    },
                    {
                        SourceColumn:     aws.String("p95_cpu_utilization"),
                        MeasureValueType:
aws.String(timestreamquery.MeasureValueTypeDouble),
                    },
                    {
                        SourceColumn:     aws.String("p99_cpu_utilization"),
                        MeasureValueType:
aws.String(timestreamquery.MeasureValueTypeDouble),
                    },
                },
            },
        },
    }
    return timestreamBuilder.createScheduledQuery(topicArn, roleArn,
s3ErrorReportBucketName,
        fmt.Sprintf(QUERY, databaseName, tableName), targetConfiguration)
}
```

Python

```
HOSTNAME = "host-24Gju"
SQ_NAME = "daily-sample"
ERROR_BUCKET_NAME = "scheduledquerysamplerrorbucket" +
 ''.join([choice(ascii_lowercase) for _ in range(5)])
QUERY = \
    "SELECT region, az, hostname, BIN(time, 15s) AS binned_timestamp, " \
    "    ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " \
    "    ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, "
 \
    "    ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization,
 " \
    "    ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization "
 \
    "FROM " + database_name + "." + table_name + " " \
    "WHERE measure_name = 'metrics' " \
    "AND hostname = '" + self.HOSTNAME + "' " \
    "AND time > ago(2h) " \
    "GROUP BY region, hostname, az, BIN(time, 15s) " \
    "ORDER BY binned_timestamp ASC " \
```

```
    "LIMIT 5"

def create_scheduled_query_helper(self, topic_arn, role_arn, query,
 target_configuration):
    print("\nCreating Scheduled Query")
    schedule_configuration = {
        'ScheduleExpression': 'cron(0/2 * * * ? *)'
    }
    notification_configuration = {
        'SnsConfiguration': {
            'TopicArn': topic_arn
        }
    }
    error_report_configuration = {
        'S3Configuration': {
            'BucketName': ERROR_BUCKET_NAME
        }
    }

    try:
        create_scheduled_query_response = \
            query_client.create_scheduled_query(Name=self.SQ_NAME,
                 QueryString=query,
                 ScheduleConfiguration=schedule_configuration,
                 NotificationConfiguration=notification_configuration,
                 TargetConfiguration=target_configuration,
                 ScheduledQueryExecutionRoleArn=role_arn,
                 ErrorReportConfiguration=error_report_configuration
                 )
        print("Successfully created scheduled query : ",
 create_scheduled_query_response['Arn'])
        return create_scheduled_query_response['Arn']
    except Exception as err:
        print("Scheduled Query creation failed:", err)
        raise err

def create_valid_scheduled_query(self, topic_arn, role_arn):
    target_configuration = {
        'TimestreamConfiguration': {
            'DatabaseName': self.sq_database_name,
            'TableName': self.sq_table_name,
            'TimeColumn': 'binned_timestamp',
            'DimensionMappings': [
                {'Name': 'region', 'DimensionValueType': 'VARCHAR'},
```

```
                {'Name': 'az', 'DimensionValueType': 'VARCHAR'},
                {'Name': 'hostname', 'DimensionValueType': 'VARCHAR'}
            ],
            'MultiMeasureMappings': {
                'TargetMultiMeasureName': 'target_name',
                'MultiMeasureAttributeMappings': [
                    {'SourceColumn': 'avg_cpu_utilization', 'MeasureValueType':
'DOUBLE',
                     'TargetMultiMeasureAttributeName': 'avg_cpu_utilization'},
                    {'SourceColumn': 'p90_cpu_utilization', 'MeasureValueType':
'DOUBLE',
                     'TargetMultiMeasureAttributeName': 'p90_cpu_utilization'},
                    {'SourceColumn': 'p95_cpu_utilization', 'MeasureValueType':
'DOUBLE',
                     'TargetMultiMeasureAttributeName': 'p95_cpu_utilization'},
                    {'SourceColumn': 'p99_cpu_utilization', 'MeasureValueType':
'DOUBLE',
                     'TargetMultiMeasureAttributeName': 'p99_cpu_utilization'},
                ]
            }
        }
    }

    return self.create_scheduled_query_helper(topic_arn, role_arn, QUERY,
target_configuration)
```

Node.js

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
const DATABASE_NAME = 'devops_sample_application';
const TABLE_NAME = 'host_metrics_sample_application';
const SQ_DATABASE_NAME = 'sq_result_database';
const SQ_TABLE_NAME = 'sq_result_table';
const HOSTNAME = "host-24Gju";
const SQ_NAME = "daily-sample";
const SCHEDULE_EXPRESSION = "cron(0/1 * * * ? *)";

// Find the average, p90, p95, and p99 CPU utilization for a specific EC2 host over
 the past 2 hours.
```

```
const VALID_QUERY = "SELECT region, az, hostname, BIN(time, 15s) AS
 binned_timestamp, " +
    " ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " +
    " ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, " +
    " ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization, " +
    " ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization " +
    "FROM " + DATABASE_NAME + "." + TABLE_NAME + " " +
    "WHERE measure_name = 'metrics' " +
    " AND hostname = '" + HOSTNAME + "' " +
    " AND time > ago(2h) " +
    "GROUP BY region, hostname, az, BIN(time, 15s) " +
    "ORDER BY binned_timestamp ASC " +
    "LIMIT 5";

async function createScheduledQuery(topicArn, roleArn, s3ErrorReportBucketName) {
    console.log("Creating Valid Scheduled Query");
    const DimensionMappingList = [{
            'Name': 'region',
            'DimensionValueType': 'VARCHAR'
        },
        {
            'Name': 'az',
            'DimensionValueType': 'VARCHAR'
        },
        {
            'Name': 'hostname',
            'DimensionValueType': 'VARCHAR'
        }
    ];

    const MultiMeasureMappings = {
        TargetMultiMeasureName: "multi-metrics",
        MultiMeasureAttributeMappings: [{
                'SourceColumn': 'avg_cpu_utilization',
                'MeasureValueType': 'DOUBLE'
            },
            {
                'SourceColumn': 'p90_cpu_utilization',
                'MeasureValueType': 'DOUBLE'
            },
            {
                'SourceColumn': 'p95_cpu_utilization',
                'MeasureValueType': 'DOUBLE'
            },
```

```
            {
                'SourceColumn': 'p99_cpu_utilization',
                'MeasureValueType': 'DOUBLE'
            },
        ]
    }

    const timestreamConfiguration = {
        DatabaseName: SQ_DATABASE_NAME,
        TableName: SQ_TABLE_NAME,
        TimeColumn: "binned_timestamp",
        DimensionMappings: DimensionMappingList,
        MultiMeasureMappings: MultiMeasureMappings
    }

    const createScheduledQueryRequest = {
        Name: SQ_NAME,
        QueryString: VALID_QUERY,
        ScheduleConfiguration: {
            ScheduleExpression: SCHEDULE_EXPRESSION
        },
        NotificationConfiguration: {
            SnsConfiguration: {
                TopicArn: topicArn
            }
        },
        TargetConfiguration: {
            TimestreamConfiguration: timestreamConfiguration
        },
        ScheduledQueryExecutionRoleArn: roleArn,
        ErrorReportConfiguration: {
            S3Configuration: {
                BucketName: s3ErrorReportBucketName
            }
        }
    };
    try {
        const data = await
queryClient.createScheduledQuery(createScheduledQueryRequest).promise();
        console.log("Successfully created scheduled query: " + data.Arn);
        return data.Arn;
    } catch (err) {
        console.log("Scheduled Query creation failed: ", err);
        throw err;
```

```
    }
}
```

.NET

```
public const string Hostname = "host-24Gju";
public const string SqName = "timestream-sample";
public const string SqDatabaseName = "sq_result_database";
public const string SqTableName = "sq_result_table";

public const string ErrorConfigurationS3BucketNamePrefix = "error-configuration-
sample-s3-bucket-";
public const string ScheduleExpression = "cron(0/2 * * * ? *)";

// Find the average, p90, p95, and p99 CPU utilization for a specific EC2 host over
 the past 2 hours.
public const string ValidQuery = "SELECT region, az, hostname, BIN(time, 15s) AS
 binned_timestamp, " +
      "ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " +
      "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, " +
      "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization, "
 +
      "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization " +
      "FROM " + Constants.DATABASE_NAME + "." + Constants.TABLE_NAME + " " +
      "WHERE measure_name = 'metrics' " +
      "AND hostname = '" + Hostname + "' " +
      "AND time > ago(2h) " +
      "GROUP BY region, hostname, az, BIN(time, 15s) " +
      "ORDER BY binned_timestamp ASC " +
      "LIMIT 5";

private async Task<String> CreateValidScheduledQuery(string topicArn, string
 roleArn,
             string databaseName, string tableName, string s3ErrorReportBucketName)
 {
     List<MultiMeasureAttributeMapping> sourceColToMeasureValueTypes =
         new List<MultiMeasureAttributeMapping>()
         {
             new()
             {
                 SourceColumn = "avg_cpu_utilization",
                 MeasureValueType = MeasureValueType.DOUBLE.Value
             },
```

```
            new()
            {
                SourceColumn = "p90_cpu_utilization",
                MeasureValueType = MeasureValueType.DOUBLE.Value
            },
            new()
            {
                SourceColumn = "p95_cpu_utilization",
                MeasureValueType = MeasureValueType.DOUBLE.Value
            },
            new()
            {
                SourceColumn = "p99_cpu_utilization",
                MeasureValueType = MeasureValueType.DOUBLE.Value
            }
        };

    TargetConfiguration targetConfiguration = new TargetConfiguration()
    {
        TimestreamConfiguration = new TimestreamConfiguration()
        {
            DatabaseName = databaseName,
            TableName = tableName,
            TimeColumn = "binned_timestamp",
            DimensionMappings = new List<DimensionMapping>()
            {
                new()
                {
                    Name = "region",
                    DimensionValueType = "VARCHAR"
                },
                new()
                {
                    Name = "az",
                    DimensionValueType = "VARCHAR"
                },
                new()
                {
                    Name = "hostname",
                    DimensionValueType = "VARCHAR"
                }
            },
            MultiMeasureMappings = new MultiMeasureMappings()
            {
```

```
                TargetMultiMeasureName = "multi-metrics",
                MultiMeasureAttributeMappings = sourceColToMeasureValueTypes
            }
        }
    };
    return await CreateScheduledQuery(topicArn, roleArn, s3ErrorReportBucketName,
        ScheduledQueryConstants.ValidQuery, targetConfiguration);
}

private async Task<String> CreateScheduledQuery(string topicArn, string roleArn,
            string s3ErrorReportBucketName, string query, TargetConfiguration
 targetConfiguration)
{
    try
    {
        Console.WriteLine("Creating Scheduled Query");
        CreateScheduledQueryResponse response = await
 _amazonTimestreamQuery.CreateScheduledQueryAsync(
            new CreateScheduledQueryRequest()
            {
                Name = ScheduledQueryConstants.SqName,
                QueryString = query,
                ScheduleConfiguration = new ScheduleConfiguration()
                {
                    ScheduleExpression = ScheduledQueryConstants.ScheduleExpression
                },
                NotificationConfiguration = new NotificationConfiguration()
                {
                    SnsConfiguration = new SnsConfiguration()
                    {
                        TopicArn = topicArn
                    }
                },
                TargetConfiguration = targetConfiguration,
                ErrorReportConfiguration = new ErrorReportConfiguration()
                {
                    S3Configuration = new S3Configuration()
                    {
                        BucketName = s3ErrorReportBucketName
                    }
                },
                ScheduledQueryExecutionRoleArn = roleArn
            });
```

```
        Console.WriteLine($"Successfully created scheduled query :
 {response.Arn}");
        return response.Arn;
    }
    catch (Exception e)
    {
        Console.WriteLine($"Scheduled Query creation failed: {e}");
        throw;
    }
}
```

スケジュールされたクエリを一覧表示する

次のコードスニペットを使用して、スケジュールされたクエリを一覧表示できます。

Java

```
public void listScheduledQueries() {
    System.out.println("Listing Scheduled Query");
    try {
        String nextToken = null;
        List<String> scheduledQueries = new ArrayList<>();

        do {
            ListScheduledQueriesResult listScheduledQueriesResult =
                    queryClient.listScheduledQueries(new
 ListScheduledQueriesRequest()
                            .withNextToken(nextToken).withMaxResults(10));
            List<ScheduledQuery> scheduledQueryList =
 listScheduledQueriesResult.getScheduledQueries();

            printScheduledQuery(scheduledQueryList);
            nextToken = listScheduledQueriesResult.getNextToken();
        } while (nextToken != null);
    }
    catch (Exception e) {
        System.out.println("List Scheduled Query failed: " + e);
        throw e;
    }
}

public void printScheduledQuery(List<ScheduledQuery> scheduledQueryList) {
```

```
    for (ScheduledQuery scheduledQuery: scheduledQueryList) {
        System.out.println(scheduledQuery.getArn());
    }
}
```

Java v2

```
public void listScheduledQueries() {
    System.out.println("Listing Scheduled Query");
    try {
        String nextToken = null;

        do {
            ListScheduledQueriesResponse listScheduledQueriesResult =

 queryClient.listScheduledQueries(ListScheduledQueriesRequest.builder()
                            .nextToken(nextToken).maxResults(10)
                            .build());
            List<ScheduledQuery> scheduledQueryList =
 listScheduledQueriesResult.scheduledQueries();

            printScheduledQuery(scheduledQueryList);
            nextToken = listScheduledQueriesResult.nextToken();
        } while (nextToken != null);
    }
    catch (Exception e) {
        System.out.println("List Scheduled Query failed: " + e);
        throw e;
    }
}

public void printScheduledQuery(List<ScheduledQuery> scheduledQueryList) {
    for (ScheduledQuery scheduledQuery: scheduledQueryList) {
        System.out.println(scheduledQuery.arn());
    }
}
```

Go

```
func (timestreamBuilder TimestreamBuilder) ListScheduledQueries()
 ([]*timestreamquery.ScheduledQuery, error) {

     var nextToken *string = nil
```

```
    var scheduledQueries []*timestreamquery.ScheduledQuery
    for ok := true; ok; ok = nextToken != nil {
        listScheduledQueriesInput := &timestreamquery.ListScheduledQueriesInput{
            MaxResults: aws.Int64(15),
        }
        if nextToken != nil {
            listScheduledQueriesInput.NextToken = aws.String(*nextToken)
        }

        listScheduledQueriesOutput, err :=
 timestreamBuilder.QuerySvc.ListScheduledQueries(listScheduledQueriesInput)
        if err != nil {
            fmt.Printf("Error: %s", err.Error())
            return nil, err
        }
        scheduledQueries = append(scheduledQueries,
 listScheduledQueriesOutput.ScheduledQueries...)
        nextToken = listScheduledQueriesOutput.NextToken
    }
    return scheduledQueries, nil
}
```

Python

```
def list_scheduled_queries(self):
    print("\nListing Scheduled Queries")
    try:
        response = self.query_client.list_scheduled_queries(MaxResults=10)
        self.print_scheduled_queries(response['ScheduledQueries'])
        next_token = response.get('NextToken', None)
        while next_token:
            response =
 self.query_client.list_scheduled_queries(NextToken=next_token, MaxResults=10)
            self.print_scheduled_queries(response['ScheduledQueries'])
            next_token = response.get('NextToken', None)
    except Exception as err:
        print("List scheduled queries failed:", err)
        raise err

@staticmethod
def print_scheduled_queries(scheduled_queries):
    for scheduled_query in scheduled_queries:
        print(scheduled_query['Arn'])
```

Node.js

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
async function listScheduledQueries() {
     console.log("Listing Scheduled Query");
     try {
         var nextToken = null;
         do {
             var params = {
                 MaxResults: 10,
                 NextToken: nextToken
             }
             var data = await queryClient.listScheduledQueries(params).promise();
             var scheduledQueryList = data.ScheduledQueries;
             printScheduledQuery(scheduledQueryList);
             nextToken = data.NextToken;
         }
         while (nextToken != null);
     }  catch (err) {
         console.log("List Scheduled Query failed: ", err);
         throw err;
     }
 }

 async function printScheduledQuery(scheduledQueryList) {
     scheduledQueryList.forEach(element => console.log(element.Arn));
 }
```

.NET

```
private async Task ListScheduledQueries()
 {
     try
     {
         Console.WriteLine("Listing Scheduled Query");
         string nextToken;
         do
         {
             ListScheduledQueriesResponse response =
```

```
                await _amazonTimestreamQuery.ListScheduledQueriesAsync(new
 ListScheduledQueriesRequest());
            foreach (var scheduledQuery in response.ScheduledQueries)
            {
                Console.WriteLine($"{scheduledQuery.Arn}");
            }

            nextToken = response.NextToken;
        } while (nextToken != null);
    }
    catch (Exception e)
    {
        Console.WriteLine($"List Scheduled Query failed: {e}");
        throw;
    }
 }
```

スケジュールされたクエリを記述する

次のコードスニペットを使用して、スケジュールされたクエリを記述できます。

Java

```
public void describeScheduledQueries(String scheduledQueryArn) {
    System.out.println("Describing Scheduled Query");
    try {
        DescribeScheduledQueryResult describeScheduledQueryResult =
 queryClient.describeScheduledQuery(new
 DescribeScheduledQueryRequest().withScheduledQueryArn(scheduledQueryArn));
        System.out.println(describeScheduledQueryResult);
    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
    catch (Exception e) {
        System.out.println("Describe Scheduled Query failed: " + e);
        throw e;
    }
}
```

Java v2

```
public void describeScheduledQueries(String scheduledQueryArn) {
    System.out.println("Describing Scheduled Query");
    try {
        DescribeScheduledQueryResponse describeScheduledQueryResult =

 queryClient.describeScheduledQuery(DescribeScheduledQueryRequest.builder()
                        .scheduledQueryArn(scheduledQueryArn)
                        .build());
        System.out.println(describeScheduledQueryResult);
    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
    catch (Exception e) {
        System.out.println("Describe Scheduled Query failed: " + e);
        throw e;
    }
}
```

Go

```
func (timestreamBuilder TimestreamBuilder) DescribeScheduledQuery(scheduledQueryArn
 string) error {

    describeScheduledQueryInput := &timestreamquery.DescribeScheduledQueryInput{
        ScheduledQueryArn: aws.String(scheduledQueryArn),
    }
    describeScheduledQueryOutput, err :=
 timestreamBuilder.QuerySvc.DescribeScheduledQuery(describeScheduledQueryInput)

    if err != nil {
        if aerr, ok := err.(awserr.Error); ok {
            switch aerr.Code() {
            case timestreamquery.ErrCodeResourceNotFoundException:
                fmt.Println(timestreamquery.ErrCodeResourceNotFoundException,
 aerr.Error())
            default:
                fmt.Printf("Error: %s", err.Error())
            }
        } else {
```

```
            fmt.Printf("Error: %s", aerr.Error())
        }
        return err
    } else {
        fmt.Println("DescribeScheduledQuery is successful, below is the output:")
        fmt.Println(describeScheduledQueryOutput.ScheduledQuery)
        return nil
    }
}
```

Python

```
def describe_scheduled_query(self, scheduled_query_arn):
    print("\nDescribing Scheduled Query")
    try:
        response =
 self.query_client.describe_scheduled_query(ScheduledQueryArn=scheduled_query_arn)
        if 'ScheduledQuery' in response:
            response = response['ScheduledQuery']
            for key in response:
                print("{} :{}".format(key, response[key]))
    except self.query_client.exceptions.ResourceNotFoundException as err:
        print("Scheduled Query doesn't exist")
        raise err
    except Exception as err:
        print("Scheduled Query describe failed:", err)
        raise err
```

Node.js

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
async function describeScheduledQuery(scheduledQueryArn) {
    console.log("Describing Scheduled Query");
    var params = {
        ScheduledQueryArn: scheduledQueryArn
    }
    try {
        const data = await queryClient.describeScheduledQuery(params).promise();
        console.log(data.ScheduledQuery);
    } catch (err) {
```

```
        console.log("Describe Scheduled Query failed: ", err);
        throw err;
    }
}
```

.NET

```
private async Task DescribeScheduledQuery(string scheduledQueryArn)
{
    try
    {
        Console.WriteLine("Describing Scheduled Query");
        DescribeScheduledQueryResponse response = await
_amazonTimestreamQuery.DescribeScheduledQueryAsync(
            new DescribeScheduledQueryRequest()
            {
                ScheduledQueryArn = scheduledQueryArn
            });

Console.WriteLine($"{JsonConvert.SerializeObject(response.ScheduledQuery)}");
    }
    catch (ResourceNotFoundException e)
    {
        Console.WriteLine($"Scheduled Query doesn't exist: {e}");
        throw;
    }
    catch (Exception e)
    {
        Console.WriteLine($"Describe Scheduled Query failed: {e}");
        throw;
    }
}
```

スケジュールされたクエリを実行する

次のコードスニペットを使用して、スケジュールされたクエリを実行できます。

Java

```
public void executeScheduledQueries(String scheduledQueryArn, Date invocationTime) {
    System.out.println("Executing Scheduled Query");
    try {
```

```
        ExecuteScheduledQueryResult executeScheduledQueryResult =
 queryClient.executeScheduledQuery(new ExecuteScheduledQueryRequest()
                .withScheduledQueryArn(scheduledQueryArn)
                .withInvocationTime(invocationTime)
        );

    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
    catch (Exception e) {
        System.out.println("Execution Scheduled Query failed: " + e);
        throw e;
    }
}
```

Java v2

```
public void executeScheduledQuery(String scheduledQueryArn) {
    System.out.println("Executing Scheduled Query");
    try {
        ExecuteScheduledQueryResponse executeScheduledQueryResult =
 queryClient.executeScheduledQuery(ExecuteScheduledQueryRequest.builder()
                .scheduledQueryArn(scheduledQueryArn)
                .invocationTime(Instant.now())
                .build()
        );

        System.out.println("Execute ScheduledQuery response code: " +
 executeScheduledQueryResult.sdkHttpResponse().statusCode());

    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
    catch (Exception e) {
        System.out.println("Execution Scheduled Query failed: " + e);
        throw e;
    }
}
```

Go

```
func (timestreamBuilder TimestreamBuilder) ExecuteScheduledQuery(scheduledQueryArn
 string, invocationTime time.Time) error {

    executeScheduledQueryInput := &timestreamquery.ExecuteScheduledQueryInput{
        ScheduledQueryArn: aws.String(scheduledQueryArn),
        InvocationTime:    aws.Time(invocationTime),
    }
    executeScheduledQueryOutput, err :=
 timestreamBuilder.QuerySvc.ExecuteScheduledQuery(executeScheduledQueryInput)

    if err != nil {
        if aerr, ok := err.(awserr.Error); ok {
            switch aerr.Code() {
            case timestreamquery.ErrCodeResourceNotFoundException:
                fmt.Println(timestreamquery.ErrCodeResourceNotFoundException,
 aerr.Error())
            default:
                fmt.Printf("Error: %s", aerr.Error())
            }
        } else {
            fmt.Printf("Error: %s", err.Error())
        }
        return err
    } else {
        fmt.Println("ExecuteScheduledQuery is successful, below is the output:")
        fmt.Println(executeScheduledQueryOutput.GoString())
        return nil
    }
}
```

Python

```
def execute_scheduled_query(self, scheduled_query_arn, invocation_time):
    print("\nExecuting Scheduled Query")
    try:

 self.query_client.execute_scheduled_query(ScheduledQueryArn=scheduled_query_arn,
 InvocationTime=invocation_time)
        print("Successfully started executing scheduled query")
    except self.query_client.exceptions.ResourceNotFoundException as err:
        print("Scheduled Query doesn't exist")
```

```
        raise err
    except Exception as err:
        print("Scheduled Query execution failed:", err)
        raise err
```

Node.js

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
async function executeScheduledQuery(scheduledQueryArn, invocationTime) {
     console.log("Executing Scheduled Query");
     var params = {
         ScheduledQueryArn: scheduledQueryArn,
         InvocationTime: invocationTime
     }
     try {
         await queryClient.executeScheduledQuery(params).promise();
     } catch (err) {
         console.log("Execute Scheduled Query failed: ", err);
         throw err;
     }
 }
```

.NET

```
private async Task ExecuteScheduledQuery(string scheduledQueryArn, DateTime
 invocationTime)
 {
     try
     {
         Console.WriteLine("Running Scheduled Query");
         await _amazonTimestreamQuery.ExecuteScheduledQueryAsync(new
 ExecuteScheduledQueryRequest()
         {
             ScheduledQueryArn = scheduledQueryArn,
             InvocationTime = invocationTime
         });
         Console.WriteLine("Successfully started manual run of scheduled query");
     }
     catch (ResourceNotFoundException e)
     {
```

```
            Console.WriteLine($"Scheduled Query doesn't exist: {e}");
            throw;
        }
        catch (Exception e)
        {
            Console.WriteLine($"Execute Scheduled Query failed: {e}");
            throw;
        }
    }
```

スケジュールされたクエリを更新する

次のコードスニペットを使用して、スケジュールされたクエリを更新できます。

Java

```
public void updateScheduledQueries(String scheduledQueryArn) {
    System.out.println("Updating Scheduled Query");
    try {
        queryClient.updateScheduledQuery(new UpdateScheduledQueryRequest()
                .withScheduledQueryArn(scheduledQueryArn)
                .withState(ScheduledQueryState.DISABLED));
        System.out.println("Successfully update scheduled query state");
    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
    catch (Exception e) {
        System.out.println("Execution Scheduled Query failed: " + e);
        throw e;
    }
}
```

Java v2

```
public void updateScheduledQuery(String scheduledQueryArn, ScheduledQueryState
 state) {
    System.out.println("Updating Scheduled Query");
    try {
        queryClient.updateScheduledQuery(UpdateScheduledQueryRequest.builder()
                .scheduledQueryArn(scheduledQueryArn)
```

```
                .state(state)
                .build());
        System.out.println("Successfully update scheduled query state");
    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
    catch (Exception e) {
        System.out.println("Execution Scheduled Query failed: " + e);
        throw e;
    }
}
```

Go

```
func (timestreamBuilder TimestreamBuilder) UpdateScheduledQuery(scheduledQueryArn
 string) error {

    updateScheduledQueryInput := &timestreamquery.UpdateScheduledQueryInput{
        ScheduledQueryArn: aws.String(scheduledQueryArn),
        State:             aws.String(timestreamquery.ScheduledQueryStateDisabled),
    }
    _, err :=
 timestreamBuilder.QuerySvc.UpdateScheduledQuery(updateScheduledQueryInput)

    if err != nil {
        if aerr, ok := err.(awserr.Error); ok {
            switch aerr.Code() {
            case timestreamquery.ErrCodeResourceNotFoundException:
                fmt.Println(timestreamquery.ErrCodeResourceNotFoundException,
 aerr.Error())
            default:
                fmt.Printf("Error: %s", aerr.Error())
            }
        } else {
            fmt.Printf("Error: %s", err.Error())
        }
        return err
    } else {
        fmt.Println("UpdateScheduledQuery is successful")
        return nil
    }
```

```
 }
```

Python

```
def update_scheduled_query(self, scheduled_query_arn, state):
    print("\nUpdating Scheduled Query")
    try:

 self.query_client.update_scheduled_query(ScheduledQueryArn=scheduled_query_arn,
                                                    State=state)
        print("Successfully update scheduled query state to", state)
    except self.query_client.exceptions.ResourceNotFoundException as err:
        print("Scheduled Query doesn't exist")
        raise err
    except Exception as err:
        print("Scheduled Query deletion failed:", err)
        raise err
```

Node.js

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
async function updateScheduledQueries(scheduledQueryArn) {
     console.log("Updating Scheduled Query");
     var params = {
         ScheduledQueryArn: scheduledQueryArn,
         State: "DISABLED"
     }
     try {
         await queryClient.updateScheduledQuery(params).promise();
         console.log("Successfully update scheduled query state");
     } catch (err) {
         console.log("Update Scheduled Query failed: ", err);
         throw err;
     }
 }
```

.NET

```
private async Task UpdateScheduledQuery(string scheduledQueryArn,
 ScheduledQueryState state)
```

```
{
    try
    {
        Console.WriteLine("Updating Scheduled Query");
        await _amazonTimestreamQuery.UpdateScheduledQueryAsync(new
UpdateScheduledQueryRequest()
        {
            ScheduledQueryArn = scheduledQueryArn,
            State = state
        });
        Console.WriteLine("Successfully update scheduled query state");
    }
    catch (ResourceNotFoundException e)
    {
        Console.WriteLine($"Scheduled Query doesn't exist: {e}");
        throw;
    }
    catch (Exception e)
    {
        Console.WriteLine($"Update Scheduled Query failed: {e}");
        throw;
    }
}
```

スケジュールされたクエリを削除する

次のコードスニペットを使用して、スケジュールされたクエリを削除できます。

Java

```
public void deleteScheduledQuery(String scheduledQueryArn) {
    System.out.println("Deleting Scheduled Query");

    try {
        queryClient.deleteScheduledQuery(new
DeleteScheduledQueryRequest().withScheduledQueryArn(scheduledQueryArn));
        System.out.println("Successfully deleted scheduled query");
    }
    catch (Exception e) {
        System.out.println("Scheduled Query deletion failed: " + e);
    }
}
```

Java v2

```
public void deleteScheduledQuery(String scheduledQueryArn) {
    System.out.println("Deleting Scheduled Query");

    try {
        queryClient.deleteScheduledQuery(DeleteScheduledQueryRequest.builder()
                .scheduledQueryArn(scheduledQueryArn).build());
        System.out.println("Successfully deleted scheduled query");
    }
    catch (Exception e) {
        System.out.println("Scheduled Query deletion failed: " + e);
    }
}
```

Go

```
func (timestreamBuilder TimestreamBuilder) DeleteScheduledQuery(scheduledQueryArn
 string) error {

     deleteScheduledQueryInput := &timestreamquery.DeleteScheduledQueryInput{
         ScheduledQueryArn: aws.String(scheduledQueryArn),
     }
     _, err :=
 timestreamBuilder.QuerySvc.DeleteScheduledQuery(deleteScheduledQueryInput)

     if err != nil {
         fmt.Println("Error:")
         if aerr, ok := err.(awserr.Error); ok {
             switch aerr.Code() {
             case timestreamquery.ErrCodeResourceNotFoundException:
                 fmt.Println(timestreamquery.ErrCodeResourceNotFoundException,
 aerr.Error())
             default:
                 fmt.Printf("Error: %s", aerr.Error())
             }
         } else {
             fmt.Printf("Error: %s", err.Error())
         }
         return err
     } else {
         fmt.Println("DeleteScheduledQuery is successful")
         return nil
```

```
    }
}
```

Python

```
def delete_scheduled_query(self, scheduled_query_arn):
    print("\nDeleting Scheduled Query")
    try:

 self.query_client.delete_scheduled_query(ScheduledQueryArn=scheduled_query_arn)
        print("Successfully deleted scheduled query :", scheduled_query_arn)
    except Exception as err:
        print("Scheduled Query deletion failed:", err)
        raise err
```

Node.js

次のスニペットは JavaScript V2 スタイルに AWS SDKを使用します。これは、Node.js のサンプルアプリケーションに基づいており、 の LiveAnalytics アプリケーション用の Amazon Timestream サンプルです GitHub。

```
async function deleteScheduleQuery(scheduledQueryArn) {
     console.log("Deleting Scheduled Query");
     const params = {
         ScheduledQueryArn: scheduledQueryArn
     }
     try {
         await queryClient.deleteScheduledQuery(params).promise();
         console.log("Successfully deleted scheduled query");
     } catch (err) {
         console.log("Scheduled Query deletion failed: ", err);
     }
}
```

.NET

```
private async Task DeleteScheduledQuery(string scheduledQueryArn)
 {
     try
     {
         Console.WriteLine("Deleting Scheduled Query");
         await _amazonTimestreamQuery.DeleteScheduledQueryAsync(new
 DeleteScheduledQueryRequest()
```

```
        {
            ScheduledQueryArn = scheduledQueryArn
        });
        Console.WriteLine($"Successfully deleted scheduled query :
{scheduledQueryArn}");
    }
    catch (Exception e)
    {
        Console.WriteLine($"Scheduled Query deletion failed: {e}");
        throw;
    }
}
```

の Timestream でのバッチロードの使用 LiveAnalytics

の Amazon Timestream のバッチロードを使用すると LiveAnalytics、Amazon S3 に保存されているCSVファイルをバッチで Timestream に取り込むことができます。 Amazon S3 この新しい機能により、他のツールに頼ったり、カスタムコードを記述 LiveAnalytics したりすることなく、 の Timestream にデータを取得できます。バッチロードを使用して、クエリや分析にすぐに必要ではないデータなど、柔軟な待機時間でデータをバックフィルできます。

バッチロードタスクは、 AWS Management Console、、 AWS CLIおよび を使用して作成できます AWS SDKs。詳細については、コンソールでのバッチロードの使用、でのバッチロードの使用 AWS CLI、およびでのバッチロードの使用 AWS SDKsを参照してください。

バッチロードに加えて、 WriteRecords API オペレーションと同時に複数のレコードを書き込むことができます。使用する方法のガイダンスについては、「」を参照してくださいオペレーションとバッチロードの選択 WriteRecords API 。

トピック

- Timestream でのバッチロードの概念
- バッチロードの前提条件
- バッチロードのベストプラクティス
- バッチロードデータファイルの準備
- バッチロードのデータモデルマッピング
- コンソールでのバッチロードの使用
- でのバッチロードの使用 AWS CLI

- でのバッチロードの使用 AWS SDKs
- バッチロードエラーレポートの使用

Timestream でのバッチロードの概念

バッチロードの機能をよりよく理解するには、次の概念を確認してください。

バッチロードタスク – Amazon Timestream でソースデータと送信先を定義するタスク。バッチロードタスクを作成するときに、データモデルなどの追加の設定を指定します。バッチロードタスクは、AWS Management Console、、 AWS CLIおよび を通じて作成できます AWS SDKs。

インポート先 – Timestream のインポート先データベースとテーブル。データベースとテーブルの作成については、 データベースを作成する「」および「」を参照してくださいテーブルを作成する。

データソース – S3 バケットに保存されているソースCSVファイル。データファイルの準備については、「」を参照してくださいバッチロードデータファイルの準備。S3 料金の詳細については、Amazon S3料金」を参照してください。

バッチロードエラーレポート – バッチロードタスクのエラーに関する情報を保存するレポート。バッチロードタスクの一部として、バッチロードエラーレポートの S3 の場所を定義します。レポートの情報については、「」を参照してくださいバッチロードエラーレポートの使用。

データモデルマッピング – S3 ロケーションのデータソースから LiveAnalytics テーブルのターゲット Timestream への時間、ディメンション、および測定値のバッチロードマッピング。詳細については、「バッチロードのデータモデルマッピング」を参照してください。

バッチロードの前提条件

これは、バッチロードを使用するための前提条件のリストです。ベストプラクティスについては、バッチロードのベストプラクティスを参照してください。

- バッチロードソースデータは、ヘッダー付きのCSV形式で Amazon S3 に保存されます。
- Amazon S3 ソースバケットごとに、アタッチされたポリシーで次のアクセス許可が必要です。

```
"s3:GetObject",
"s3:GetBucketAcl"
"s3:ListBucket"
```

同様に、レポートが書き込まれる Amazon S3 出力バケットごとに、アタッチされたポリシーに次のアクセス許可が必要です。

```
"s3:PutObject",
"s3:GetBucketAcl"
```

例:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:GetObject",
        "s3:GetBucketAcl"
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::inputs-source-bucket-name-A”
        "arn:aws:s3:::inputs-source-bucket-name-B”
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "s3:PutObject",
        "s3:GetBucketAcl"
      ],
      "Resource": [
        "arn:aws:s3:::reports-output-bucket-name”
      ]
      "Effect": "Allow"
    }
  ]
}
```

- の Timestream は、データモデルで提供された情報をCSVヘッダーにマッピングCSVすることで、を LiveAnalytics 解析します。データには、タイムスタンプを表す列、少なくとも 1 つのディメンション列、および少なくとも 1 つのメジャー列が必要です。
- バッチロードで使用される S3 バケットは、バッチロードで使用される LiveAnalytics テーブルの Timestream と同じリージョンにあり、同じアカウントにある必要があります。

- `timestamp` 列は、Unix エポックからの時間を表す長いデータ型である必要があります。例えば、タイムスタンプは として表`2021-03-25T08:45:21Z`されます`1616661921`。Timestream は、タイムスタンプの精度に対して秒、ミリ秒、マイクロ秒、ナノ秒をサポートします。クエリ言語を使用する場合、 などの関数を使用して形式を変換できます`to_unixtime`。詳細については、「日付/時刻関数」を参照してください。
- Timestream は、ディメンション値の文字列データ型をサポートしています。測定列のロング、ダブル、文字列、ブールデータ型をサポートします。

バッチロードの制限とクォータについては、「」を参照してくださいバッチロード。

バッチロードのベストプラクティス

バッチロードは、次の条件と推奨事項に従う場合に最適です (高スループット) 。

1. CSV 取り込み用に送信されたファイルは小さく、特にファイルサイズは 100MB ~ 1GB で、並列処理と取り込み速度が向上します。
2. バッチロードが進行中の場合、同じテーブルに同時にデータを取り込み (WriteRecords APIオペレーションやスケジュールされたクエリの使用など) ないようにしてください。これによりスロットルが発生する可能性があり、バッチロードタスクは失敗します。
3. バッチロードタスクの実行中は、バッチロードで使用される S3 バケットからファイルを追加、変更、または削除しないでください。
4. テーブルやソースからアクセス許可を削除または取り消したり、バッチロードタスクがスケジュールされているか進行中の S3 バケットを報告したりしないでください。
5. カーディナリティの高いディメンション値のセットを持つデータを取り込む場合は、「」のガイダンスに従ってくださいマルチメジャーレコードをパーティション分割するための推奨事項。
6. 小さなファイルを送信して、データの正確性をテストしてください。バッチロードに送信されたデータには、正確性に関係なく課金されます。料金の詳細については、「Amazon Timestream の料金」を参照してください。
7. `ActiveMagneticStorePartitions` が 250 未満でない限り、バッチロードタスクを再開しないでください。ジョブがスロットリングされて失敗する可能性があります。同じデータベースに対して複数のジョブを同時に送信すると、数が減ります。

コンソールのベストプラクティスを次に示します。

1. マルチメジャーレコードに 1 つのメジャー名のみを使用するシンプルなデータモデリングにのみビルダーを使用します。

2. より複雑なデータモデリングには、 を使用しますJSON。例えば、マルチメジャーレコードJSON を使用するときに複数のメジャー名を使用する場合、 を使用します。

LiveAnalytics ベストプラクティスの詳細については、「」を参照してくださいベストプラクティス。

バッチロードデータファイルの準備

ソースデータファイルには区切り文字で区切られた値があります。より具体的な用語であるカンマ区切り値 (CSV) は、一般的に使用されます。有効な列区切り文字には、カンマとパイプが含まれます。レコードは新しい行で区切られます。ファイルは Amazon S3 に保存する必要があります。新しいバッチロードタスクを作成すると、ソースデータの場所は ファイルの によって指定ARNされます。ファイルにはヘッダーが含まれます。1 つの列はタイムスタンプを表します。少なくとも 1 つの他の列はメジャーを表します。

バッチロードで使用される S3 バケットは、バッチロードで使用される LiveAnalytics テーブルの Timestream と同じリージョンにある必要があります。バッチロードタスクの送信後に、バッチロードで使用される S3 バケットからファイルを追加または削除しないでください。S3 バケットの操作については、Amazon S3 の開始方法」を参照してください。

> **Note**
>
> CSV Excel などの一部のアプリケーションによって生成されるファイルには、予想されるエンコーディングと競合するバイト順序マーク (BOM) が含まれている場合があります。プログラムで処理されたときにエラーがBOMスローされるCSVファイルを参照する LiveAnalytics バッチロードタスクのタイムストリーム。これを回避するには、非表示の文字BOMである を削除できます。
> 例えば、新しいエンコーディングを指定できる Notepad++ などのアプリケーションからファイルを保存できます。1 行目を読み取って行から文字を削除し、ファイル内の 1 行目に新しい値を書き込むプログラムオプションを使用することもできます。
> Excel から保存する場合、複数のCSVオプションがあります。別のCSVオプションで を保存すると、記載されている問題が防止される可能性があります。ただし、エンコーディングの変更が一部の文字に影響する可能性があるため、結果を確認する必要があります。

CSV フォーマットパラメータ

エスケープ文字は、形式パラメータによって予約されている値を表す場合に使用します。例えば、引用符文字が二重引用符の場合、データ内の二重引用符を表すには、二重引用符の前にエスケープ文字を配置します。

バッチロードタスクの作成時にこれらを指定するタイミングについては、「」を参照してくださいバッチロードタスクを作成する。

パラメータ	オプション
列区切り文字	(カンマ (',') | パイプ ('|') | セミコロン (';') | タブ ('/t') | 空白 (' '))
エスケープ文字	なし
引用符文字	コンソール: (二重引用符 (「) | 単一引用符 ('))
Null 値	空白 (' ')
トリミング空白	コンソール: (いいえ | はい)

バッチロードのデータモデルマッピング

次に、データモデルマッピングのスキーマと、 および の例を示します。

データモデルマッピングスキーマ

バッチロードDataModel用の を含むBatchLoadTaskDescriptionオブジェクトDescribeBatchLoadTaskを含めるための呼び出しによって返されるCreateBatchLoadTaskリクエスト構文とDataModelConfigurationオブジェクト。は、S3 ロケーションに CSV 形式で保存されたソースデータから LiveAnalytics 、データベースとテーブルのターゲット Timestream へのマッピングDataModelを定義します。

TimeColumn フィールドは、 の Timestream の宛先テーブルのtime列にマッピングされる値のソースデータの場所を示します LiveAnalytics。は の単位TimeUnitを指定しTimeColumn、MILLISECONDS、、SECONDSMICROSECONDS、または のいずれかを指定できますNANOSECONDS。ディメンションとメジャーのマッピングもあります。属性マッピングは、ソース列とターゲットフィールドで構成されます。

詳細については、「」を参照してくださいDimensionMapping。メジャーのマッピングには、MixedMeasureMappingsの 2 つのオプションがありますMultiMeasureMappings。

要約すると、 には、S3 ロケーションのデータソースから、次の LiveAnalytics テーブルのターゲット Timestream へのマッピングDataModelが含まれています。

- 時間
- ディメンション
- メジャー

可能であれば、 の Timestream のマルチメジャーレコードにメジャーデータをマッピングすることをお勧めします LiveAnalytics。マルチメジャーレコードの利点については、「」を参照してくださいマルチメジャーレコード。

ソースデータ内の複数のメジャーが 1 行に保存されている場合は、これらの複数のメジャーを Timestream のマルチメジャーレコードにマッピングして、 LiveAnalytics を使用できますMultiMeasureMappings。単一メジャーレコードにマッピングする必要がある値がある場合は、を使用できますMixedMeasureMappings。

MixedMeasureMappings と MultiMeasureMappings の両方に が含まれますMultiMeasureAttributeMappings。マルチメジャーレコードは、単一メジャーレコードが必要かどうかに関係なくサポートされます。

の Timestream でマルチメジャーターゲットレコードのみが必要な場合は LiveAnalytics、次の構造でメジャーマッピングを定義できます。

```
CreateBatchLoadTask
    MeasureNameColumn
    MultiMeasureMappings
        TargetMultiMeasureName
        MultiMeasureAttributeMappings array
```

> **Note**
>
> MultiMeasureMappings 可能な限り を使用することをお勧めします。

の Timestream で単一測定ターゲットレコードが必要な場合は LiveAnalytics、次の構造で測定マッピングを定義できます。

```
CreateBatchLoadTask
    MeasureNameColumn
    MixedMeasureMappings array
        MixedMeasureMapping
            MeasureName
            MeasureValueType
            SourceColumn
            TargetMeasureName
            MultiMeasureAttributeMappings array
```

を使用する場合MultiMeasureMappings、MultiMeasureAttributeMappings配列は常に必要です。MixedMeasureMappings 配列を使用する場合、 MeasureValueTypeが特定のMULTI用である場合MixedMeasureMapping、その には MultiMeasureAttributeMappingsが必要ですMixedMeasureMapping。それ以外の場合は、単一メジャーレコードのメジャータイプMeasureValueTypeを示します。

いずれにしても、MultiMeasureAttributeMapping利用可能な配列があります。各 でマルチメジャーレコードへのマッピングをMultiMeasureAttributeMapping次のように定義します。

SourceColumn

Amazon S3 にあるソースデータの列。

TargetMultiMeasureAttributeName

宛先テーブル内のターゲットマルチメジャー名の名前。この入力は、 MeasureNameColumn が指定されていない場合に必要です。MeasureNameColumn を指定すると、その列の値がマルチメジャー名として使用されます。

MeasureValueType

DOUBLE、BIGINTBOOLEAN、VARCHAR、または のいずれかTIMESTAMP。

MultiMeasureMappings 例を使用したデータモデルマッピング

この例では、各測定値を専用列に保存する優先アプローチであるマルチメジャーレコードへのマッピングを示しています。サンプルは、サンプル CSVCSVでダウンロードできます。サンプルには、LiveAnalytics テーブルの Timestream のターゲット列にマッピングする次の見出しがあります。

- time
- measure_name

- region
- location
- hostname
- memory_utilization
- cpu_utilization

CSV ファイル内の 列timeと measure_name列を特定します。この場合、これらは同じ名前の LiveAnalytics テーブル列の Timestream に直接マッピングされます。

- time マップ先 time
- measure_name マップ先 measure_name (または選択した値)

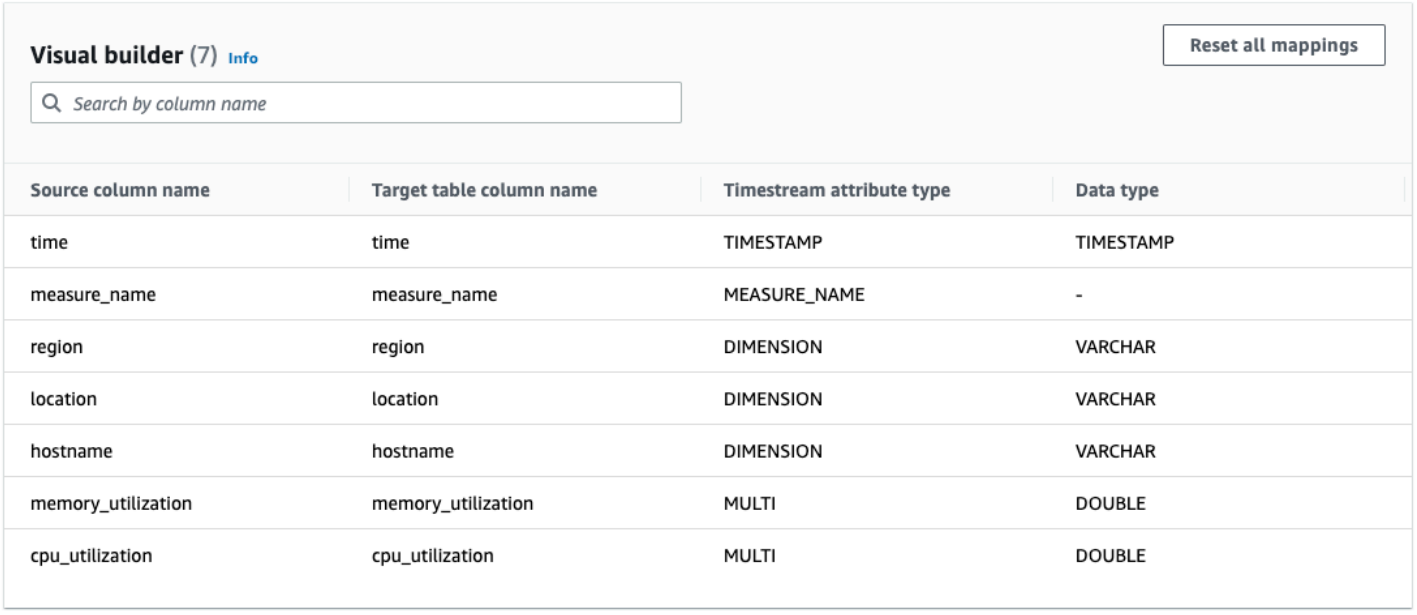
を使用する場合API、 TimeColumnフィールドtimeで を指定し、 TimeUnitフィールドMILLISECONDSで などのサポートされている時間単位値を指定します。これらは、コンソールのソース列名とタイムスタンプ時間入力に対応します。MeasureNameColumn キーでmeasure_name定義されている を使用して、レコードをグループ化またはパーティション化できます。

サンプルでは、region、location、 hostnameはディメンションです。ディメンションはDimensionMappingオブジェクトの配列にマッピングされます。

メジャーの場合、TargetMultiMeasureAttributeName値は LiveAnalytics テーブルの Timestream の列になります。この例では、 のように同じ名前を保持できます。または、新しい を指定できます。 MeasureValueTypeは、DOUBLE、、BIGINT、BOOLEANVARCHAR、または のいずれかですTIMESTAMP。

```
{
  "TimeColumn": "time",
  "TimeUnit": "MILLISECONDS",
  "DimensionMappings": [
    {
      "SourceColumn": "region",
      "DestinationColumn": "region"
    },
    {
      "SourceColumn": "location",
      "DestinationColumn": "location"
    },
```

```
    {
      "SourceColumn": "hostname",
      "DestinationColumn": "hostname"
    }
  ],
  "MeasureNameColumn": "measure_name",
  "MultiMeasureMappings": {
    "MultiMeasureAttributeMappings": [
      {
        "SourceColumn": "memory_utilization",
        "TargetMultiMeasureAttributeName": "memory_utilization",
        "MeasureValueType": "DOUBLE"
      },
      {
        "SourceColumn": "cpu_utilization",
        "TargetMultiMeasureAttributeName": "cpu_utilization",
        "MeasureValueType": "DOUBLE"
      }
    ]
  }
}
```



MixedMeasureMappings 例を使用したデータモデルマッピング

このアプローチは、 の Timestream の単一測定レコードにマッピングする必要がある場合にのみ使用することをお勧めします LiveAnalytics。

コンソールでのバッチロードの使用

でバッチロードを使用する手順は次のとおりです AWS Management Console。サンプルは、サンプル CSVCSVでダウンロードできます。

トピック

- バッチロードにアクセスする
- バッチロードタスクを作成する
- バッチロードタスクを再開する
- Visual Builder の使用

バッチロードにアクセスする

を使用してバッチロードにアクセスするには、次の手順に従います AWS Management Console。

1. Amazon Timestream コンソール を開きます。
2. ナビゲーションペインで、管理ツール を選択し、バッチロードタスク を選択します。
3. ここから、バッチロードタスクのリストを表示し、特定のタスクにドリルダウンして詳細を確認できます。タスクを作成および再開することもできます。

バッチロードタスクを作成する

を使用してバッチロードタスクを作成するには、次の手順に従います AWS Management Console。

1. Amazon Timestream コンソール を開きます。
2. ナビゲーションペインで、管理ツール を選択し、バッチロードタスク を選択します。
3. バッチロードタスクの作成 を選択します。
4. インポート先 で、以下を選択します。

   - ターゲットデータベース – で作成されたデータベースの名前を選択します データベースを作成する。
   - ターゲットテーブル – で作成されたテーブルの名前を選択しますテーブルを作成する。

   必要に応じて、新しいテーブルの作成ボタンを使用して、このパネルからテーブルを追加できます。

5. データソース のデータソース S3 の場所から、ソースデータが保存されている S3 バケットを選択します。 Browse S3 ボタンを使用して、アクティブなAWSアカウントがアクセスできる S3 リソースを表示するか、S3 の場所 を入力しますURL。データソースは同じリージョンに配置する必要があります。
6. ファイル形式設定 (拡張可能なセクション) では、デフォルト設定を使用して入力データを解析できます。詳細設定 を選択することもできます。そこからCSVフォーマットパラメータ を選択し、パラメータを選択して入力データを解析できます。これらのパラメータの詳細については、「」を参照してくださいCSV フォーマットパラメータ。
7. データモデルマッピングの設定 から、データモデルを設定します。データモデルに関するその他のガイダンスについては、「」を参照してください。 バッチロードのデータモデルマッピング

   - データモデルマッピング から、マッピング設定入力 を選択し、次のいずれかを選択します。
     - Visual Builder – データを視覚的にマッピングするには、 TargetMultiMeasureNameまたはを選択しますMeasureNameColumn。次に、Visual Builder から列をマッピングします。

       Visual Builder は、単一のファイルがデータソースとして選択されたときに、データソースCSVファイルからソース列ヘッダーを自動的に検出してロードします。属性とデータ型を選択してマッピングを作成します。

       Visual Builder の使用の詳細については、「」を参照してくださいVisual Builder の使用。
     - JSON エディタ – データモデルを設定するためのフリーフォームJSONエディタ。の Timestream に精通し LiveAnalytics ていて、高度なデータモデルマッピングを構築する場合は、このオプションを選択します。
     - JSON S3 からのファイル – S3 に保存したJSONモデルファイルを選択します。データモデルを既に設定していて、追加のバッチロードに再利用する場合は、このオプションを選択します。
8. エラーログレポート のエラーログ S3 の場所から、エラーの報告に使用される S3 の場所を選択します。 このレポートの使用方法については、「」を参照してくださいバッチロードエラーレポートの使用。
9. 暗号化キータイプ では、次のいずれかを選択します。

   - Amazon S3-managedキー (SSE-S3) — Amazon S3 が作成、管理、使用する暗号化キー。
   - AWS KMS key （ SSE-KMS ） – AWS Key Management Service () で保護されている暗号化キーAWS KMS。
10. [Next (次へ)] を選択します。

11. レビューと作成ページで、設定を確認し、必要に応じて編集します。

> **Note**
>
> タスクの作成後にバッチロードタスク設定を変更することはできません。タスクの完了時間は、インポートされるデータの量によって異なります。

12. バッチロードタスクの作成 を選択します。

バッチロードタスクを再開する

ステータスが「進行停止」で、再開可能なバッチロードタスクを選択すると、タスクを再開するように求められます。これらのタスクの詳細を表示するときに、タスクを再開ボタンが付いたバナーもあります。再開可能なタスクには「再開基準」の日付があります。その日付が過ぎると、タスクを再開することはできません。

Visual Builder の使用

Visual Builder を使用して、S3 バケットに保存されている 1 つ以上のCSVファイル列を、LiveAnalytics テーブルの Timestream の送信先列にマッピングできます。

> **Note**
>
> ロールには、 ファイルの`SelectObjectContent`アクセス許可が必要です。これを行わない場合、列を手動で追加および削除する必要があります。

ソース列の自動ロードモード

の Timestream LiveAnalytics は、バケットを 1 つだけ指定すると、ソースCSVファイルで列名を自動的にスキャンできます。既存のマッピングがない場合は、ソース列のインポート を選択できます。

1. マッピング設定入力設定 から Visual Builder オプションを選択し、タイムスタンプ時間入力を設定します。 `Milliseconds`がデフォルト設定です。
2. ソースデータファイルにある列ヘッダーをインポートするには、ソース列のロードボタンをクリックします。テーブルには、データソースファイルのソース列ヘッダー名が入力されます。
3. ソース列ごとに、ターゲットテーブルの列名 、Timestream 属性タイプ 、およびデータタイプを選択します。

これらの列と可能な値の詳細については、「」を参照してくださいフィールドのマッピング。

4. drag-to-fill 機能を使用して、複数の列の値を一度に設定します。

ソース列を手動で追加する

バケットまたはCSVプレフィックスを使用していて、単一の を使用していない場合はCSV、列マッピングの追加ボタンと列マッピングの削除ボタンを使用して、ビジュアルエディタから列マッピングを追加および削除できます。マッピングをリセットするボタンもあります。

フィールドのマッピング

- ソース列名 – インポートするメジャーを表すソースファイル内の列の名前。の Timestream LiveAnalytics は、インポートソース列 を使用すると、この値を自動的に入力できます。
- ターゲットテーブルの列名 – ターゲットテーブル内のメジャーの列名を示すオプションの入力。
- Timestream 属性タイプ – など、指定されたソース列内のデータの属性タイプDIMENSION。
  - TIMESTAMP – メジャーが収集された日時を指定します。
  - MULTI – 複数の測定値が表示されます。
  - DIMENSION – 時系列メタデータ。
  - MEASURE_NAME – 単一メジャーレコードの場合、これはメジャー名です。
- データ型 – など、Timestream 列のタイプBOOLEAN。
  - BIGINT – 64 ビット整数。
  - BOOLEAN – ロジックの 2 つの真値 — true と false。
  - DOUBLE – 64 ビット可変精度数。
  - TIMESTAMP – でナノ秒の精度時間を使用しUTC、Unix エポックからの時間を追跡する時間内のインスタンス。

でのバッチロードの使用 AWS CLI

セットアップ

バッチロードの使用を開始するには、次の手順を実行します。

1. の手順 AWS CLI を使用して をインストールします LiveAnalytics を使用するための Amazon Timestream へのアクセス AWS CLI。

2. 次のコマンドを実行して、Timestream CLI コマンドが更新されていることを確認します。がリストにあることを確認します create-batch-load-task。

   ```
   aws timestream-write help
   ```

3. の手順を使用してデータソースを準備しますバッチロードデータファイルの準備。
4. の手順を使用して、データベースとテーブルを作成します LiveAnalytics を使用するための Amazon Timestream へのアクセス AWS CLI。
5. レポート出力用の S3 バケットを作成します。バケットは同じリージョンにある必要があります。バケットの詳細については、Amazon S3バケットの作成、設定、および操作」を参照してください。
6. バッチロードタスクを作成します。この手順については、「バッチロードタスクを作成する」を参照してください。
7. タスクのステータスを確認します。この手順については、「バッチロードタスクの説明」を参照してください。

バッチロードタスクを作成する

create-batch-load-task コマンドを使用してバッチロードタスクを作成できます。を使用してバッチロードタスクを作成する場合CLI、JSONパラメータ を使用できます。これによりcli-input-json、パラメータを 1 JSON つのフラグメントに集約できます。また、、data-model-configuration、、data-source-configuration、 など、他のいくつかのパラメータを使用してreport-configurationtarget-database-name、これらの詳細を分割することもできますtarget-table-name。

例については、「バッチロードタスクの作成例」を参照してください。

バッチロードタスクの説明

バッチロードタスクの説明は、次のように取得できます。

```
aws timestream-write describe-batch-load-task --task-id <value>
```

以下に、応答の例を示します。

```
{
    "BatchLoadTaskDescription": {
        "TaskId": "<TaskId>",
        "DataSourceConfiguration": {
```

```
            "DataSourceS3Configuration": {
                "BucketName": "test-batch-load-west-2",
                "ObjectKeyPrefix": "sample.csv"
            },
            "CsvConfiguration": {},
            "DataFormat": "CSV"
        },
        "ProgressReport": {
            "RecordsProcessed": 2,
            "RecordsIngested": 0,
            "FileParseFailures": 0,
            "RecordIngestionFailures": 2,
            "FileFailures": 0,
            "BytesIngested": 119
        },
        "ReportConfiguration": {
            "ReportS3Configuration": {
                "BucketName": "test-batch-load-west-2",
                "ObjectKeyPrefix": "<ObjectKeyPrefix>",
                "EncryptionOption": "SSE_S3"
            }
        },
        "DataModelConfiguration": {
            "DataModel": {
                "TimeColumn": "timestamp",
                "TimeUnit": "SECONDS",
                "DimensionMappings": [
                    {
                        "SourceColumn": "vehicle",
                        "DestinationColumn": "vehicle"
                    },
                    {
                        "SourceColumn": "registration",
                        "DestinationColumn": "license"
                    }
                ],
                "MultiMeasureMappings": {
                    "TargetMultiMeasureName": "test",
                    "MultiMeasureAttributeMappings": [
                        {
                            "SourceColumn": "wgt",
                            "TargetMultiMeasureAttributeName": "weight",
                            "MeasureValueType": "DOUBLE"
                        },
```

```
                        {
                            "SourceColumn": "spd",
                            "TargetMultiMeasureAttributeName": "speed",
                            "MeasureValueType": "DOUBLE"
                        },
                        {
                            "SourceColumn": "fuel",
                            "TargetMultiMeasureAttributeName": "fuel",
                            "MeasureValueType": "DOUBLE"
                        },
                        {
                            "SourceColumn": "miles",
                            "TargetMultiMeasureAttributeName": "miles",
                            "MeasureValueType": "DOUBLE"
                        }
                    ]
                }
            }
        },
        "TargetDatabaseName": "BatchLoadExampleDatabase",
        "TargetTableName": "BatchLoadExampleTable",
        "TaskStatus": "FAILED",
        "RecordVersion": 1,
        "CreationTime": 1677167593.266,
        "LastUpdatedTime": 1677167602.38
    }
}
```

バッチロードタスクを一覧表示する

バッチロードタスクを次のように一覧表示できます。

```
aws timestream-write list-batch-load-tasks
```

出力は次のように表示されます。

```
{
    "BatchLoadTasks": [
        {
            "TaskId": "<TaskId>",
            "TaskStatus": "FAILED",
            "DatabaseName": "BatchLoadExampleDatabase",
            "TableName": "BatchLoadExampleTable",
```

```
            "CreationTime": 1677167593.266,
            "LastUpdatedTime": 1677167602.38
        }
    ]
}
```

バッチロードタスクを再開する

バッチロードタスクは、次のように再開できます。

```
aws timestream-write resume-batch-load-task --task-id <value>
```

レスポンスは、成功を示すか、エラー情報を含むことができます。

バッチロードタスクの作成例

Example

1. という名前の LiveAnalytics データベースBatchLoadと という名前のテーブルの Timestream を作成しますBatchLoadTest。を確認し、必要に応じて MemoryStoreRetentionPeriodInHoursと の値を調整しますMagneticStoreRetentionPeriodInDays。

   ```
   aws timestream-write create-database --database-name BatchLoad \

   aws timestream-write create-table --database-name BatchLoad \
   --table-name BatchLoadTest \
   --retention-properties "{\"MemoryStoreRetentionPeriodInHours\": 12,
    \"MagneticStoreRetentionPeriodInDays\": 100}"
   ```

2. コンソールを使用して S3 バケットを作成し、その場所にsample.csvファイルをコピーします。サンプルは、サンプル CSVCSVでダウンロードできます。
3. コンソールを使用して、バッチロードタスクがエラーで完了した場合に がレポートを書き込む LiveAnalytics ための Timestream 用の S3 バケットを作成します。
4. バッチロードタスクを作成します。必ず を置き換えてください *$INPUT_BUCKET* また、*$REPORT_BUCKET* 前のステップで作成したバケットを使用します。

   ```
   aws timestream-write create-batch-load-task \
   --data-model-configuration "{\
              \"DataModel\": {\
                \"TimeColumn\": \"timestamp\",\
   ```

```
              \"TimeUnit\": \"SECONDS\",\
              \"DimensionMappings\": [\
                {\
                  \"SourceColumn\": \"vehicle\"\
                },\
                {\
                  \"SourceColumn\": \"registration\",\
                  \"DestinationColumn\": \"license\"\
                }\
              ],
              \"MultiMeasureMappings\": {\
                \"TargetMultiMeasureName\": \"mva_measure_name\",\
                \"MultiMeasureAttributeMappings\": [\
                  {\
                    \"SourceColumn\": \"wgt\",\
                    \"TargetMultiMeasureAttributeName\": \"weight\",\
                    \"MeasureValueType\": \"DOUBLE\"\
                  },\
                  {\
                    \"SourceColumn\": \"spd\",\
                    \"TargetMultiMeasureAttributeName\": \"speed\",\
                    \"MeasureValueType\": \"DOUBLE\"\
                  },\
                  {\
                    \"SourceColumn\": \"fuel_consumption\",\
                    \"TargetMultiMeasureAttributeName\": \"fuel\",\
                    \"MeasureValueType\": \"DOUBLE\"\
                  },\
                  {\
                    \"SourceColumn\": \"miles\",\
                    \"MeasureValueType\": \"BIGINT\"\
                  }\
                ]\
              }\
            }\
          }" \
--data-source-configuration "{
            \"DataSourceS3Configuration\": {\
              \"BucketName\": \"$INPUT_BUCKET\",\
              \"ObjectKeyPrefix\": \"$INPUT_OBJECT_KEY_PREFIX\"
            },\
            \"DataFormat\": \"CSV\"\
          }" \
--report-configuration "{\
```

```
            \"ReportS3Configuration\": {\
              \"BucketName\": \"$REPORT_BUCKET\",\
              \"EncryptionOption\": \"SSE_S3\"\
            }\
          }" \
--target-database-name BatchLoad \
--target-table-name BatchLoadTest
```

前述のコマンドは、次の出力を返します。

```
{
    "TaskId": "TaskId "
}
```

5. タスクの進行状況を確認します。必ず *$TASK_ID* 前のステップで返されたタスク ID を使用します。

```
aws timestream-write describe-batch-load-task --task-id $TASK_ID
```

出力例

```
{
    "BatchLoadTaskDescription": {
        "ProgressReport": {
            "BytesIngested": 1024,
            "RecordsIngested": 2,
            "FileFailures": 0,
            "RecordIngestionFailures": 0,
            "RecordsProcessed": 2,
            "FileParseFailures": 0
        },
        "DataModelConfiguration": {
            "DataModel": {
                "DimensionMappings": [
                    {
                        "SourceColumn": "vehicle",
                        "DestinationColumn": "vehicle"
                    },
                    {
                        "SourceColumn": "registration",
                        "DestinationColumn": "license"
```

```
                }
            ],
            "TimeUnit": "SECONDS",
            "TimeColumn": "timestamp",
            "MultiMeasureMappings": {
                "MultiMeasureAttributeMappings": [
                    {
                        "TargetMultiMeasureAttributeName": "weight",
                        "SourceColumn": "wgt",
                        "MeasureValueType": "DOUBLE"
                    },
                    {
                        "TargetMultiMeasureAttributeName": "speed",
                        "SourceColumn": "spd",
                        "MeasureValueType": "DOUBLE"
                    },
                    {
                        "TargetMultiMeasureAttributeName": "fuel",
                        "SourceColumn": "fuel_consumption",
                        "MeasureValueType": "DOUBLE"
                    },
                    {
                        "TargetMultiMeasureAttributeName": "miles",
                        "SourceColumn": "miles",
                        "MeasureValueType": "DOUBLE"
                    }
                ],
                "TargetMultiMeasureName": "mva_measure_name"
            }
        }
    },
    "TargetDatabaseName": "BatchLoad",
    "CreationTime": 1672960381.735,
    "TaskStatus": "SUCCEEDED",
    "RecordVersion": 1,
    "TaskId": "TaskId ",
    "TargetTableName": "BatchLoadTest",
    "ReportConfiguration": {
        "ReportS3Configuration": {
            "EncryptionOption": "SSE_S3",
            "ObjectKeyPrefix": "ObjectKeyPrefix ",
            "BucketName": "test-report-bucket"
        }
    },
```

```
        "DataSourceConfiguration": {
            "DataSourceS3Configuration": {
                "ObjectKeyPrefix": "sample.csv",
                "BucketName": "test-input-bucket"
            },
            "DataFormat": "CSV",
            "CsvConfiguration": {}
        },
        "LastUpdatedTime": 1672960387.334
    }
}
```

でのバッチロードの使用 AWS SDKs

を使用して AWS バッチロードタスクを作成、説明、一覧表示する方法については、SDKs、、バッチロードタスクの作成バッチロードタスクの説明バッチロードタスクを一覧表示する、および を参照してくださいバッチロードタスクを再開する。

バッチロードエラーレポートの使用

バッチロードタスクには、次のいずれかのステータス値があります。

- CREATED (作成済み) – タスクが作成されます。
- IN_PROGRESS (進行中) – タスクが進行中です。
- FAILED (失敗) – タスクが完了しました。ただし、1 つ以上のエラーが検出されました。
- SUCCEEDED (完了済み) – タスクはエラーなしで完了しました。
- PROGRESS_STOPPED (進捗停止) – タスクは停止しましたが、完了していません。タスクを再開できます。
- PENDING_RESUME (再開待ち) – タスクは再開待ちです。

エラーが発生すると、エラーログレポートは、そのために定義された S3 バケットに作成されます。エラーは、別々の配列で taskErrors または fileErrors に分類されます。以下はエラーレポートの例です。

```
{
    "taskId": "9367BE28418C5EF902676482220B631C",
    "taskErrors": [],
    "fileErrors": [
        {
```

```
            "fileName": "example.csv",
            "errors": [
                {
                    "reason": "The record timestamp is outside the time range of the
 data ingestion window.",
                    "lineRanges": [
                        [
                            2,
                            3
                        ]
                    ]
                }
            ]
        }
    ]
}
```

の Timestream でスケジュールされたクエリを使用する LiveAnalytics

Amazon Timestream for のスケジュールされたクエリ機能は、オペレーションダッシュボード、ビジネスレポート、アドホック分析、その他のアプリケーションに通常使用される、集計、ロールアップ、その他の形式の前処理済みデータを計算して保存するための、 LiveAnalytics フルマネージド型のサーバーレスでスケーラブルなソリューションです。スケジュールされたクエリにより、リアルタイム分析のパフォーマンスとコスト効率が向上し、データから追加のインサイトを導き出し、より良いビジネス上の意思決定を引き続き行うことができます。

スケジュールされたクエリでは、データの集計、ロールアップ、およびその他のオペレーションを計算するリアルタイム分析クエリと、 の Amazon Timestream を定義します。Amazon Timestream は、これらのクエリ LiveAnalytics を定期的に自動的に実行し、クエリ結果を別のテーブルに確実に書き込みます。通常、データは数分以内に計算され、これらのテーブルに更新されます。

その後、ダッシュボードとレポートをポイントして、かなり大きなソーステーブルをクエリする代わりに、集計データを含むテーブルをクエリできます。これにより、パフォーマンスとコストは大幅に向上します。これは、集計データを含むテーブルに含まれるデータ量がソーステーブルよりもはるかに少ないため、クエリが高速化され、データストレージが安価になるためです。

さらに、スケジュールされたクエリを持つテーブルは、 LiveAnalytics テーブルの Timestream の既存の機能をすべて提供します。例えば、 を使用してテーブルをクエリできますSQL。Grafana を使用して、テーブルに保存されているデータを視覚化できます。Amazon Kinesis 、Amazon 、MSK

AWS IoT Core、Telegraf を使用してテーブルにデータを取り込むこともできます。自動データライフサイクル管理のために、これらのテーブルのデータ保持ポリシーを設定できます。

集約されたデータを含むテーブルのデータ保持はソーステーブルのデータ保持と完全に切り離されるため、ソーステーブルのデータ保持を減らし、集約されたデータをデータストレージコストのごく一部ではるかに長い期間保持することもできます。スケジュールされたクエリにより、リアルタイム分析が速く、安価になり、より多くの顧客がアクセスしやすくなるため、顧客はアプリケーションをモニタリングし、データ主導型のビジネス上の意思決定を改善できます。

トピック

スケジュールされたクエリの利点

スケジュールされたクエリの利点は次のとおりです。

- 運用の容易さ – スケジュールされたクエリはサーバーレスでフルマネージドです。
- パフォーマンスとコスト – スケジュールされたクエリは、データの集計、ロールアップ、またはその他のリアルタイム分析オペレーションを事前計算し、結果をテーブルに保存するため、スケジュールされたクエリによって入力されたテーブルにアクセスするクエリには、ソーステーブルよりも少ないデータしか含まれません。したがって、これらのテーブルで実行されるクエリは、より高速で安価になります。スケジュールされた計算によって入力されたテーブルには、ソーステーブルよりも少ないデータが含まれているため、ストレージコストを削減できます。また、ソースデータをメモリストアに保持するコストのごく一部で、このデータをメモリストアに長期間保持することもできます。
- 相互運用性 – スケジュールされたクエリによって入力されたテーブルは、 LiveAnalytics テーブルの Timestream のすべての既存の機能を提供し、 の Timestream で動作するすべてのサービスや

ツールで使用できます LiveAnalytics。詳細については、「他の サービスの使用」を参照してください。

スケジュールされたクエリのユースケース

アプリケーションからのエンドユーザーアクティビティを要約するビジネスレポートにスケジュールされたクエリを使用して、パーソナライズのための機械学習モデルをトレーニングできます。異常、ネットワーク侵入、または不正行為を検出するアラームにスケジュールされたクエリを使用できるため、すぐに修正アクションを実行できます。

さらに、スケジュールされたクエリを使用して、より効果的なデータガバナンスを実現できます。これを行うには、スケジュールされたクエリにのみソーステーブルアクセスを許可し、スケジュールされたクエリによって入力されたテーブルのみへのアクセスをデベロッパーに許可します。これにより、意図しない長時間実行されるクエリの影響が最小限に抑えられます。

例: リアルタイム分析を使用して不正な支払いを検出し、より良いビジネス上の意思決定を行う

米国の大都市に分散された複数の point-of-saleターミナルから送信されたトランザクションを処理する支払いシステムを検討してください。 LiveAnalytics の Amazon Timestream を使用してトランザクションデータを保存および分析すると、不正なトランザクションを検出し、リアルタイム分析クエリを実行できます。これらのクエリは、1 時間あたりの最も混雑しているターミナルと最も使用 point-of-saleされていないターミナル、各都市の 1 日のうち最も混雑している時間、1 時間あたりのトランザクションが多い都市など、ビジネス上の質問に答えるのに役立ちます。

システムは 1 分あたり約 100,000 件のトランザクションを処理します。の LiveAnalytics Amazon Timestream に保存される各トランザクションは 100 バイトです。さまざまな種類の不正な支払いを検出するために、1 分ごとに実行される 10 個のクエリを設定しました。また、ビジネス上の質問に答えるために、さまざまなディメンションに沿ってデータを集約してスライス/ダイスする 25 個のクエリを作成しました。これらのクエリはそれぞれ、過去 1 時間のデータを処理します。

これらのクエリによって生成されたデータを表示するダッシュボードを作成しました。ダッシュボードには 25 個のウィジェットが含まれており、1 時間ごとに更新され、通常は 10 人のユーザーがいつでもアクセスできます。最後に、メモリストアには 2 時間のデータ保持期間が設定され、マグネティックストアには 6 か月のデータ保持期間が設定されました。

この場合、ダッシュボードにアクセスして更新するたびにデータを再計算するリアルタイム分析クエリを使用するか、ダッシュボードに派生テーブルを使用できます。リアルタイム分析クエリに基

づくダッシュボードのクエリコストは、1 か月あたり 120.70 USD になります。対照的に、派生テーブルを使用したダッシュボードクエリのコストは、月額 12.27 USD になります (料金については LiveAnalytics、「Amazon Timestream」を参照してください)。この場合、派生テーブルを使用すると、クエリコストが約 10 倍削減されます。

スケジュールされたクエリの概念

クエリ文字列 - これは、結果を事前に計算し、 LiveAnalytics テーブルの別の Timestream に保存しているクエリです。スケジュールされたクエリは、 の Timestream の全SQL領域を使用して定義できます。これにより LiveAnalytics、一般的なテーブル式、ネストされたクエリ、ウィンドウ関数、またはクエリ言語 の Timestream でサポートされているあらゆる種類の集計関数とスカラー関数を使用して LiveAnalytics クエリを柔軟に記述できます。

スケジュール式 - スケジュールされたクエリインスタンスの実行日時を指定できます。式は、cron 式 (UTC毎日午前 8 時に実行するなど) またはレート式 (10 分ごとに実行するなど) を使用して指定できます。

ターゲット設定 - スケジュールされたクエリの結果を、このスケジュールされたクエリの結果が保存される宛先テーブルにマッピングする方法を指定できます。

通知設定 - のタイムストリームは、スケジュール式に基づいてスケジュールされたクエリのインスタンス LiveAnalytics を自動的に実行します。スケジュールされたクエリの作成時に設定したSNSトピックで実行されるクエリごとに通知を受け取ります。この通知は、インスタンスが正常に実行されたか、エラーが発生したかを指定します。さらに、計測されたバイト数、ターゲットテーブルに書き込まれたデータ、次の呼び出し時間などの情報を提供します。

以下は、この種の通知メッセージの例です。

```
{
    "type":"AUTO_TRIGGER_SUCCESS",
    "arn":"arn:aws:timestream:us-east-1:123456789012:scheduled-query/
 PT1mPerMinutePerRegionMeasureCount-9376096f7309",
    "nextInvocationEpochSecond":1637302500,
    "scheduledQueryRunSummary":
    {
        "invocationEpochSecond":1637302440,
        "triggerTimeMillis":1637302445697,
        "runStatus":"AUTO_TRIGGER_SUCCESS",
        "executionStats":
        {
            "executionTimeInMillis":21669,
```

```
            "dataWrites":36864,
            "bytesMetered":13547036820,
            "recordsIngested":1200,
            "queryResultRows":1200
        }
    }
}
```

この通知メッセージでは、 bytesMeteredはソーステーブルでクエリがスキャンしたバイトで、dataWrites はターゲットテーブルに書き込まれたバイトです。

> **Note**
>
> これらの通知をプログラムで使用している場合は、今後新しいフィールドが通知メッセージに追加される可能性があることに注意してください。

エラーレポートの場所 - スケジュールされたクエリは非同期的に実行され、ターゲットテーブルにデータを保存します。インスタンスにエラーが発生した場合 (保存できなかった無効なデータなど) 、エラーが発生したレコードは、スケジュールされたクエリの作成時に指定したエラーレポートの場所のエラーレポートに書き込まれます。S3 バケットとロケーションのプレフィックスを指定します。の Timestream は、スケジュールされたクエリの名前と呼び出し時間をこのプレフィックス LiveAnalytics に追加し、スケジュールされたクエリの特定のインスタンスに関連付けられたエラーを特定するのに役立ちます。

タグ付け - スケジュールされたクエリに関連付けるタグをオプションで指定できます。詳細については、「リソースの LiveAnalytics Timestream のタグ付け」を参照してください。

例

次の例では、スケジュールされたクエリを使用して単純な集計を計算します。

```
SELECT region, bin(time, 1m) as minute,
    SUM(CASE WHEN measure_name = 'metrics' THEN 20 ELSE 5 END) as numDataPoints
FROM raw_data.devops
WHERE time BETWEEN @scheduled_runtime - 10m AND @scheduled_runtime + 1m
GROUP BY bin(time, 1m), region
```

@scheduled_runtime parameter - この例では、クエリが特別な名前付きパラメータ を受け入れることに気付くでしょう@scheduled_runtime。これは、スケジュールされたクエリの特定のイン

スタンスを呼び出すときにサービスが設定する特別なパラメータ (タイプのタイムスタンプ) です。これにより、スケジュールされたクエリの特定のインスタンスがソーステーブル内のデータを分析する時間範囲を決定的に制御できます。クエリ`@scheduled_runtime`では、タイムスタンプタイプが予想される任意の場所で を使用できます。

スケジュール式を設定する例を考えてみます。cron(0/5 * * * ? *) では、スケジュールされたクエリは 1 時間ごとに 0、5、10、15、20、25、30、35、40、45、50、55 分に実行されます。2021-12-01 00:05:00 にトリガーされるインスタンスの場合、@scheduled_runtime パラメータはこの値に初期化されるため、現在のインスタンスは 2021-11-30 23:55:00 から 2021-12-01 00:06:00 までのデータで動作します。

時間範囲が重複するインスタンス - この例で示すように、スケジュールされたクエリの後続の 2 つのインスタンスは、その時間範囲で重複する可能性があります。これは、要件、指定した時間述語、スケジュール式に基づいて制御できます。この場合、この重複により、これらの計算は、この例で到着がわずかに遅延したデータに基づいて集計を更新できます。2021-12-01 00:00:00 にトリガーされたクエリ実行は、2021-11-30 23:50:00 から 2021-12-30 00:01:00 の時間範囲をカバーし、2021-12-01 00:05:00 にトリガーされたクエリ実行は、2021-11-30 23:55:00 から 2021-12-01 00:06:00 の範囲をカバーします。

正確性を確保し、ターゲットテーブルに保存されている集計がソーステーブルから計算された集計と一致するように、 の Timestream は、2021-12-01 00:05:00 の計算が完了した後にのみ 2021-12-01 00:00:00 の計算が実行されるように LiveAnalytics します。後者の計算結果は、新しい値が生成された場合、以前にマテリアライズドされた集計を更新できます。Timestream for は内部的にレコードバージョン LiveAnalytics を使用し、スケジュールされたクエリの後者のインスタンスによって生成されたレコードには、より高いバージョン番号が割り当てられます。したがって、2021-12-01 00:05:00 の呼び出しによって計算された集計は、ソーステーブルで新しいデータが利用可能であると仮定して、2021-12-01 00:00:00 の呼び出しによって計算された集計を更新できます。

自動トリガーと手動トリガー - スケジュールされたクエリが作成されると、 の Timestream LiveAnalytics は指定されたスケジュールに基づいてインスタンスを自動的に実行します。このような自動トリガーは、 サービスによって完全に管理されます。

ただし、スケジュールされたクエリの一部のインスタンスを手動で開始するシナリオがある場合があります。例えば、クエリの実行で特定のインスタンスが失敗した場合、自動スケジュール実行後にソーステーブルに遅延受信データや更新があった場合、自動クエリ実行の対象とならない時間範囲 (スケジュールされたクエリの作成前の時間範囲など) のターゲットテーブルを更新する場合などです。

を使用して ExecuteScheduledQuery API、@scheduled_runtime InvocationTime パラメータに使用される値である パラメータを渡すことで、スケジュールされたクエリの特定のインスタンスを手動で開始できます。を使用する際の重要な考慮事項を以下に示します ExecuteScheduledQuery API。

- これらの呼び出しの複数をトリガーする場合は、これらの呼び出しが重複する時間範囲で結果を生成しないようにする必要があります。重複しない時間範囲を確保できない場合は、これらのクエリの実行が順番に開始されていることを確認してください。時間範囲内で重複する複数のクエリ実行を同時に開始すると、トリガーエラーが表示され、これらのクエリ実行のエラーレポートにバージョン競合が表示される可能性があります。
- @scheduled_runtime の任意のタイムスタンプ値を使用して呼び出しを開始できます。したがって、ソーステーブルでデータが更新された範囲に対応するターゲットテーブルで適切な時間範囲が更新されるように、値を適切に設定するのはユーザーの責任です。

スケジュールされたクエリのスケジュール式

cron 式またはレート式を使用するスケジュールされたクエリには Amazon Timestream を使用して、LiveAnalytics スケジュールされたクエリを自動スケジュールで作成できます。スケジュールされたクエリはすべてUTCタイムゾーンを使用し、スケジュールの最小精度は 1 分です。

スケジュール式を指定する 2 つの方法は、cron と rate です。Cron 式はよりきめ細かなスケジュール制御を提供しますが、レート式は表現が簡単ですが、きめ細かな制御はありません。

例えば、cron 式を使用すると、毎週または毎月の特定の日の特定時刻、または月曜日から金曜日までは 1 時間ごとに指定された分にトリガーされるスケジュールされたクエリを定義できます。対照的に、レート式は、スケジュールされたクエリが作成された正確な時刻から、1 分、1 時間、または 1 日に 1 回など、スケジュールされたクエリを定期的に開始します。

Cron 式

- [Syntax] (構文)

```
cron(fields)
```

cron 式には 6 つの必須フィールドがあり、それらは空白で区切られます。

フィールド	値	ワイルドカード
分	0-59	, - * /

フィールド	値	ワイルドカード
時間	0-23	, - * /
Day-of-month	1-31	, - * ? / L W
月	1 ~ 12 または JAN-DEC	, - * /
Day-of-week	1 ~ 7 または SUN-SAT	, - * ? L #
年	1970-2199	, - * /

ワイルドカード文字

- *、* (コンマ) ワイルドカードには追加の値が含まれます。Month フィールドの JANには、1 月 FEB、MAR2 月、3 月が含まれます。
- *-* (ダッシュ) ワイルドカードは範囲を指定します。日フィールドの、「1-15」は、指定した月の 1 日から 15 日を含みます。
- *** (アスタリスク) ワイルドカードには、 フィールド内のすべての値が含まれます。時間 フィールドでは、*** には 1 時間ごとに が含まれます。フィールドと Day-of-week フィールドの両方で Day-of-month *** を使用することはできません。一方で使用する場合は、他方で *?* を使用する必要があります。
- */* (フォワードスラッシュ) ワイルドカードは増分を指定します。Minutes フィールドでは、10 分ごとに 1/10 と入力して、1 時間の最初の 1 分から開始して指定できます (11、21、31 分など)。
- *?* (疑問符) ワイルドカードは、どちらかを指定します。 Day-of-month フィールドに *7* と入力し、7 日がどの曜日であるかを気にしない場合は、フィールドに *?* と Day-of-week入力できます。
- または Day-of-week フィールドの Day-of-month *L* ワイルドカードは、月または週の最終日を指定します。
- フィールドの Day-of-month W ワイルドカードは平日を指定します。 Day-of-month フィールドで、3W は月の 3 日に最も近い平日を指定します。
- フィールドの Day-of-week *#* ワイルドカードは、1 か月内の指定した曜日の特定のインスタンスを指定します。例えば、3#2 は、月の第 2 火曜日を示します。3 は週の 3 番目の日 (火曜日) を示し、2 は月のそのタイプの 2 番目の日を示します。

> **Note**
>
> '#' 文字を使用する場合は、 day-of-weekフィールドで定義できる式は 1 つだけです。例えば、「3#1,6#3」は 2 つの式として解釈されるため、無効です。

制約事項

- 同じ cron 式で Day-of-monthおよび Day-of-week フィールドを指定することはできません。いずれかのフィールドに値 (または *) を指定する場合は、もう一方のフィールドに *?* (疑問符) を使用する必要があります。
- 1 分より短い間隔を導き出す cron 式はサポートされていません。

例

分	時間	日	月	曜日	年	意味
0	10	*	*	?	*	毎日午前 10:00 (UTC) に実行します。
15	12	*	*	?	*	毎日午後 12:15 (UTC) に実行します。
0	18	?	*	MON-FRI	*	毎週月曜日から金曜日の午後 6 時 (UTC) に実行します。

分	時間	日	月	曜日	年	意味
0	8	1	*	?	*	毎月 1 日午前 8:00 (UTC) に実行します。
0/15	*	*	*	?	*	15 分ごとに を実行します。
0/10	*	*	*	MON-FRI	*	月曜日から金曜日まで 10 分ごとに実行します。
0/5	8-17	?	*	MON-FRI	*	月曜日から金曜日の午前 8 時から午後 5 時 55 分 () まで 5 分ごとに実行しますUTC。

rate 式

- rate 式は、予定されたイベントルールを作成すると開始され、その定義済されたスケジュールに基づいて実行されます。rate 式は 2 つの必須フィールドがあります。フィールドは空白で区切ります。

  [Syntax] (構文)

```
rate(value unit)
```

- value: 正の数。
- unit: 時間単位。1 の値 (分など) と 1 を超える値 (分など) には、異なる単位が必要です。有効な値: minute | minutes | hour | hours | day | days

スケジュールされたクエリのデータモデルマッピング

の Timestream は、テーブル内のデータの柔軟なモデリング LiveAnalytics をサポートし、同じ柔軟性が LiveAnalytics 、テーブルの別の Timestream にマテリアライズされたスケジュールされたクエリの結果に適用されます。スケジュールされたクエリでは、マルチメジャーレコードまたは単一メジャーレコードにデータがあるかどうかにかかわらず、任意のテーブルをクエリし、マルチメジャーレコードまたは単一メジャーレコードを使用してクエリ結果を記述できます。

スケジュールされたクエリの仕様 TargetConfiguration で を使用して、クエリ結果を宛先派生テーブルの適切な列にマッピングします。以下のセクションでは、派生テーブルで異なるデータモデル TargetConfiguration を実現するためにこれを指定するさまざまな方法について説明します。具体的には、以下が表示されます。

- クエリ結果にメジャー名がなく、 でターゲットメジャー名を指定した場合に、マルチメジャーレコードに書き込む方法 TargetConfiguration。
- クエリ結果でメジャー名を使用してマルチメジャーレコードを書き込む方法。
- マルチメジャー属性が異なる複数のレコードを書き込むモデルを定義する方法。
- 派生テーブルの単一測定レコードに書き込むモデルを定義する方法。
- スケジュールされたクエリで単一測定レコードおよび/または複数測定レコードをクエリし、結果を単一測定レコードまたは複数測定レコードのいずれかにマテリアライズする方法。これにより、データモデルの柔軟性を選択できます。

例: マルチメジャーレコードのターゲットメジャー名

この例では、クエリがマルチメジャーデータを含むテーブルからデータを読み取り、マルチメジャーレコードを使用して結果を別のテーブルに書き込んでいることがわかります。スケジュールされたクエリ結果には、自然なメジャー名列がありません。ここでは、 の TargetMultiMeasureName プロパティを使用して、派生テーブルのメジャー名を指定します TargetConfigurationTimestreamConfiguration。

```
{
    "Name" : "CustomMultiMeasureName",
    "QueryString" : "SELECT region, bin(time, 1h) as hour, AVG(memory_cached)
 as avg_mem_cached_1h, MIN(memory_free) as min_mem_free_1h, MAX(memory_used) as
 max_mem_used_1h, SUM(disk_io_writes) as sum_1h, AVG(disk_used) as avg_disk_used_1h,
 AVG(disk_free) as avg_disk_free_1h, MAX(cpu_user) as max_cpu_user_1h, MIN(cpu_idle) as
 min_cpu_idle_1h, MAX(cpu_system) as max_cpu_system_1h FROM raw_data.devops_multi WHERE
 time BETWEEN bin(@scheduled_runtime, 1h) - 14h AND bin(@scheduled_runtime, 1h) - 2h
 AND measure_name = 'metrics' GROUP BY region, bin(time, 1h)",
    "ScheduleConfiguration" : {
        "ScheduleExpression" : "cron(0 0/1 * * ? *)"
    },
    "NotificationConfiguration" : {
        "SnsConfiguration" : {
            "TopicArn" : "******"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******",
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName" : "derived",
            "TableName" : "dashboard_metrics_1h_agg_1",
            "TimeColumn" : "hour",
            "DimensionMappings" : [
                {
                    "Name": "region",
                    "DimensionValueType" : "VARCHAR"
                }
            ],
            "MultiMeasureMappings" : {
                "TargetMultiMeasureName": "dashboard-metrics",
                "MultiMeasureAttributeMappings" : [
                    {
                        "SourceColumn" : "avg_mem_cached_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName" : "avgMemCached"
                    },
                    {
                        "SourceColumn" : "min_mem_free_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "max_mem_used_1h",
```

```
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "sum_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName" : "totalDiskWrites"
                    },
                    {
                        "SourceColumn" : "avg_disk_used_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "avg_disk_free_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "max_cpu_user_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName" : "CpuUserP100"
                    },
                    {
                        "SourceColumn" : "min_cpu_idle_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "max_cpu_system_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName" : "CpuSystemP100"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    }
}
```

この例のマッピングでは、測定名のダッシュボードメトリクスと
属性名 avgMemCached、min_mem_free_1h、max_mem_used_1h
totalDiskWrites、avg_disk_used_1h、avg_disk_free_1h、 CpuUserP100、min_cpu_idle_1h、
CpuSystemP100 を含む 1 つのマルチメジャーレコードを作成します。オプション
TargetMultiMeasureAttributeName で を使用して、クエリ出力列の名前を結果マテリアライズに使用される別の属性名に変更します。

このスケジュールされたクエリがマテリアライズされた後の宛先テーブルのスキーマを次に示します。次の結果の LiveAnalytics 属性タイプの Timestream からわかるように、結果は、メジャースキーマに示すようにdashboard-metrics、単一メジャー名 を持つマルチメジャーレコードにマテリアライズされます。

列	タイプ	LiveAnalytics 属性タイプのタイムストリーム
region	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
時系	timestamp	TIMESTAMP
CpuSystemP100	double	MULTI
avgMemCached	double	MULTI
min_cpu_idle_1h	double	MULTI
avg_disk_free_1h	double	MULTI
avg_disk_used_1h	double	MULTI
totalDiskWrites	double	MULTI
max_mem_used_1h	double	MULTI
最小_mem_free_1h	double	MULTI
CpuUserP100	double	MULTI

以下は、SHOWMEASURESクエリで取得した対応する測定値です。

measure_name	data_type	ディメンション
ダッシュボードメトリクス	マルチ	[{'dimension_name': 'region', 'data_type': 'varchar'}]

例: マルチメジャーレコードでスケジュールされたクエリからのメジャー名を使用する

この例では、単一測定レコードを含むテーブルからのクエリ読み取りと、結果を複数測定レコードにマテリアライズしています。この場合、スケジュールされたクエリ結果には、スケジュールされたクエリの結果がマテリアライズされたターゲットテーブルのメジャー名として値を使用できる列があります。次に、 の MeasureNameColumn プロパティを使用して、派生テーブルのマルチメジャーレコードのメジャー名を指定できます TargetConfigurationTimestreamConfiguration。

```
{
    "Name" : "UsingMeasureNameFromQueryResult",
    "QueryString" : "SELECT region, bin(time, 1h) as hour, measure_name, AVG(CASE WHEN
 measure_name IN ('memory_cached', 'disk_used', 'disk_free') THEN measure_value::double
 ELSE NULL END) as avg_1h, MIN(CASE WHEN measure_name IN ('memory_free', 'cpu_idle')
 THEN measure_value::double ELSE NULL END) as min_1h, SUM(CASE WHEN measure_name
 IN ('disk_io_writes') THEN measure_value::double ELSE NULL END) as sum_1h,
 MAX(CASE WHEN measure_name IN ('memory_used', 'cpu_user', 'cpu_system') THEN
 measure_value::double ELSE NULL END) as max_1h FROM raw_data.devops WHERE time
 BETWEEN bin(@scheduled_runtime, 1h) - 14h AND bin(@scheduled_runtime, 1h) - 2h AND
 measure_name IN ('memory_free', 'memory_used', 'memory_cached', 'disk_io_writes',
 'disk_used', 'disk_free', 'cpu_user', 'cpu_system', 'cpu_idle') GROUP BY region,
 measure_name, bin(time, 1h)",
    "ScheduleConfiguration" : {
        "ScheduleExpression" : "cron(0 0/1 * * ? *)"
    },
    "NotificationConfiguration" : {
        "SnsConfiguration" : {
            "TopicArn" : "******"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******",
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName" : "derived",
            "TableName" : "dashboard_metrics_1h_agg_2",
            "TimeColumn" : "hour",
            "DimensionMappings" : [
```

```
                {
                    "Name": "region",
                    "DimensionValueType" : "VARCHAR"
                }
            ],
            "MeasureNameColumn" : "measure_name",
            "MultiMeasureMappings" : {
                "MultiMeasureAttributeMappings" : [
                    {
                        "SourceColumn" : "avg_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "min_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName": "p0_1h"
                    },
                    {
                        "SourceColumn" : "sum_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "max_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName": "p100_1h"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    }
}
```

この例のマッピングでは、属性 avg_1h、p0_1h、sum_1h、p100_1h を持つマルチメジャーレコードが作成され、クエリ結果の measure_name 列の値が宛先テーブルのマルチメジャーレコードのメジャー名として使用されます。さらに、前の例では、オプション TargetMultiMeasureAttributeName

で をマッピングのサブセットとともに使用して、属性の名前を変更することに注意してください。例えば、min_1h の名前が p0_1h に、max_1h の名前が p100_1h に変更されました。

このスケジュールされたクエリがマテリアライズされた後の宛先テーブルのスキーマを次に示します。次の結果の LiveAnalytics 属性タイプの Timestream からわかるように、結果はマルチメジャーレコードにマテリアライズされます。メジャースキーマを見ると、クエリ結果に表示される値に対応する 9 つの異なるメジャー名が取り込まれていました。

列	タイプ	LiveAnalytics 属性タイプのタイムストリーム
region	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
時系	timestamp	TIMESTAMP
sum_1h	double	MULTI
p100_1h	double	MULTI
p0_1h	double	MULTI
avg_1h	double	MULTI

以下は、SHOWMEASURESクエリで取得した対応する測定値です。

measure_name	data_type	ディメンション
cpu_idle	マルチ	[{'dimension_name': 'region', 'data_type': 'varchar'}]
cpu_system	マルチ	[{'dimension_name': 'region', 'data_type': 'varchar'}]
cpu_user	マルチ	[{'dimension_name': 'region', 'data_type': 'varchar'}]

measure_name	data_type	ディメンション
disk_free	マルチ	[{'dimension_name': 'region', 'data_type': 'varchar'}]
disk_io_writes	マルチ	[{'dimension_name': 'region', 'data_type': 'varchar'}]
disk_used	マルチ	[{'dimension_name': 'region', 'data_type': 'varchar'}]
memory_cached	マルチ	[{'dimension_name': 'region', 'data_type': 'varchar'}]
memory_free	マルチ	[{'dimension_name': 'region', 'data_type': 'varchar'}]
memory_free	マルチ	[{'dimension_name': 'region', 'data_type': 'varchar'}]

例: 異なる属性を持つ異なるマルチメジャーレコードに結果をマッピングする

次の例は、クエリ結果の異なる列を、異なるメジャー名を持つ異なるマルチメジャーレコードにマッピングする方法を示しています。スケジュールされたクエリ定義が次の場合、クエリの結果には次の列があります。リージョン、時間、avg_mem_cached_1h、min_mem_free_1h、max_mem_used_1h、total_disk_io_writes_1h、avg_disk_us
region はディメンションにマッピングされ、時間列にマッピングhourされます。

の MixedMeasureMappings プロパティ TargetConfiguration。TimestreamConfiguration派生テーブルのマルチメジャーレコードにメジャーをマッピングする方法を指定します。

この特定の例では、avg_mem_cached_1h、min_mem_free_1h、max_mem_used_1h がメジャー名 mem_aggregates の 1 つのマルチメジャーレコードで使用され、total_disk_io_writes_1h、avg_disk_used_1h、avg_disk_free_1h がメジャー名 disk_aggregates の別のマルチメジャーレコードで使用され、最後に max_cpu_user_1h、max_cpu_system_1h、min_cpu_system_1h がメジャー名 cpu_aggregates の別のマルチメジャーレコードで使用されます。

これらのマッピングでは、オプション TargetMultiMeasureAttributeName で を使用してクエリ結果列の名前を変更し、宛先テーブルに異なる属性名を含めることもできます。例えば、結果列 avg_mem_cached_1h の名前が に変更され avgMemCached、total_disk_io_writes_1h の名前が に変更されますtotalIOWrites。

マルチメジャーレコードのマッピングを定義する場合、 の Timestream はクエリ結果のすべての行 LiveAnalytics を検査し、NULL値を持つ列値を自動的に無視します。その結果、複数のメジャー名を持つマッピングの場合、マッピング内のそのグループのすべての列値が特定の行NULLに対するものである場合、そのメジャー名の値はその行に対して取り込まれません。

例えば、次のマッピングでは、avg_mem_cached_1h、min_mem_free_1h、max_mem_used_1h が名前 mem_aggregates を測定するためにマッピングされます。クエリ結果の特定の行について、これらの列値がすべて の場合NULL、 の Timestream LiveAnalytics はその行の measure mem_aggregates を取り込まない。特定の行の 9 つの列がすべて の場合NULL、エラーレポートにユーザーエラーが報告されます。

```
{
    "Name" : "AggsInDifferentMultiMeasureRecords",
    "QueryString" : "SELECT region, bin(time, 1h) as hour, AVG(CASE WHEN measure_name
 = 'memory_cached' THEN measure_value::double ELSE NULL END) as avg_mem_cached_1h,
 MIN(CASE WHEN measure_name = 'memory_free' THEN measure_value::double ELSE
 NULL END) as min_mem_free_1h, MAX(CASE WHEN measure_name = 'memory_used' THEN
 measure_value::double ELSE NULL END) as max_mem_used_1h, SUM(CASE WHEN measure_name =
 'disk_io_writes' THEN measure_value::double ELSE NULL END) as total_disk_io_writes_1h,
 AVG(CASE WHEN measure_name = 'disk_used' THEN measure_value::double ELSE NULL END) as
 avg_disk_used_1h, AVG(CASE WHEN measure_name = 'disk_free' THEN measure_value::double
 ELSE NULL END) as avg_disk_free_1h, MAX(CASE WHEN measure_name = 'cpu_user' THEN
 measure_value::double ELSE NULL END) as max_cpu_user_1h, MAX(CASE WHEN measure_name
 = 'cpu_system' THEN measure_value::double ELSE NULL END) as max_cpu_system_1h,
 MIN(CASE WHEN measure_name = 'cpu_idle' THEN measure_value::double ELSE NULL END)
 as min_cpu_system_1h FROM raw_data.devops WHERE time BETWEEN bin(@scheduled_runtime,
 1h) - 14h AND bin(@scheduled_runtime, 1h) - 2h AND measure_name IN ('memory_cached',
 'memory_free', 'memory_used', 'disk_io_writes', 'disk_used', 'disk_free', 'cpu_user',
 'cpu_system', 'cpu_idle') GROUP BY region, bin(time, 1h)",
   "ScheduleConfiguration" : {
       "ScheduleExpression" : "cron(0 0/1 * * ? *)"
   },
   "NotificationConfiguration" : {
       "SnsConfiguration" : {
           "TopicArn" : "******"
       }
   },
```

```
    "ScheduledQueryExecutionRoleArn": "******",
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName" : "derived",
            "TableName" : "dashboard_metrics_1h_agg_3",
            "TimeColumn" : "hour",
            "DimensionMappings" : [
                {
                    "Name": "region",
                    "DimensionValueType" : "VARCHAR"
                }
            ],
            "MixedMeasureMappings" : [
                {
                    "MeasureValueType" : "MULTI",
                    "TargetMeasureName" : "mem_aggregates",
                    "MultiMeasureAttributeMappings" : [
                        {
                            "SourceColumn" : "avg_mem_cached_1h",
                            "MeasureValueType" : "DOUBLE",
                            "TargetMultiMeasureAttributeName": "avgMemCached"
                        },
                        {
                            "SourceColumn" : "min_mem_free_1h",
                            "MeasureValueType" : "DOUBLE"
                        },
                        {
                            "SourceColumn" : "max_mem_used_1h",
                            "MeasureValueType" : "DOUBLE",
                            "TargetMultiMeasureAttributeName": "maxMemUsed"
                        }
                    ]
                },
                {
                    "MeasureValueType" : "MULTI",
                    "TargetMeasureName" : "disk_aggregates",
                    "MultiMeasureAttributeMappings" : [
                        {
                            "SourceColumn" : "total_disk_io_writes_1h",
                            "MeasureValueType" : "DOUBLE",
                            "TargetMultiMeasureAttributeName": "totalIOWrites"
                        },
                        {
                            "SourceColumn" : "avg_disk_used_1h",
```

```
                            "MeasureValueType" : "DOUBLE"
                        },
                        {
                            "SourceColumn" : "avg_disk_free_1h",
                            "MeasureValueType" : "DOUBLE"
                        }
                    ]
                },
                {
                    "MeasureValueType" : "MULTI",
                    "TargetMeasureName" : "cpu_aggregates",
                    "MultiMeasureAttributeMappings" : [
                        {
                            "SourceColumn" : "max_cpu_user_1h",
                            "MeasureValueType" : "DOUBLE"
                        },
                        {
                            "SourceColumn" : "max_cpu_system_1h",
                            "MeasureValueType" : "DOUBLE"
                        },
                        {
                            "SourceColumn" : "min_cpu_idle_1h",
                            "MeasureValueType" : "DOUBLE",
                            "TargetMultiMeasureAttributeName": "minCpuIdle"
                        }
                    ]
                }
            ]
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    }
}
```

このスケジュールされたクエリがマテリアライズされた後の宛先テーブルのスキーマを次に示します。

列	タイプ	LiveAnalytics 属性タイプのタイムストリーム
region	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
時系	timestamp	TIMESTAMP
minCpuIdle	double	MULTI
最大_cpu_system_1h	double	MULTI
max_cpu_user_1h	double	MULTI
avgMemCached	double	MULTI
maxMemUsed	double	MULTI
最小_mem_free_1h	double	MULTI
avg_disk_free_1h	double	MULTI
avg_disk_used_1h	double	MULTI
totalIOWrites	double	MULTI

以下は、SHOWMEASURESクエリで取得した対応する測定値です。

measure_name	data_type	ディメンション
cpu_aggregates	マルチ	[{'dimension_name': 'region', 'data_type': 'varchar'}]
disk_aggregates	マルチ	[{'dimension_name': 'region', 'data_type': 'varchar'}]
mem_aggregates	マルチ	[{'dimension_name': 'region', 'data_type': 'varchar'}]

例: クエリ結果からメジャー名を持つ単一メジャーレコードに結果をマッピングする

以下は、結果が単一測定レコードにマテリアライズされたスケジュールされたクエリの例です。この例では、クエリ結果には measure_name 列があり、その値はターゲットテーブルのメジャー名として使用されます。 TargetConfiguration. の MixedMeasureMappings 属性を使用して、クエリ結果列のターゲットテーブルのスカラーメジャーへのマッピングTimestreamConfiguration を指定します。

次の定義例では、クエリ結果は 9 つの異なる measure_name 値であることが予想されます。これらのメジャー名をすべてマッピングに一覧表示し、そのメジャー名の単一メジャー値に使用する列を指定します。例えば、このマッピングでは、特定の結果行に memory_cached のメジャー名が表示される場合、データがターゲットテーブルに書き込まれるとき、avg_1h 列の値がメジャーの値として使用されます。オプションで TargetMeasureName を使用して、この値の新しいメジャー名を指定できます。

```
{
    "Name" : "UsingMeasureNameColumnForSingleMeasureMapping",
    "QueryString" : "SELECT region, bin(time, 1h) as hour, measure_name, AVG(CASE WHEN
 measure_name IN ('memory_cached', 'disk_used', 'disk_free') THEN measure_value::double
 ELSE NULL END) as avg_1h, MIN(CASE WHEN measure_name IN ('memory_free', 'cpu_idle')
 THEN measure_value::double ELSE NULL END) as min_1h, SUM(CASE WHEN measure_name
 IN ('disk_io_writes') THEN measure_value::double ELSE NULL END) as sum_1h,
 MAX(CASE WHEN measure_name IN ('memory_used', 'cpu_user', 'cpu_system') THEN
 measure_value::double ELSE NULL END) as max_1h FROM raw_data.devops WHERE time
 BETWEEN bin(@scheduled_runtime, 1h) - 14h AND bin(@scheduled_runtime, 1h) - 2h AND
 measure_name IN ('memory_free', 'memory_used', 'memory_cached', 'disk_io_writes',
 'disk_used', 'disk_free', 'cpu_user', 'cpu_system', 'cpu_idle') GROUP BY region,
 bin(time, 1h), measure_name",
    "ScheduleConfiguration" : {
        "ScheduleExpression" : "cron(0 0/1 * * ? *)"
    },
    "NotificationConfiguration" : {
        "SnsConfiguration" : {
            "TopicArn" : "******"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******",
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName" : "derived",
            "TableName" : "dashboard_metrics_1h_agg_4",
            "TimeColumn" : "hour",
            "DimensionMappings" : [
```

```
                {
                    "Name": "region",
                    "DimensionValueType" : "VARCHAR"
                }
            ],
            "MeasureNameColumn" : "measure_name",
            "MixedMeasureMappings" : [
                {
                    "MeasureName" : "memory_cached",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "avg_1h",
                    "TargetMeasureName" : "AvgMemCached"
                },
                {
                    "MeasureName" : "disk_used",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "avg_1h"
                },
                {
                    "MeasureName" : "disk_free",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "avg_1h"
                },
                {
                    "MeasureName" : "memory_free",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "min_1h",
                    "TargetMeasureName" : "MinMemFree"
                },
                {
                    "MeasureName" : "cpu_idle",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "min_1h"
                },
                {
                    "MeasureName" : "disk_io_writes",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "sum_1h",
                    "TargetMeasureName" : "total-disk-io-writes"
                },
                {
                    "MeasureName" : "memory_used",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_1h",
```

```
                    "TargetMeasureName" : "maxMemUsed"
                },
                {
                    "MeasureName" : "cpu_user",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_1h"
                },
                {
                    "MeasureName" : "cpu_system",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_1h"
                }
            ]
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    }
}
```

このスケジュールされたクエリがマテリアライズされた後の宛先テーブルのスキーマを次に示します。スキーマからわかるように、テーブルは単一測定レコードを使用しています。テーブルのメジャースキーマを一覧表示すると、仕様で提供されているマッピングに基づいて書き込まれた 9 つのメジャーが表示されます。

列	タイプ	LiveAnalytics 属性タイプのタイムストリーム
region	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
時系	timestamp	TIMESTAMP
measure_value::double	double	MEASURE_VALUE

以下は、SHOWMEASURESクエリで取得した対応する測定値です。

measure_name	data_type	ディメンション
AvgMemCached	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
MinMemFree	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
cpu_idle	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
cpu_system	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
cpu_user	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
disk_free	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
disk_used	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
maxMemUsed	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
total-disk-io-writes	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]

例: クエリ結果列をメジャー名とする単一メジャーレコードに結果をマッピングする

この例では、結果にメジャー名列がないクエリがあります。代わりに、出力を単一メジャーレコードにマッピングするときに、クエリ結果の列名をメジャー名として使用します。以前は、同様の結果がマルチメジャーレコードに書き込まれた例がありました。この例では、 がアプリケーションシナリオに適合する場合に、単一測定レコードにマッピングする方法を示します。

ここでも、 の MixedMeasureMappings プロパティを使用してこのマッピングを指定します TargetConfigurationTimestreamConfiguration。次の例では、クエリ結果に 9 つの列があることがわかります。結果列をメジャー名として使用し、値を単一メジャー値として使用します。

例えば、クエリ結果内の特定の行では、列名 avg_mem_cached_1h が列名および列に関連付けられた値として使用され、avg_mem_cached_1h が単一測定レコードの測定値として使用されます。 TargetMeasureName を使用して、ターゲットテーブルで別のメジャー名を使用することもできます。例えば、列 sum_1h の値の場合、マッピングはターゲットテーブルのメジャー名として total_disk_io_writes_1h を使用するように指定します。列の値が の場合NULL、対応するメジャーは無視されます。

```
{
    "Name" : "SingleMeasureMappingWithoutMeasureNameColumnInQueryResult",
    "QueryString" : "SELECT region, bin(time, 1h) as hour, AVG(CASE WHEN measure_name
 = 'memory_cached' THEN measure_value::double ELSE NULL END) as avg_mem_cached_1h,
 AVG(CASE WHEN measure_name = 'disk_used' THEN measure_value::double ELSE NULL END) as
 avg_disk_used_1h, AVG(CASE WHEN measure_name = 'disk_free' THEN measure_value::double
 ELSE NULL END) as avg_disk_free_1h, MIN(CASE WHEN measure_name = 'memory_free' THEN
 measure_value::double ELSE NULL END) as min_mem_free_1h, MIN(CASE WHEN measure_name =
 'cpu_idle' THEN measure_value::double ELSE NULL END) as min_cpu_idle_1h, SUM(CASE WHEN
 measure_name = 'disk_io_writes' THEN measure_value::double ELSE NULL END) as sum_1h,
 MAX(CASE WHEN measure_name = 'memory_used' THEN measure_value::double ELSE NULL END)
 as max_mem_used_1h, MAX(CASE WHEN measure_name = 'cpu_user' THEN measure_value::double
 ELSE NULL END) as max_cpu_user_1h, MAX(CASE WHEN measure_name = 'cpu_system' THEN
 measure_value::double ELSE NULL END) as max_cpu_system_1h FROM raw_data.devops WHERE
 time BETWEEN bin(@scheduled_runtime, 1h) - 14h AND bin(@scheduled_runtime, 1h) - 2h
 AND measure_name IN ('memory_free', 'memory_used', 'memory_cached', 'disk_io_writes',
 'disk_used', 'disk_free', 'cpu_user', 'cpu_system', 'cpu_idle') GROUP BY region,
 bin(time, 1h)",
    "ScheduleConfiguration" : {
        "ScheduleExpression" : "cron(0 0/1 * * ? *)"
    },
    "NotificationConfiguration" : {
        "SnsConfiguration" : {
            "TopicArn" : "******"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******",
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName" : "derived",
            "TableName" : "dashboard_metrics_1h_agg_5",
            "TimeColumn" : "hour",
            "DimensionMappings" : [
                {
                    "Name": "region",
                    "DimensionValueType" : "VARCHAR"
```

```
                }
            ],
            "MixedMeasureMappings" : [
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "avg_mem_cached_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "avg_disk_used_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "avg_disk_free_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "min_mem_free_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "min_cpu_idle_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "sum_1h",
                    "TargetMeasureName" : "total_disk_io_writes_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_mem_used_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_cpu_user_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_cpu_system_1h"
                }
            ]
        }
    },
    "ErrorReportConfiguration": {
```

```
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    }
}
```

このスケジュールされたクエリがマテリアライズされた後の宛先テーブルのスキーマを次に示します。ご覧のように、ターゲットテーブルには、単一測定値が double 型のレコードが保存されています。同様に、テーブルのメジャースキーマには 9 つのメジャー名が表示されます。また、sum_1h という名前のマッピングが total_disk_io_writes_1h に変更されたため、メジャー名 total_disk_io_writes_1h が存在することに注意してください。

列	タイプ	LiveAnalytics 属性タイプのタイムストリーム
region	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
時系	timestamp	TIMESTAMP
measure_value::double	double	MEASURE_VALUE

以下は、SHOWMEASURESクエリで取得した対応する測定値です。

measure_name	data_type	ディメンション
avg_disk_free_1h	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
avg_disk_used_1h	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
avg_mem_cached_1h	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]

measure_name	data_type	ディメンション
最大_cpu_system_1h	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
max_cpu_user_1h	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
max_mem_used_1h	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
min_cpu_idle_1h	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
最小_mem_free_1h	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
total-disk-io-writes	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]

スケジュールされたクエリ通知メッセージ

このセクションでは、スケジュールされたクエリの状態を作成、削除、実行、または更新 LiveAnalytics するときに Timestream によって に送信されるメッセージについて説明します。

通知メッセージ名	構造	説明
CreatingNotificationMessage	`CreatingNotificationMessage { String arn; NotificationType type; }`	この通知メッセージは、 のレスポンスを送信する前に送信されますCreateScheduledQuery 。スケジュールされたクエリは、この通知の送信後に有効になります。 arn - 作成中のスケジュールされたクエリARNの 。

通知メッセージ名	構造	説明
		タイプ - SCHEDULED _QUERY_CREATING
UpdateNotificationMessage	`UpdateNotification Message { String arn; NotificationType type; QueryState state; }`	この通知メッセージは、スケジュールされたクエリが更新されると送信されます。の Timestream は、次のような回復不可能なエラーが発生した場合に備えて、スケジュールされたクエリを自動的に無効に LiveAnalytics できます。 • AssumeRole 失敗 • カスタマーマネージドKMSキーが指定されKMSたときに との通信中に発生した 4xx エラー。 • スケジュールされたクエリの実行中に発生した 4xx エラー。 • クエリ結果の取り込み中に発生した 4xx エラー arn - 更新されるスケジュールされたクエリARNの 。 タイプ - SCHEDULED _QUERY_UPDATE 状態 - ENABLEDまたは DISABLED

通知メッセージ名	構造	説明
DeleteNotificationMessage	`DeletionNotificationMessage { String arn; NotificationType type; }`	この通知メッセージは、スケジュールされたクエリが削除されたときに送信されます。 arn - 作成中のスケジュールされたクエリARNの 。 タイプ - SCHEDULED_QUERY_DELETED

通知メッセージ名	構造	説明

SuccessNotificationMessage	SuccessNotificationMessage {
	    NotificationType type;
	    String arn;
	    Date nextInvocationEpochSecond;
	    ScheduledQueryRunSummary runSummary;
	}
	
	ScheduledQueryRunSummary {
	    Date invocationTime;
	    Date triggerTime;
	    String runStatus;
	    ExecutionStats executionstats;
	    ErrorReportLocation errorReportLocation;
	    String failureReason;
	}
	
	ExecutionStats {
	    Long bytesMetered;
	    Long dataWrites;
	    Long queryResultRows;
	    Long recordsIngested;
	    Long executionTimeInMillis;
	}
	
	ErrorReportLocation {
		この通知メッセージは、スケジュールされたクエリが実行され、結果が正常に取り込まれた後に送信されます。
		
		ARN - 削除されるスケジュールされたクエリARNの 。
		
		NotificationType - AUTO_TRIGGER_SUCCESS または MANUAL_TRIGGER_SUCCESS。
		
		nextInvocationEpoch2 番目 - 次回 の LiveAnalytics Timestream がスケジュールされたクエリを実行します。
		
		runSummary - スケジュールされたクエリ実行に関する情報。

通知メッセージ名	構造	説明

	S3ReportLocation s3ReportLocation;
	}
	
	S3ReportLocation {
	String bucketName;
	String objectKey;
	}	

通知メッセージ名	構造	説明

FailureNotificationMessage	FailureNotificationMessage {
	    NotificationType type;
	    String arn;
	    ScheduledQueryRunSummary runSummary;
	}
	
	ScheduledQueryRunSummary {
	    Date invocationTime;
	    Date triggerTime;
	    String runStatus;
	    ExecutionStats executionstats;
	    ErrorReportLocation errorReportLocation;
	    String failureReason;
	}
	
	ExecutionStats {
	    Long bytesMetered;
	    Long dataWrites;
	    Long queryResultRows;
	    Long recordsIngested;
	    Long executionTimeInMillis;
	}
	
	ErrorReportLocation {
	    S3ReportLocation s3ReportLocation;
	}
		この通知メッセージは、スケジュールされたクエリの実行中に障害が発生した場合、またはクエリ結果を取り込むときに送信されます。
		
		arn - 実行中のスケジュールされたクエリARNの 。
		
		タイプ - AUTO_TRIGGER_FAILURE または MANUAL_TRIGGER_FAILURE。
		
		runSummary - スケジュールされたクエリ実行に関する情報。

通知メッセージ名	構造	説明
	`S3ReportLocation { String bucketName; String objectKey; }`	

スケジュールされたクエリエラーレポート

このセクションでは、スケジュールされたクエリの実行によってエラーが発生した場合に Timestream LiveAnalytics によって生成されるエラーレポートの場所、形式、および理由について説明します。

トピック

- スケジュールされたクエリエラーレポートの理由
- スケジュールされたクエリエラーレポートの場所
- スケジュールされたクエリエラーレポートの形式
- スケジュールされたクエリエラータイプ
- スケジュールされたクエリエラーレポートの例

スケジュールされたクエリエラーレポートの理由

エラーレポートは、回復可能なエラーに対して生成されます。回復不可能なエラーについては、エラーレポートは生成されません。のタイムストリーム LiveAnalytics は、回復不可能なエラーが発生した場合に、スケジュールされたクエリを自動的に無効にできます。具体的には次のとおりです。

- `AssumeRole` 失敗
- カスタマーマネージドKMSキーが指定されKMSたときに との通信中に発生した 4xx エラー
- スケジュールされたクエリの実行時に発生した 4xx エラー
- クエリ結果の取り込み中に発生した 4xx エラー

回復不可能なエラーの場合、 の Timestream は回復不可能なエラーメッセージとともに障害通知 LiveAnalytics を送信します。また、スケジュールされたクエリが無効になっていることを示す更新通知も送信されます。

スケジュールされたクエリエラーレポートの場所

スケジュールされたクエリエラーレポートの場所には、次の命名規則があります。

```
s3://customer-bucket/customer-prefix/
```

スケジュールされたクエリの例を次に示しますARN。

```
arn:aws:timestream:us-east-1:000000000000:scheduled-query/test-query-hd734tegrgfd
```

```
s3://customer-bucket/customer-prefix/test-query-hd734tegrgfd/<InvocationTime>/<Auto or
 Manual>/<Actual Trigger Time>
```

Auto は、 LiveAnalytics と の Timestream によって自動的にスケジュールされたスケジュールされたクエリを示します。*Manual* は、Amazon Timestream for Query の `ExecuteScheduledQuery` API アクションを介してユーザーが手動でトリガーしたスケジュールされた LiveAnalytics クエリを示します。の詳細についてはExecuteScheduledQuery、「」を参照してくださいExecuteScheduledQuery。

スケジュールされたクエリエラーレポートの形式

エラーレポートには次のJSON形式があります。

```
{
    "reportId": <String>,            // A unique string ID for all error reports
 belonging to a particular scheduled query run
    "errors": [ <Error>, ... ],      // One or more errors
}
```

スケジュールされたクエリエラータイプ

`Error` オブジェクトは、次の 3 つのタイプのいずれかになります。

- レコード取り込みエラー

```
{
```

```
    "reason": <String>,              // The error message String
    "records": [ <Record>, ... ],    // One or more rejected records )
}
```

- 行の解析と検証エラー

```
{
    "reason": <String>,        // The error message String
    "rawLine": <String>,       // [Optional] The raw line String that is being parsed
 into record(s) to be ingested. This line has encountered the above-mentioned parse
 error.
}
```

- 一般的なエラー

```
{
    "reason": <String>,        // The error message
}
```

スケジュールされたクエリエラーレポートの例

以下は、取り込みエラーが原因で生成されたエラーレポートの例です。

```
{
    "reportId": "C9494AABE012D1FBC162A67EA2C18255",
    "errors": [
        {
            "reason": "The record timestamp is outside the time range
 [2021-11-12T14:18:13.354Z, 2021-11-12T16:58:13.354Z) of the memory store.",
            "records": [
                {
                    "dimensions": [
                        {
                            "name": "dim0",
                            "value": "d0_1",
                            "dimensionValueType": null
                        },
                        {
                            "name": "dim1",
                            "value": "d1_1",
                            "dimensionValueType": null
                        }
```

```
                    ],
                    "measureName": "random_measure_value",
                    "measureValue": "3.141592653589793",
                    "measureValues": null,
                    "measureValueType": "DOUBLE",
                    "time": "1637166175635000000",
                    "timeUnit": "NANOSECONDS",
                    "version": null
                },
                {
                    "dimensions": [
                        {
                            "name": "dim0",
                            "value": "d0_2",
                            "dimensionValueType": null
                        },
                        {
                            "name": "dim1",
                            "value": "d1_2",
                            "dimensionValueType": null
                        }
                    ],
                    "measureName": "random_measure_value",
                    "measureValue": "6.283185307179586",
                    "measureValues": null,
                    "measureValueType": "DOUBLE",
                    "time": "1637166175636000000",
                    "timeUnit": "NANOSECONDS",
                    "version": null
                },
                {
                    "dimensions": [
                        {
                            "name": "dim0",
                            "value": "d0_3",
                            "dimensionValueType": null
                        },
                        {
                            "name": "dim1",
                            "value": "d1_3",
                            "dimensionValueType": null
                        }
                    ],
                    "measureName": "random_measure_value",
```

```
                    "measureValue": "9.42477796076938",
                    "measureValues": null,
                    "measureValueType": "DOUBLE",
                    "time": "1637166175637000000",
                    "timeUnit": "NANOSECONDS",
                    "version": null
                },
                {
                    "dimensions": [
                        {
                            "name": "dim0",
                            "value": "d0_4",
                            "dimensionValueType": null
                        },
                        {
                            "name": "dim1",
                            "value": "d1_4",
                            "dimensionValueType": null
                        }
                    ],
                    "measureName": "random_measure_value",
                    "measureValue": "12.566370614359172",
                    "measureValues": null,
                    "measureValueType": "DOUBLE",
                    "time": "1637166175638000000",
                    "timeUnit": "NANOSECONDS",
                    "version": null
                }
            ]
        }
    ]
}
```

スケジュールされたクエリパターンと例

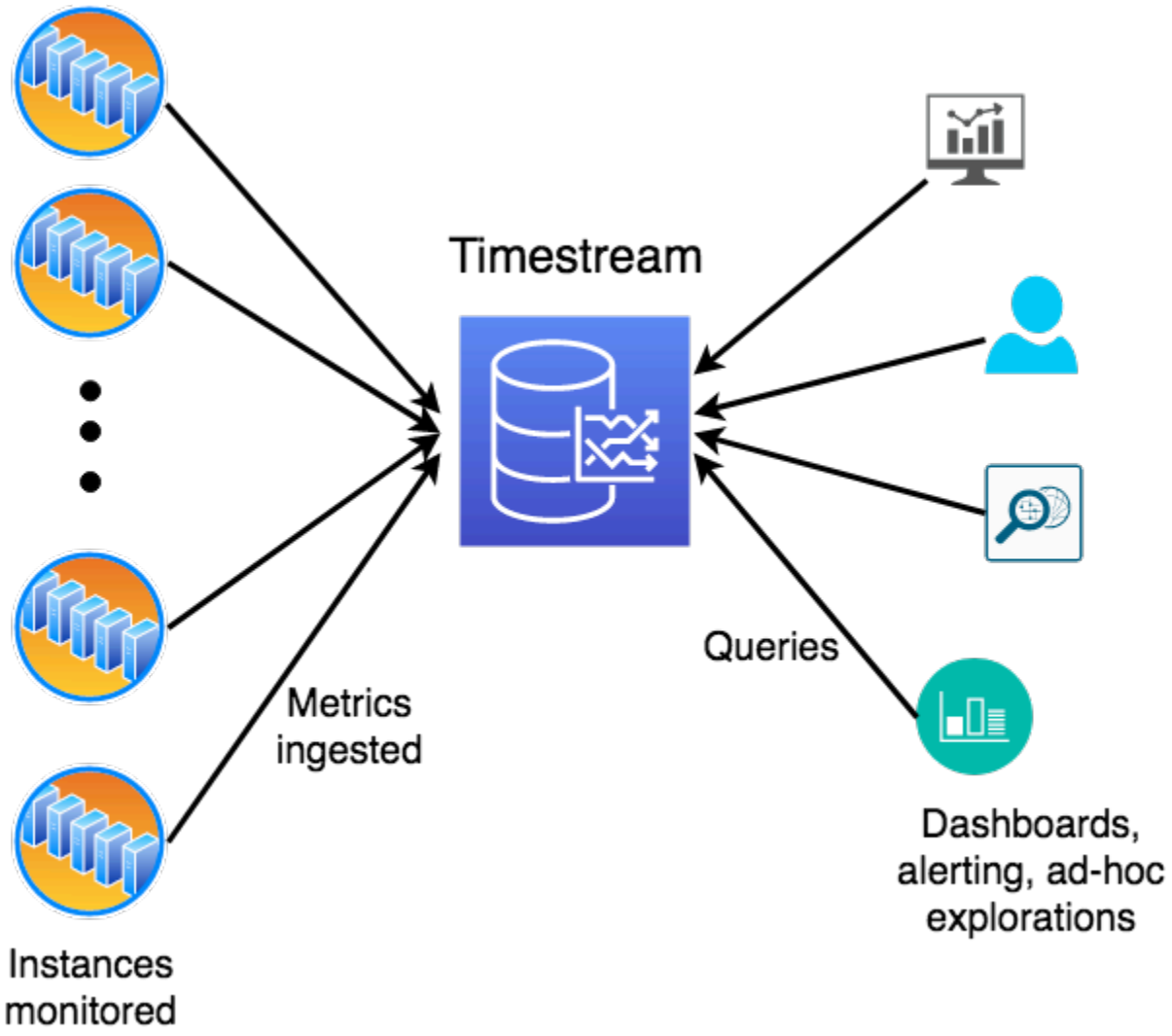
このセクションでは、スケジュールされたクエリの使用パターンと end-to-end例について説明します。

トピック

- スケジュールされたクエリのサンプルスキーマ
- スケジュールされたクエリパターン
- スケジュールされたクエリの例

スケジュールされたクエリのサンプルスキーマ

この例では、大規模なサーバーフリートからの DevOps シナリオモニタリングメトリクスを模倣したサンプルアプリケーションを使用します。ユーザーは、異常なリソース使用状況を警告し、フリートの集計動作と使用率に関するダッシュボードを作成し、最近および過去のデータに対して高度な分析を実行して相関関係を見つけたいと考えています。次の図は、モニタリング対象のインスタンスのセットが の Timestream にメトリクスを出力するセットアップを示しています LiveAnalytics。別の同時ユーザーのセットは、クエリと取り込みが並行して実行されるアラート、ダッシュボード、またはアドホック分析のクエリを発行します。



モニタリング対象のアプリケーションは、世界中の複数のリージョンにデプロイされる高度にスケールアウトされたサービスとしてモデル化されています。各リージョンは、リージョン内のインフラストラクチャの観点から分離レベルがあるセルと呼ばれるいくつかのスケーリングユニットにさらに細分化されます。各セルはさらにサイロに細分化され、ソフトウェア分離のレベルを表します。各サ

イロには、サービスの 1 つの分離されたインスタンスを構成する 5 つのマイクロサービスがあります。各マイクロサービスには、異なるインスタンスタイプと OS バージョンを持つ複数のサーバーがあり、3 つのアベイラビリティーゾーンにデプロイされます。メトリクスを発行するサーバーを識別するこれらの属性は、 の Timestream でディメンションとしてモデル化されます LiveAnalytics。このアーキテクチャでは、ディメンションの階層 (リージョン、セル、サイロ、microservice_name など) と、階層を横断するその他のディメンション (インスタンスタイプ、アベイラビリティーゾーンなど) があります。

アプリケーションは、さまざまなメトリクス (cpu_user や memory_free など) やイベント (task_completed や gc_reclaimed など) を出力します。各メトリクスまたはイベントは、それを出力するサーバーを一意に識別する 8 つのディメンション (リージョンやセルなど) に関連付けられます。データは、メジャー名メトリクスを含むマルチメジャーレコードに一緒に保存された 20 個のメトリクスで書き込まれ、5 つのイベントはすべて、メジャー名イベントを含む別のマルチメジャーレコードに一緒に保存されます。データモデル、スキーマ、およびデータ生成は、オープンソースのデータジェネレーター にあります。スキーマとデータディストリビューションに加えて、データジェネレーターは、複数のライターを使用してデータを並列に取り込む例を提供し、 の Timestream の取り込みスケーリングを使用して 1 秒あたり数百万回の測定値を取り LiveAnalytics 込みます。以下に、スキーマ (テーブルスキーマとメジャースキーマ) とデータセットのサンプルデータを示します。

トピック

- マルチメジャーレコード
- 単一測定レコード

マルチメジャーレコード

テーブルスキーマ

マルチメジャーレコードを使用してデータが取り込まれた後のテーブルスキーマを次に示します。これはDESCRIBEクエリの出力です。データがデータベース raw_data とテーブル devops に取り込まれると仮定すると、以下がクエリです。

```
DESCRIBE "raw_data"."devops"
```

列	タイプ	LiveAnalytics 属性タイプのタイムストリーム
availability_zone	varchar	DIMENSION
microservice_name	varchar	DIMENSION
instance_name	varchar	DIMENSION
process_name	varchar	DIMENSION
os_version	varchar	DIMENSION
jdk_version	varchar	DIMENSION
セル	varchar	DIMENSION
region	varchar	DIMENSION
サイロ	varchar	DIMENSION
instance_type	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
時系	timestamp	TIMESTAMP
memory_free	double	MULTI
cpu_steal	double	MULTI
cpu_iowait	double	MULTI
cpu_user	double	MULTI
memory_cached	double	MULTI
disk_io_reads	double	MULTI
cpu_hi	double	MULTI
latency_per_read	double	MULTI

列	タイプ	LiveAnalytics 属性タイプのタイムストリーム
network_bytes_out	double	MULTI
cpu_idle	double	MULTI
disk_free	double	MULTI
memory_used	double	MULTI
cpu_system	double	MULTI
file_descriptors_in_use	double	MULTI
disk_used	double	MULTI
cpu_nice	double	MULTI
disk_io_writes	double	MULTI
cpu_si	double	MULTI
latency_per_write	double	MULTI
network_bytes_in	double	MULTI
task_end_state	varchar	MULTI
gc_pause	double	MULTI
task_completed	bigint	MULTI
gc_reclaimed	double	MULTI

スキーマの測定

以下は、SHOWMEASURESクエリによって返されるメジャースキーマです。

```
SHOW MEASURES FROM "raw_data"."devops"
```

measure_name	data_type	ディメンション
events	マルチ	[{「data_type」、「dim ension_name」、「data _type」、「varchar」、 「dimension_name」、 「microservice_name」、 「data_type」、「varch ar」、「dimension_nam e」、「data_type」、「va rchar_name」、「data_ type」、「dimension_n ame」、「data_type」、」 、「data_type」、」、「di mension_name_name」 、」、「data_type」
メトリクス	マルチ	[{「data_type」varch ar「dimension_name」 availability_zone」,{「data_t ype」varchar「dimens ion_name」microserv ice_name」,{「data_t ype」varchar「dimens ion_name」instance_name」, {「data_type」varchar「d imension_name」cell 」,{「data_type」varc har「dimension_name _name」silo」,{「data _type_type」

データの例

re	セ ル	サ イ ロ	a ity	m ic	ir n	ir ty	o n	p a	jo o	m a	時 間	c	c m	c	c	c t	c	c	c	m e	m cl	d e	la e	d ri	la e	d	d	n yt	n yt	fil ri n	m e	ta st	g	ta le	gscencompla mled
u e	u e - c	u e - c s il	u e -	a	i- z k a u e - c s il 0 z m	m e	A			> ト リ ク ス	1 1 1.	6.	0	3	0	0	0	0	0	5	8	5.	9	3	2	6	2	8	4	3.	5				
u e	u e - c	u e - c s il	u e -	a	i- z k a u e - c s il 0 z m	m e	A			> ト リ ク ス	1 1 1.	5	0	3	0	0	0	0	0	9	3	5	3	2	9	5.	3	7	5	6	2				
u e	u e - c	u e - c	u e -	a	i- z k a u	m e	A			> ト リ ク ス	1 1 1.	4	0	4	0	0	0	0	0	9	4	5	3.	8	3	7.	6	8	5	4	8				

re	セ ル	サ イ ロ	a ity	m ic	in n	in ty	o n	p a	jo o	m a	時 間	c	c m	c	c	c t	c	c	c	m e	m cl	d e	la e	d ri	la e	d	d	n y	n y	fil ri n	m e	ta st	g	ta le	g led
		s il			e - c s il 0 z m																														
u e	u e - c	u e - c s il	u e -1	a	i- z k a u e - c s il 0 z m	m e	A			メ ト リ ク ス	1 1 1.	3	0	5	0	0	0	0	0	4	7	2	4	4	3	8	6	2	1	2	1				

re	セ ル	セ イ ロ	a ity	m ic	ir n	ir ty	o n	p a	jo o	m a	時 間	c	c m	c	c	c t	c	c	c	m e	m cl	d e	la e	d ri	la e	d	d	n y	n y	fil ri n	m e	ta st	g	ta le	g med scomp led
u e	u e - c	u e - c s il	u e -1	a	i- z k a u e - c s il 0 z m	m e	A			> ト リ ク ス	1 1 1.	5	0	3	0	0	0	0	0	5	3	8	5	7	1	8	6	7	6	5	3				
u e	u e - c	u e - c s il	u e -1	a	i- z k a u e - c s il 0 z m			h g	J	e	1 1 1.																				7	S T U	8	3	6 3/8I ES

re	ヒJl	サイD	at ity	mic	in n	in ty	o n	pa	jo o	m a	時間	c	c m	c	c	c t	c	c	c	m e	m cl	d e	la e	d ri	la e	d	d	n yt	n yt	fil rin	m e	ta st	g	ta le	gsecoml	scenla
us e	us e- co	us e- cs ilo	us e- -1	at	i- za k- au us e- cs ilo 0 zo m			ho g	JI	e	1: 1 1:																			7	SI T	2	4	73	45	W JL
us e	us e- co	us e- cs ilo	us e- -1	at	i- za k- au us e- cs ilo 0 zo m			ho g	JI	e	1: 1 1:																			6	SI T	6	2	72	2.3	W JL

re	ナル	サイロ	ait	mic	irn	irty	on	pa	joo	ma	時間	c	cm	c	c	ct	c	c	c	me	mcl	de	lae	dri	lae	d	d	nyt	nyt	firin	me	tast	g	tale	gsecmpa med
ue	ue-c	ue-csil	ue-1	a	i-zkaue-csil0zm			hg	Jl	e	111.																				2	SIT	5	1	99_W JL
ue	ue-c	ue-csil	ue-1	a	i-zkaue-csil0zm			hg	Jl	e	111.																				3	STU	7	2	82.9I RES

単一測定レコード

の Timestream LiveAnalytics では、時系列レコードごとに 1 つのメジャーでデータを取り込むこともできます。単一メジャーレコードを使用して取り込む場合のスキーマの詳細を以下に示します。

テーブルスキーマ

マルチメジャーレコードを使用してデータが取り込まれた後のテーブルスキーマを以下に示します。これはDESCRIBEクエリの出力です。データがデータベース raw_data とテーブル devops に取り込まれると仮定すると、以下がクエリです。

```
DESCRIBE "raw_data"."devops_single"
```

列	タイプ	LiveAnalytics 属性タイプのタイムストリーム
availability_zone	varchar	DIMENSION
microservice_name	varchar	DIMENSION
instance_name	varchar	DIMENSION
process_name	varchar	DIMENSION
os_version	varchar	DIMENSION
jdk_version	varchar	DIMENSION
セル	varchar	DIMENSION
region	varchar	DIMENSION
サイロ	varchar	DIMENSION
instance_type	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
時系	timestamp	TIMESTAMP
measure_value::double	double	MEASURE_VALUE
measure_value::bigint	bigint	MEASURE_VALUE
measure_value::varchar	varchar	MEASURE_VALUE

スキーマの測定

以下は、SHOWMEASURESクエリによって返されるメジャースキーマです。

```
SHOW MEASURES FROM "raw_data"."devops_single"
```

measure_name	data_type	ディメンション
cpu_hi	double	[{'dimension_name': 'availability_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_type': 'varchar'}, {'dimension_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'os_version', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar': 'varchar' }, {'dimension_name': '
cpu_idle	double	[{'dimension_name': 'availability_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_type': 'varchar'}, {'dimension_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'os_version', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar': 'varchar' }, {'dimension_name': '
cpu_iowait	double	[{'dimension_name': 'availability_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ

measure_name	data_type	ディメンション
		e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'os_versi on', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar': 'varchar' }, {'dimension_name': '
cpu_nice	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'os_versi on', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar': 'varchar' }, {'dimension_name': '
cpu_si	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'os_versi on', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar': 'varchar' }, {'dimension_name': '

measure_name	data_type	ディメンション
cpu_steal	double	[{'dimension_name': 'availability_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_type': 'varchar'}, {'dimension_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'os_version', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar': 'varchar' }, {'dimension_name': '
cpu_system	double	[{'dimension_name': 'availability_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_type': 'varchar'}, {'dimension_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'os_version', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar': 'varchar' }, {'dimension_name': '

measure_name	data_type	ディメンション
cpu_user	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'os_versi on', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar': 'varchar' }, {'dimension_name': '
disk_free	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'os_versi on', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar': 'varchar' }, {'dimension_name': '

measure_name	data_type	ディメンション
disk_io_reads	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'os_versi on', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar': 'varchar' }, {'dimension_name': '
disk_io_writes	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'os_versi on', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar': 'varchar' }, {'dimension_name': '

measure_name	data_type	ディメンション
disk_used	double	[{'dimension_name': 'availability_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_type': 'varchar'}, {'dimension_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'os_version', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar': 'varchar' }, {'dimension_name': '
file_descriptors_in_use	double	[{'dimension_name': 'availability_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_type': 'varchar'}, {'dimension_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'os_version', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar': 'varchar' }, {'dimension_name': '

measure_name	data_type	ディメンション
gc_pause	double	[{'dimension_name': 'availability_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_type': 'varchar'}, {'dimension_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'process_name', 'data_type': 'varchar'}, {'dimension_name': 'jdk_version', 'data_type': 'varchar'}, {'dimension_name': 'data_type':
gc_reclaimed	double	[{'dimension_name': 'availability_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_type': 'varchar'}, {'dimension_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'process_name', 'data_type': 'varchar'}, {'dimension_name': 'jdk_version', 'data_type': 'varchar'}, {'dimension_name': 'data_type':

measure_name	data_type	ディメンション
latency_per_read	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'os_versi on', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar': 'varchar' }, {'dimension_name': '
latency_per_write	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'os_versi on', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar': 'varchar' }, {'dimension_name': '

measure_name	data_type	ディメンション
memory_cached	double	[{'dimension_name': 'availability_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_type': 'varchar'}, {'dimension_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'os_version', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar': 'varchar' }, {'dimension_name': '

measure_name	data_type	ディメンション
memory_free	double	[{'dimension_name': 'availability_zone', 'data_type': 'varchar'}、 {'dimension_name': 'microservice_name', 'data_type': 'varchar'}、 {'dimension_name': 'instance_name', 'data_type': 'varchar'}、 {'dimension_name': 'process_name', 'data_type': 'varchar'}、 {'dimension_name': 'os_version', 'data_type': 'varchar'}、 {'dimension_name': 'jdk_version', 'data_type': 'varchar'}、 {'dimension_name': 「セル」 , 'data_type': 'varchar'}、 {'dimension_name': 「リージョン」、 'data_type': 'varchar'}、 {'dimension_name': 'サイロ', 'data_type': 'varchar'}、 {'dimension_name': 'instance_type', 'data_type': 'varchar'}]

measure_name	data_type	ディメンション
memory_used	double	[{'dimension_name': 'availability_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_type': 'varchar'}, {'dimension_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'os_version', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar': 'varchar' }, {'dimension_name': '
network_bytes_in	double	[{'dimension_name': 'availability_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_type': 'varchar'}, {'dimension_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'os_version', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar': 'varchar' }, {'dimension_name': '

measure_name	data_type	ディメンション
network_bytes_out	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'os_versi on', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar': 'varchar' }, {'dimension_name': '
task_completed	bigint	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'process_ name', 'data_type': 'varchar'}, {'dimension_name': 'jdk_vers ion', 'data_type': 'varchar'}, {'dimension_name': 'data_typ e':

measure_name	data_type	ディメンション
task_end_state	varchar	[{'dimension_name': 'availability_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_type': 'varchar'}, {'dimension_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'process_name', 'data_type': 'varchar'}, {'dimension_name': 'jdk_version', 'data_type': 'varchar'}, {'dimension_name': 'data_type':

データの例

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	セル	regio	サイロ	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west--1	ヘラクレス	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200000mazocom		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	cpu_l	34:57	0.871		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	セル	regio	サイロ	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	ヘラクレス	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_i	34:57	3.462		
eu- west- -1	ヘラクレス	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_i t	34:57	0.102		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	セル	regio	サイロ	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west--1	ヘラクレス	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200000mazocom		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	cpu_r	34:57	0.630		
eu-west--1	ヘラクレス	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200000mazocom		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	cpu_s	34:57	0.164		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	セル	regio	サイロ	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	ヘラクレス	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_s	34:57	0.107		
eu- west- -1	ヘラクレス	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_s m	34:57	0.457		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	セル	regio	サイロ	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	ヘラクレス	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_u	34:57	94.20		
eu- west- -1	ヘラクレス	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	disk_u	34:57	72.51		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	セル	regio	サイロ	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west--1	ヘラクレス	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200006mazocom		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	disk_ieads	34:57	81.73		
eu-west--1	ヘラクレス	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200006mazocom		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	disk_irites	34:57	77.11		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	セル	regio	サイロ	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west--1	ヘラクレス	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200006mazocom		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	disk_	34:57	89.42		
eu-west--1	ヘラクレス	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200006mazocom		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	file_driptorsn_use	34:57	30.08		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	セル	regio	サイロ	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west--1	ヘラクレス	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200000mazocom	serve		JDK_	eu-west--cell-9	eu-west-	eu-west--cell-9silo-2		gc_pa	34:57	60.28		
eu-west--1	ヘラクレス	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200000mazocom	serve		JDK_	eu-west--cell-9	eu-west-	eu-west--cell-9silo-2		gc_remed	34:57	75.28		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	セル	regio	サイロ	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west--1	ヘラクレス	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200006mazocom		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	latenc er_re	34:57	8.076		
eu-west--1	ヘラクレス	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200006mazocom		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	latenc er_wr	34:57	58.11		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	セル	regio	サイロ	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west--1	ヘラクレス	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200006mazocom		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	memched	34:57	87.56		
eu-west--1	ヘラクレス	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200006mazocom	serve		JDK_	eu-west--cell-9	eu-west-	eu-west--cell-9silo-2		memee	34:57	18.95		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	セル	regio	サイロ	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west--1	ヘラクレス	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200006mazocom		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	memee	34:57	2052年97月		
eu-west--1	ヘラクレス	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200006mazocom		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	memed	34:57	12.37		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	セル	regio	サイロ	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	ヘラクレス	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	netwo ytes_	34:57	31.02		
eu- west- -1	ヘラクレス	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	netwo ytes_	34:57	0.514		

availa ity_zo	micro ice_n	instar name	proce ame	os_ve n	jdk_v on	セル	regio	サイロ	instar type	meas ame	時間	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west--1	ヘラクレス	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200000mazocom	serve		JDK_	eu-west--cell-9	eu-west-	eu-west--cell-9silo-2		task_leted	34:57		69	
eu-west--1	ヘラクレス	i-zaZsvk-hercus-eu-west-1-cell-9-silo-200000mazocom	serve		JDK_	eu-west--cell-9	eu-west-	eu-west--cell-9silo-2		task_state	34:57			SUCCESS_ITH_RESULT

スケジュールされたクエリパターン

このセクションでは、Amazon Timestream for LiveAnalytics Scheduled Queries を使用してダッシュボードを最適化し、より速く、低コストでロードする方法の一般的なパターンをいくつか紹介します。以下の例では、 DevOps アプリケーションシナリオを使用して、アプリケーションシナリオに関係なく、スケジュールされたクエリ全般に適用される主要な概念を示しています。

の Timestream のスケジュールされたクエリ LiveAnalytics では、 の Timestream の全SQL表面領域を使用してクエリを表現できます LiveAnalytics。クエリには、1 つ以上のソーステーブルを含めることができ、 LiveAnalyticsのSQL言語に対して Timestream で許可される集計やその他のクエリを実行してから、 の Timestream の別の宛先テーブルでクエリの結果をマテリアライズできます LiveAnalytics。説明しやすいように、このセクションでは、スケジュールされたクエリのターゲットテーブルを派生テーブル として参照します。

このセクションで取り上げる重要なポイントを次に示します。

- 単純なフリートレベルの集計を使用して、スケジュールされたクエリを定義し、いくつかの基本的な概念を理解する方法について説明します。
- スケジュールされたクエリのターゲット (派生テーブル) の結果をソーステーブルの結果と組み合わせて、スケジュールされたクエリのコストとパフォーマンスのメリットを得る方法。
- スケジュールされたクエリの更新期間を設定する際のトレードオフは何ですか。
- 一部の一般的なシナリオでスケジュールされたクエリを使用する。
  - 特定の日付より前のすべてのインスタンスからの最後のデータポイントを追跡します。
  - ダッシュボードに変数を入力するために使用されるディメンションの個別の値。
- スケジュールされたクエリのコンテキストで到着遅延データを処理する方法。
- 1 回限りの手動実行を使用して、スケジュールされたクエリの自動トリガーで直接カバーされないさまざまなシナリオを処理する方法。

トピック

- 過去の計算のバックフィル

シナリオ

次の例では、 で概説されている DevOps モニタリングシナリオを使用していますスケジュールされたクエリのサンプルスキーマ。

この例では、スケジュールされたクエリ定義を提供します。この定義では、スケジュールされたクエリの実行ステータス通知を受信する場所、スケジュールされたクエリの実行中に発生したエラーのレポートを受信する場所、およびスケジュールされたクエリがオペレーションの実行に使用するIAMロールの適切な設定をプラグインできます。

これらのスケジュールされたクエリは、上記のオプションの入力、ターゲット (または派生) テーブルの作成、 を使用した AWS の実行後に作成できますCLI。例えば、スケジュールされたクエリ定義が ファイル に保存されているとしますscheduled_query_example.json。CLI コマンドを使用してクエリを作成できます。

```
aws timestream-query create-scheduled-query --cli-input-json  file://
scheduled_query_example.json --profile aws_profile --region us-east-1
```

前のコマンドでは、--profile オプションを使用して渡されたプロファイルには、スケジュールされたクエリを作成するための適切なアクセス許可が必要です。ポリシーとアクセス許可の詳細については、「スケジュールされたクエリのアイデンティティベースのポリシー」を参照してください。

シンプルなフリートレベルの集計

この最初の例では、フリートレベルの集計計算の簡単な例を使用してスケジュールされたクエリを操作する際の基本的な概念をいくつか説明します。この例では、以下について説明します。

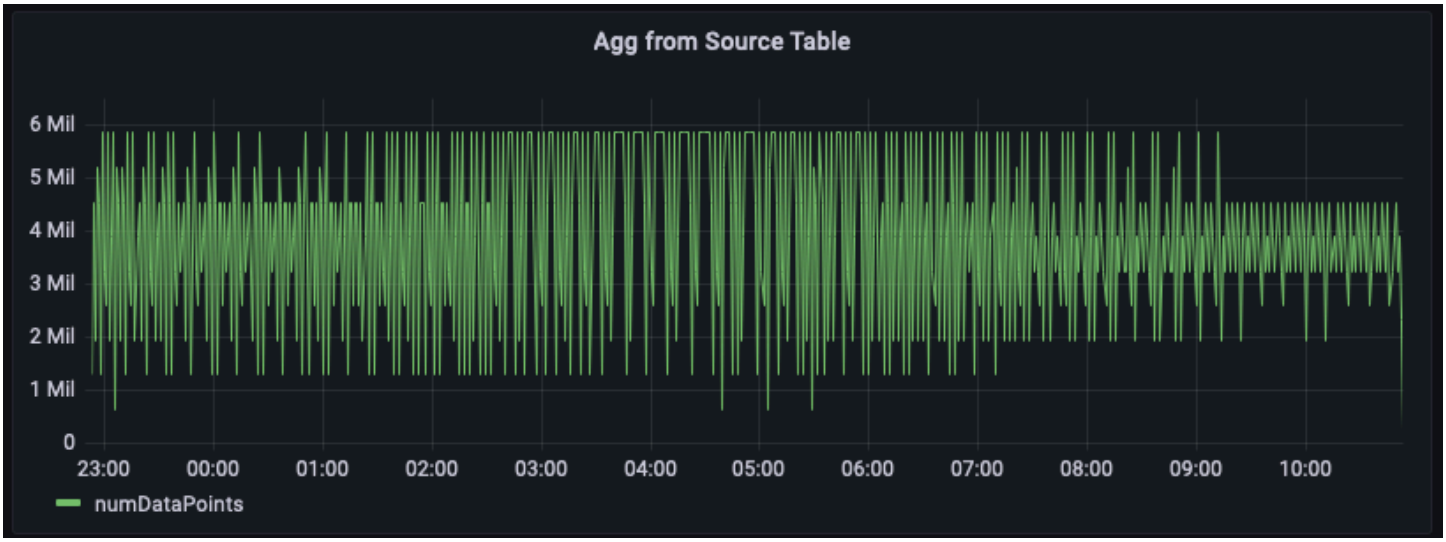
- 集計統計を取得し、スケジュールされたクエリにマッピングするために使用されるダッシュボードクエリを取得する方法。
- の Timestream がスケジュールされたクエリのさまざまなインスタンスの実行 LiveAnalytics を管理する方法。
- スケジュールされたクエリの異なるインスタンスを時間範囲内で重複させる方法と、ターゲットテーブルでデータの正確性を維持する方法により、スケジュールされたクエリの結果を使用してダッシュボードが、生データで計算されたのと同じ集計に一致する結果を確実に得ることができます。
- スケジュールされたクエリの時間範囲と更新頻度を設定する方法。

- スケジュールされたクエリの結果をセルフサービスで追跡して調整し、クエリインスタンスの実行レイテンシーがダッシュボードの更新で許容される遅延の範囲内になるようにする方法。

トピック

ソーステーブルから集約する

この例では、1 分ごとに特定のリージョン内のサーバーによって発行されたメトリクスの数を追跡しています。以下のグラフは、us-east-1 リージョンのこの時系列をプロットした例です。



以下は、生データからこの集計を計算するクエリの例です。リージョン us-east-1 の行をフィルタリングし、20 個のメトリクス (measure_name がメトリクスの場合) または 5 個のイベント (measure_name がイベントの場合) を考慮して 1 分あたりの合計を計算します。この例では、グラフ図は、出力されるメトリクスの数が 1 分あたり 150 万から 600 万の間であることを示しています。この時系列を数時間 (この図では過去 12 時間) プロットする場合、この未加工データに対するクエリは数億行を分析します。

```
WITH grouped_data AS (
    SELECT region, bin(time, 1m) as minute, SUM(CASE WHEN measure_name = 'metrics' THEN
 20 ELSE 5 END) as numDataPoints
```

```
    FROM "raw_data"."devops"
    WHERE time BETWEEN from_milliseconds(1636699996445) AND
 from_milliseconds(1636743196445)
        AND region = 'us-east-1'
    GROUP BY region, measure_name, bin(time, 1m)
)
SELECT minute, SUM(numDataPoints) AS numDataPoints
FROM grouped_data
GROUP BY minute
ORDER BY 1 desc, 2 desc
```

集計を事前計算するスケジュールされたクエリ

ダッシュボードを最適化して、より少ないデータをスキャンすることで、より速くロードし、コストを削減したい場合は、スケジュールされたクエリを使用してこれらの集計を事前計算できます。の Timestream でスケジュールされたクエリ LiveAnalytics を使用すると、これらの事前計算を LiveAnalytics テーブルの別の Timestream でマテリアライズできます。その後、ダッシュボードに使用できます。

スケジュールされたクエリを作成する最初のステップは、事前計算するクエリを特定することです。上記のダッシュボードは、us-east-1 リージョン用に描画されたことに注意してください。ただし、別のユーザーは、us-west-2 または eu-west-1 など、別のリージョンに対して同じ集計を必要とする場合があります。このようなクエリごとにスケジュールされたクエリを作成しないようにするには、リージョンごとの集計を事前に計算し、リージョンごとの集計を LiveAnalytics テーブルの別の Timestream にマテリアライズできます。

以下のクエリは、対応する計算前の例を示しています。ご覧のとおり、これは、生データのクエリで使用される一般的なテーブル式 grouped_data に似ています。ただし、1) リージョン述語を使用しないため、1 つのクエリを使用してすべてのリージョンを事前計算できます。2) パラメータ化された時間述語と、以下で詳しく説明する特別なパラメータ @scheduled_runtime を使用します。

```
SELECT region, bin(time, 1m) as minute,
    SUM(CASE WHEN measure_name = 'metrics' THEN 20 ELSE 5 END) as numDataPoints
FROM raw_data.devops
WHERE time BETWEEN @scheduled_runtime - 10m AND @scheduled_runtime + 1m
GROUP BY bin(time, 1m), region
```

上記のクエリは、次の仕様を使用してスケジュールされたクエリに変換できます。スケジュールされたクエリには、ユーザーフレンドリーなニーモニックである名前が割り当てられます。次に QueryString、cron 式 ScheduleConfigurationである である が含まれます。クエリ結果を の

Timestream の送信先テーブルに TargetConfiguration マッピングする を指定します LiveAnalytics。最後に、クエリの個々の実行の通知 NotificationConfigurationが送信される や、クエリにエラーが発生した場合に ErrorReportConfiguration レポートが書き込まれる 、スケジュールされたクエリのオペレーションを実行するために使用されるロールである など ScheduledQueryExecutionRoleArn、他の多くの設定を指定します。

```
{
    "Name": "MultiPT5mPerMinutePerRegionMeasureCount",
    "QueryString": "SELECT region, bin(time, 1m) as minute, SUM(CASE WHEN measure_name
 = 'metrics' THEN 20 ELSE 5 END) as numDataPoints FROM raw_data.devops WHERE time
 BETWEEN @scheduled_runtime - 10m AND @scheduled_runtime + 1m GROUP BY bin(time, 1m),
 region",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/5 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "per_minute_aggs_pt5m",
            "TimeColumn": "minute",
            "DimensionMappings": [
                {
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "numDataPoints",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "numDataPoints",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
```

```
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

この例では、 ScheduleExpression cron(0/5 * * * ? *) は、クエリが 5 分ごとに 1 回、毎日 5、10、15、.. 分に実行されることを意味します。このクエリの特定のインスタンスがトリガーされたときのこれらのタイムスタンプは、クエリで使用される @scheduled_runtime パラメータに変換されます。例えば、2021-12-01 00:00:00 に実行されたこのスケジュールされたクエリのインスタンスを考えてみましょう。このインスタンスでは、クエリの呼び出し時に、@scheduled_runtime パラメータがタイムスタンプ 2021-12-01 00:00:00 に初期化されます。したがって、この特定のインスタンスはタイムスタンプ 2021-12-01 00:00:00 に実行され、時間範囲 2021-11-30 23:50:00 から 2021-12-01 00:01:00 までの分あたりの集計が計算されます。同様に、このクエリの次のインスタンスはタイムスタンプ 2021-12-01 00:05:00 にトリガーされ、その場合、クエリは時間範囲 2021-11-30 23:55:00 から 2021-12-01 00:06:00 までの分あたりの集計を計算します。したがって、@scheduled_runtime パラメータは、クエリの呼び出し時間を使用して、設定された時間範囲の集計を事前計算するスケジュールされたクエリを提供します。

クエリの後続の 2 つのインスタンスは、その時間範囲内で重複することに注意してください。これは、要件に基づいて制御できます。この場合、この重複により、これらのクエリは、この例で到着がわずかに遅延したデータに基づいて集計を更新できます。マテリアライズドクエリが正しいことを確認するために、 の Timestream は、2021-12-01 00:05:00 の LiveAnalytics クエリが 2021-12-01 00:00:00 のクエリが完了した後にのみ実行され、後者のクエリの結果は、新しい値が生成された場合、 を使用して以前にマテリアライズドされた集計を更新できます。例えば、タイムスタンプ 2021-11-30 23:59:00 のデータが 2021-12-01 00:00:00 のクエリの後、 00:05:00 2021-12-01 のクエリの前に到着した場合、2021-12-01 00:05:00 の実行は分 2021-11-30 23:59:00 の集計を再計算し、以前の集計が新しく計算された値で更新されます。スケジュールされたクエリのこれらのセマンティクスに頼って、事前コンピューティングの更新速度と、到着が遅れた一部のデータを正常に処理する方法とのトレードオフを行うことができます。この更新頻度をデータの鮮度でトレードオフする方法と、さらに遅延して到着するデータの集計の更新にどのように対処するか、またはスケジュールされた計算のソースが集計を再計算する必要がある値を更新しているかどうかに関するその他の考慮事項を以下で説明します。

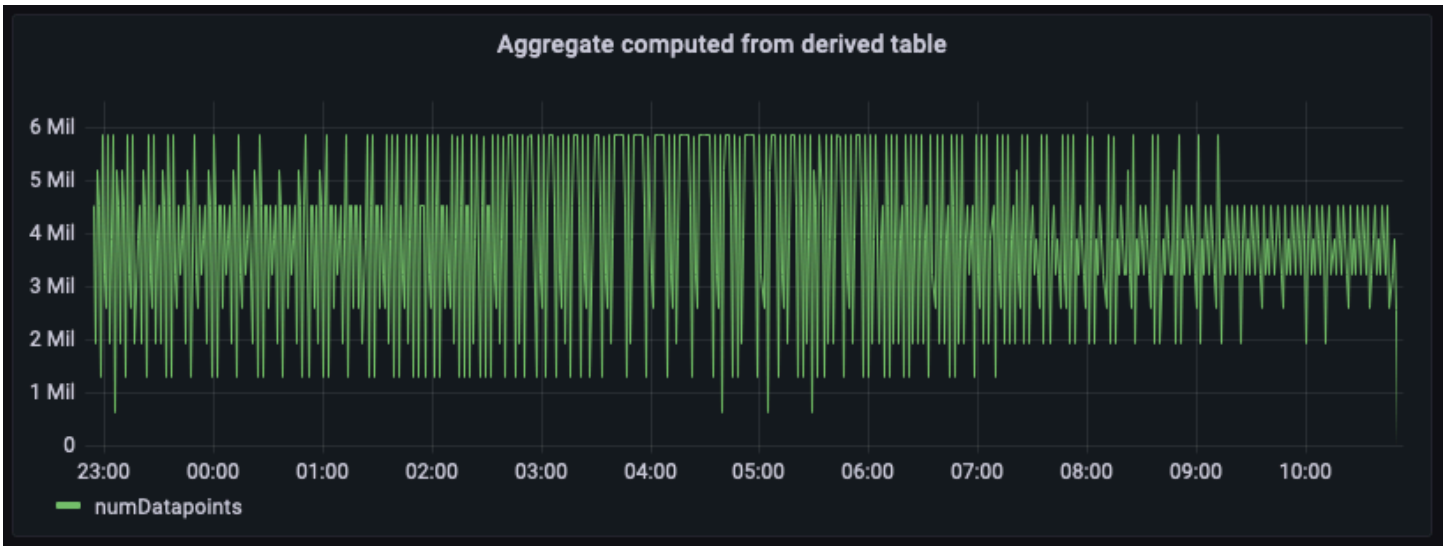
スケジュールされたすべての計算には、 の Timestream がスケジュールされた設定の実行ごとに通知 LiveAnalytics を送信する通知設定があります。のSNSトピックを設定して、呼び出しごとに通知

を受信できます。特定のインスタンスの成功または失敗ステータスに加えて、この計算の実行時間、スキャンされた計算のバイト数、計算が宛先テーブルに書き込んだバイト数など、いくつかの統計もあります。これらの統計を使用して、クエリをさらに調整したり、設定をスケジュールしたり、スケジュールされたクエリの支出を追跡したりできます。注目すべき側面の 1 つは、インスタンスの実行時間です。この例では、スケジュールされた計算は 5 分ごとに実行されるように設定されています。実行時間は、事前計算が使用可能になる遅延を決定します。これにより、ダッシュボードで事前計算されたデータを使用する場合のダッシュボードの遅延も定義されます。さらに、この遅延が更新間隔よりも一貫して長い場合、例えば、5 分ごとに更新するように設定された計算の実行時間が 5 分を超える場合は、ダッシュボードのさらなる遅延を避けるために、計算をより速く実行するように調整することが重要です。

派生テーブルからの集計

スケジュールされたクエリを設定し、集計が事前に計算され、スケジュールされた計算のターゲット設定で指定された LiveAnalytics テーブルの別の Timestream にマテリアライズされたので、そのテーブル内のデータを使用してSQLクエリを書き込み、ダッシュボードを強化できます。以下は、マテリアライズド事前集計を使用して us-east-1 の 1 分あたりのデータポイント数集計を生成するクエリと同等です。

```
SELECT bin(time, 1m) as minute, SUM(numDataPoints) as numDatapoints
FROM "derived"."per_minute_aggs_pt5m"
WHERE time BETWEEN from_milliseconds(1636699996445) AND
 from_milliseconds(1636743196445)
    AND region = 'us-east-1'
GROUP BY bin(time, 1m)
ORDER BY 1 desc
```



前の図は、集計テーブルから計算された集計をプロットしたものです。このパネルを未加工のソースデータから計算されたパネルと比較すると、これらの集計は、スケジュールされた計算に設定した更新間隔とそれを実行する時間によって制御され、数分遅れるものの、正確に一致していることに気付くでしょう。

事前計算されたデータに対するこのクエリは、未加工のソースデータに対して計算された集計と比較して、数桁少ないデータをスキャンします。集約の粒度によっては、この削減により、コストとクエリレイテンシーが 100Xに簡単に削減されます。このスケジュールされた計算を実行するにはコストがかかります。ただし、これらのダッシュボードが更新される頻度と、これらのダッシュボードをロードする同時ユーザー数によっては、これらの事前計算を使用して全体的なコストを大幅に削減できます。また、ダッシュボードのロード時間が 10-100X短縮されます。

ソーステーブルと派生テーブルを集約する

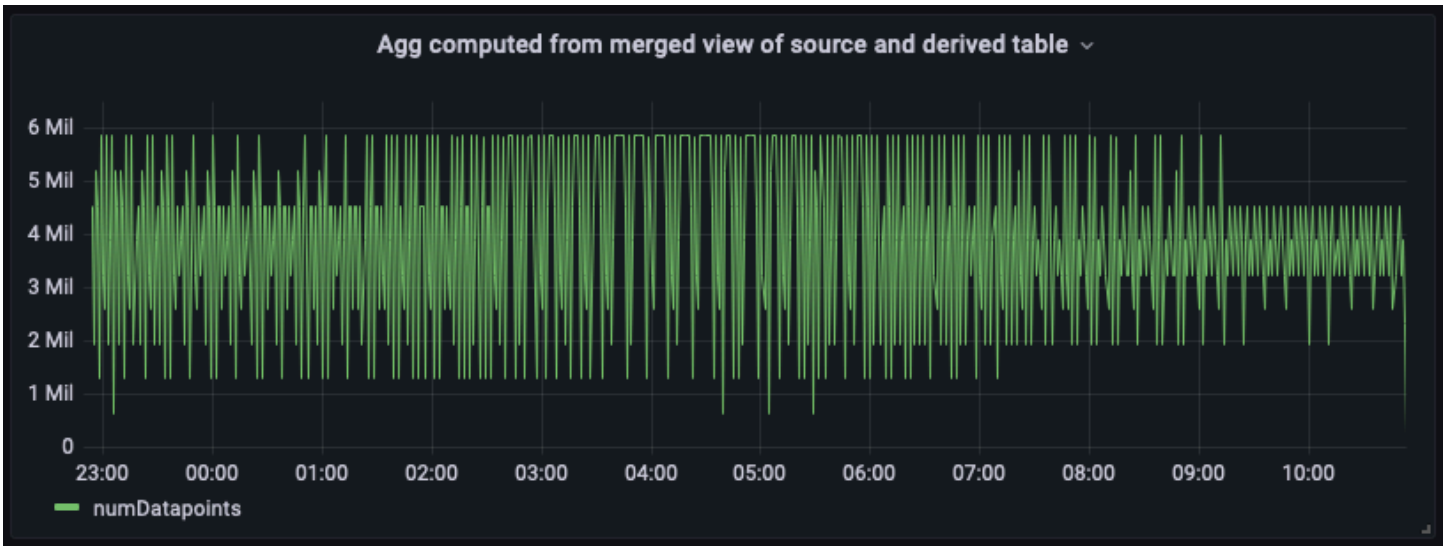
派生テーブルを使用して作成されたダッシュボードには、遅延が発生する可能性があります。アプリケーションシナリオでダッシュボードに最新のデータが必要な場合は、 LiveAnalyticsSQLのサポートに Timestream のパワーと柔軟性を使用して、ソーステーブルの最新データと派生テーブルの履歴集計を組み合わせて、マージされたビューを作成できます。このマージされたビューは、ソースSQLと派生テーブルからの重複しない時間範囲と のユニオンセマンティクスを使用します。次の例では、「derived」.「per_minute_aggs_pt5m」派生テーブルを使用しています。その派生テーブルのスケジュールされた計算は 5 分に 1 回更新されるため (スケジュール式仕様に基づく) 、 以下のこのクエリは、ソーステーブルからの最新の 15 分間のデータを使用します。 および派生テーブルから 15 分経過したデータをすべて結合し、結果を結合して、両方のワールドの長所を持つマージされたビューを作成します。 リアルタイム分析のユースケースを強化するために、派生テーブルから古い事前計算された集計とソーステーブルからの集計の鮮度を読み取ることで、経済性と低レイテンシーを実現します。

このユニオンアプローチは、派生テーブルのクエリのみと比較してクエリレイテンシーがわずかに高く、また、最新の時間間隔を埋めるためにリアルタイムで生データを集約しているため、スキャンされるデータもわずかに高いことに注意してください。ただし、このマージされたビューは、ソーステーブルからの直接の集約と比較して、特にダッシュボードが数日または数週間のデータを表示する場合、大幅に高速かつ安価になります。この例の時間範囲を調整して、アプリケーションの更新ニーズと遅延耐性を調整できます。

```
WITH aggregated_source_data AS (
    SELECT bin(time, 1m) as minute, SUM(CASE WHEN measure_name = 'metrics' THEN 20 ELSE
 5 END) as numDatapoints
    FROM "raw_data"."devops"
```

```
    WHERE time BETWEEN bin(from_milliseconds(1636743196439), 1m) - 15m AND
 from_milliseconds(1636743196439)
        AND region = 'us-east-1'
    GROUP BY bin(time, 1m)
), aggregated_derived_data AS (
    SELECT bin(time, 1m) as minute, SUM(numDataPoints) as numDatapoints
    FROM "derived"."per_minute_aggs_pt5m"
    WHERE time BETWEEN from_milliseconds(1636699996439) AND
 bin(from_milliseconds(1636743196439), 1m) - 15m
        AND region = 'us-east-1'
    GROUP BY bin(time, 1m)
)
SELECT minute, numDatapoints
FROM (
    (
    SELECT *
    FROM aggregated_derived_data
    )
    UNION
    (
    SELECT *
    FROM aggregated_source_data
    )
)
ORDER BY 1 desc
```

以下は、この統合マージビューを持つダッシュボードパネルです。ご覧のように、ダッシュボードは、最も右側の先端に最も up-to-date集計される点を除いて、派生テーブルから計算されたビューとほぼ同じに見えます。



頻繁に更新されるスケジュールされた計算からの集計

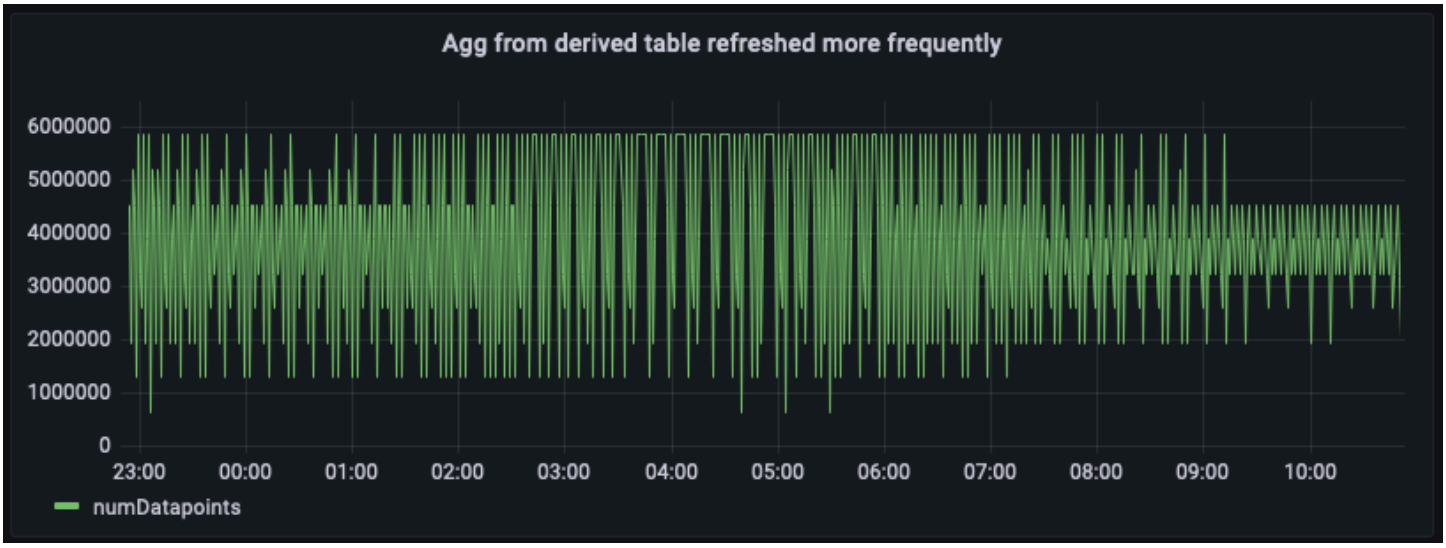
ダッシュボードがロードされる頻度とダッシュボードに必要なレイテンシーに応じて、ダッシュボードで新しい結果を取得する別の方法があります。スケジュールされた計算で集計を更新する頻度を増やすことです。例えば、1 分に 1 回更新される点を除いて、以下は同じスケジュールされた計算の設定です (スケジュールは cron(0/1 * * * * ? *))。この設定では、派生テーブル per_minute_aggs_pt1m は、計算で 5 分に 1 回の更新スケジュールが指定されたシナリオと比較して、はるかに最近の集計になります。

```
{
    "Name": "MultiPT1mPerMinutePerRegionMeasureCount",
    "QueryString": "SELECT region, bin(time, 1m) as minute, SUM(CASE WHEN measure_name
 = 'metrics' THEN 20 ELSE 5 END) as numDataPoints FROM raw_data.devops WHERE time
 BETWEEN @scheduled_runtime - 10m AND @scheduled_runtime + 1m GROUP BY bin(time, 1m),
 region",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/1 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "per_minute_aggs_pt1m",
            "TimeColumn": "minute",
            "DimensionMappings": [
                {
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "numDataPoints",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "numDataPoints",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
```

```
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

```
SELECT bin(time, 1m) as minute, SUM(numDataPoints) as numDatapoints
FROM "derived"."per_minute_aggs_pt1m"
WHERE time BETWEEN from_milliseconds(1636699996446) AND
 from_milliseconds(1636743196446)
    AND region = 'us-east-1'
GROUP BY bin(time, 1m), region
ORDER BY 1 desc
```

派生テーブルには最新の集計があるため、前のクエリと以下のダッシュボードスナップショットからわかるように、派生テーブル per_minute_aggs_pt1m に直接クエリして、より新しい集計を取得できるようになりました。



スケジュールされた計算をより高速なスケジュールで更新する (つまり、5 分に対して 1 分) と、スケジュールされた計算のメンテナンスコストが増加することに注意してください。各計算の実行の通知メッセージは、スキャンされたデータの量と、派生テーブルに書き込まれたデータの量の統計を提供します。同様に、マージされたビューを使用して派生テーブルを結合する場合、マージされた

ビューでコストをクエリし、ダッシュボードのロードレイテンシーは派生テーブルのみをクエリするよりも高くなります。したがって、選択するアプローチは、ダッシュボードが更新される頻度と、スケジュールされたクエリのメンテナンスコストによって異なります。1 分に 1 回ほどダッシュボードを更新するユーザーが何十人もいる場合は、派生テーブルの更新頻度が高くなると、全体的なコストが低下する可能性があります。

各デバイスからの最終ポイント

アプリケーションでは、デバイスによって出力された最後の測定値を読み取る必要がある場合があります。特定の より前のデバイスの最後の測定を取得する一般的なユースケースがありますdate/time or the first measurement for a device after a given date/time。数百万のデバイスと何年ものデータがある場合、この検索では大量のデータをスキャンする必要がある場合があります。

以下に、スケジュールされたクエリを使用して、デバイスによって最後に発行されたポイントの検索を最適化する方法の例を示します。同じパターンを使用して、アプリケーションが最初のポイントクエリを必要とする場合も同様に最適化できます。

トピック

- ソーステーブルから計算
- 毎日の粒度で事前計算する派生テーブル
- 派生テーブルから計算
- ソーステーブルと派生テーブルからの組み合わせ

ソーステーブルから計算

以下は、特定のデプロイ内のサービス (例えば、特定のリージョン、セル、サイロ、および Availability_zone 内の特定のマイクロサービスのサーバー) によって出力された最後の測定値を見つけるためのクエリの例です。アプリケーション例では、このクエリは数百台のサーバーの最終測定値を返します。また、このクエリには無制限の時間述語があり、特定のタイムスタンプよりも古いデータを検索することに注意してください。

> **Note**
>
> max および max_by関数の詳細については、「」を参照してください集計関数。

```
SELECT instance_name, MAX(time) AS time, MAX_BY(gc_pause, time) AS last_measure
FROM "raw_data"."devops"
```

```
WHERE time < from_milliseconds(1636685271872)
    AND measure_name = 'events'
    AND region = 'us-east-1'
    AND cell = 'us-east-1-cell-10'
    AND silo = 'us-east-1-cell-10-silo-3'
    AND availability_zone = 'us-east-1-1'
    AND microservice_name = 'hercules'
GROUP BY region, cell, silo, availability_zone, microservice_name,
    instance_name, process_name, jdk_version
ORDER BY instance_name, time DESC
```

毎日の粒度で事前計算する派生テーブル

上記のユースケースをスケジュールされた計算に変換できます。アプリケーション要件が複数のリージョン、セル、サイロ、アベイラビリティーゾーン、マイクロサービスにわたってフリート全体でこれらの値を取得する必要がある場合は、1 つのスケジュール計算を使用してフリート全体の値を事前計算できます。これは、 LiveAnalyticsのサーバーレススケジュールクエリの Timestream のパワーであり、これらのクエリをアプリケーションのスケーリング要件に合わせてスケーリングできます。

以下は、特定の日のすべてのサーバーで最後のポイントを事前計算するためのクエリです。クエリには時間述語のみがあり、ディメンションの述語ではないことに注意してください。時間述語は、クエリを、指定されたスケジュール式に基づいて計算がトリガーされた時点からの過去 1 日に制限します。

```
SELECT region, cell, silo, availability_zone, microservice_name,
    instance_name, process_name, jdk_version,
    MAX(time) AS time, MAX_BY(gc_pause, time) AS last_measure
FROM raw_data.devops
WHERE time BETWEEN bin(@scheduled_runtime, 1d) - 1d AND bin(@scheduled_runtime, 1d)
    AND measure_name = 'events'
GROUP BY region, cell, silo, availability_zone, microservice_name,
    instance_name, process_name, jdk_version
```

以下は、上記のクエリを使用してスケジュールされた計算の設定です。このクエリは、UTC毎日午前 1 時に実行され、過去 1 日の集計を計算します。スケジュール式 cron(0 1 * * ? *) はこの動作を制御し、1 日が経過してから 1 時間後に実行され、1 日遅れて到着するデータを考慮します。

```
{
    "Name": "PT1DPerInstanceLastpoint",
    "QueryString": "SELECT region, cell, silo, availability_zone, microservice_name,
 instance_name, process_name, jdk_version, MAX(time) AS time, MAX_BY(gc_pause, time)
```

```
AS last_measure FROM raw_data.devops WHERE time BETWEEN bin(@scheduled_runtime, 1d) -
1d AND bin(@scheduled_runtime, 1d) AND measure_name = 'events' GROUP BY region, cell,
silo, availability_zone, microservice_name, instance_name, process_name, jdk_version",
   "ScheduleConfiguration": {
       "ScheduleExpression": "cron(0 1 * * ? *)"
   },
   "NotificationConfiguration": {
       "SnsConfiguration": {
           "TopicArn": "******"
       }
   },
   "TargetConfiguration": {
       "TimestreamConfiguration": {
           "DatabaseName": "derived",
           "TableName": "per_timeseries_lastpoint_pt1d",
           "TimeColumn": "time",
           "DimensionMappings": [
               {
                   "Name": "region",
                   "DimensionValueType": "VARCHAR"
               },
               {
                   "Name": "cell",
                   "DimensionValueType": "VARCHAR"
               },
               {
                   "Name": "silo",
                   "DimensionValueType": "VARCHAR"
               },
               {
                   "Name": "availability_zone",
                   "DimensionValueType": "VARCHAR"
               },
               {
                   "Name": "microservice_name",
                   "DimensionValueType": "VARCHAR"
               },
               {
                   "Name": "instance_name",
                   "DimensionValueType": "VARCHAR"
               },
               {
                   "Name": "process_name",
                   "DimensionValueType": "VARCHAR"
```

```
                },
                {
                    "Name": "jdk_version",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "last_measure",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "last_measure",
                        "MeasureValueType": "DOUBLE"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

派生テーブルから計算

前述の設定を使用して派生テーブルを定義し、スケジュールされたクエリの少なくとも 1 つのインスタンスで派生テーブルにマテリアライズドデータが含まれている場合、派生テーブルをクエリして最新の測定値を取得できるようになりました。以下は、派生テーブルのクエリ例です。

```
SELECT instance_name, MAX(time) AS time, MAX_BY(last_measure, time) AS last_measure
FROM "derived"."per_timeseries_lastpoint_pt1d"
WHERE time < from_milliseconds(1636746715649)
    AND measure_name = 'last_measure'
    AND region = 'us-east-1'
    AND cell = 'us-east-1-cell-10'
    AND silo = 'us-east-1-cell-10-silo-3'
    AND availability_zone = 'us-east-1-1'
    AND microservice_name = 'hercules'
GROUP BY region, cell, silo, availability_zone, microservice_name,
```

```
    instance_name, process_name, jdk_version
ORDER BY instance_name, time DESC
```

ソーステーブルと派生テーブルからの組み合わせ

前の例と同様に、派生テーブルからのデータには最新の書き込みはありません。したがって、先ほどと同様のパターンを再度使用して、古いデータの派生テーブルのデータをマージし、残りのチップのソースデータを使用できます。以下は、同様のUNIONアプローチを使用したこのようなクエリの例です。アプリケーション要件は、特定の期間より前に最新の測定値を見つけることであり、この開始時刻は過去のものである可能性があるため、このクエリの記述方法は、指定された時間を使用し、指定された時間から最大 1 日経過したソースデータを使用し、古いデータで派生テーブルを使用することです。以下のクエリ例からわかるように、ソースデータの述語時間は制限されます。これにより、データ量が大幅に多いソーステーブルでの効率的な処理が保証され、その後、無制限の時間述語が派生テーブルにあります。

```
WITH last_point_derived AS (
    SELECT instance_name, MAX(time) AS time, MAX_BY(last_measure, time) AS last_measure
    FROM "derived"."per_timeseries_lastpoint_pt1d"
    WHERE time < from_milliseconds(1636746715649)
        AND measure_name = 'last_measure'
        AND region = 'us-east-1'
        AND cell = 'us-east-1-cell-10'
        AND silo = 'us-east-1-cell-10-silo-3'
        AND availability_zone = 'us-east-1-1'
        AND microservice_name = 'hercules'
    GROUP BY region, cell, silo, availability_zone, microservice_name,
        instance_name, process_name, jdk_version
), last_point_source AS (
    SELECT instance_name, MAX(time) AS time, MAX_BY(gc_pause, time) AS last_measure
    FROM "raw_data"."devops"
    WHERE time < from_milliseconds(1636746715649) AND time >
 from_milliseconds(1636746715649) - 26h
        AND measure_name = 'events'
        AND region = 'us-east-1'
        AND cell = 'us-east-1-cell-10'
        AND silo = 'us-east-1-cell-10-silo-3'
        AND availability_zone = 'us-east-1-1'
        AND microservice_name = 'hercules'
    GROUP BY region, cell, silo, availability_zone, microservice_name,
        instance_name, process_name, jdk_version
)
SELECT instance_name, MAX(time) AS time, MAX_BY(last_measure, time) AS last_measure
```

```
FROM (
    SELECT * FROM last_point_derived
    UNION
    SELECT * FROM last_point_source
)
GROUP BY instance_name
ORDER BY instance_name, time DESC
```

前の図は、派生テーブルを構造化する方法の一例です。何年ものデータがある場合は、より多くのレベルの集計を使用できます。例えば、1 日の集計に加えて毎月の集計を設定したり、1 日の前に 1 時間ごとの集計を設定したりできます。そのため、直近 1 時間、直近 1 日 1 時間、過去 1 か月 1 日、および月単位で古い 1 日 1 日 1 日 1 日 1 日 1 日 1 日 1 日 1 日 1 日 1 日 1 日 1 日 日 1 日 日 1 日 日 1 日 日 日 日 1 日 日 1 日 日 日 1 日 日 設定するレベルと更新スケジュールの数は、これらのクエリが問題となる頻度の要件と、これらのクエリを同時に発行するユーザーの数によって異なります。

一意のディメンション値

特定のデータスライスに対応するメトリクスをドリルダウンするために、ディメンションの一意の値を変数として使用するダッシュボードがあるユースケースがある場合があります。以下のスナップショットは、ダッシュボードがリージョン、セル、サイロ、マイクロサービス、アベイラビリティーゾーンなどの複数のディメンションの一意の値を事前に入力する例です。ここでは、スケジュールされたクエリを使用して、追跡しているメトリクスからこれらの変数のこれらの個別の値の計算を大幅に高速化する方法の例を示します。

トピック

- [未加工データ](#)
- [一意のディメンション値を事前計算する](#)
- [派生テーブルからの変数の計算](#)

未加工データ

SELECT DISTINCT を使用して、データから表示される個別の値を計算できます。例えば、リージョンの個別の値を取得する場合は、このフォームのクエリを使用できます。

```
SELECT DISTINCT region
FROM "raw_data"."devops"
WHERE time > ago(1h)
ORDER BY 1
```

数百万のデバイスと数十億の時系列を追跡している可能性があります。ただし、ほとんどの場合、これらの興味深い変数はカーディナリティのディメンションが低く、数から数十の値があります。生データDISTINCTからコンピューティングするには、大量のデータをスキャンする必要があります。

一意のディメンション値を事前計算する

これらの変数を高速にロードして、ダッシュボードがインタラクティブになるようにします。さらに、これらの変数はダッシュボードのロードごとに計算されることが多いため、費用対効果も高くする必要があります。スケジュールされたクエリを使用してこれらの変数の検出を最適化し、派生テーブルでマテリアライズできます。

まず、値の計算時に述語で使用するDISTINCT値または列を計算する必要があるディメンションを特定する必要がありますDISTINCT。

この例では、ダッシュボードがディメンションリージョン、セル、サイロ、アベイラビリティーゾーン、マイクロサービスの個別の値を入力していることがわかります。そのため、以下のクエリを使用して、これらの一意の値を事前計算できます。

```
SELECT region, cell, silo, availability_zone, microservice_name,
    min(@scheduled_runtime) AS time, COUNT(*) as numDataPoints
FROM raw_data.devops
WHERE time BETWEEN @scheduled_runtime - 15m AND @scheduled_runtime
GROUP BY region, cell, silo, availability_zone, microservice_name
```

ここで注意すべき重要な点がいくつかあります。

- 1 つのスケジュールされた計算を使用して、さまざまなクエリの値を事前計算できます。例えば、前述のクエリを使用して、5 つの異なる変数の値を事前計算します。したがって、変数ごとに 1 つは必要ありません。この同じパターンを使用して、複数のパネル間で共有計算を識別し、維持する必要があるスケジュールされたクエリの数を最適化できます。
- ディメンションの一意の値は、本質的に時系列データではありません。したがって、@scheduled_runtime を使用してこれを時系列に変換します。このデータを @scheduled_runtime パラメータに関連付けることで、特定の時点でどの一意の値が表示されたかを追跡し、そこから時系列データを作成することもできます。
- 前の例では、追跡されているメトリクス値が表示されます。この例では COUNT(*) を使用します。ダッシュボードを追跡する場合は、他の意味のある集計を計算できます。

以下は、前のクエリを使用したスケジュールされた計算の設定です。この例では、スケジュール式 cron(0/15 * * * ? を使用して 15 分に 1 回更新するように設定されています。 *)。

```
{
    "Name": "PT15mHighCardPerUniqueDimensions",
    "QueryString": "SELECT region, cell, silo, availability_zone, microservice_name,
 min(@scheduled_runtime) AS time, COUNT(*) as numDataPoints FROM raw_data.devops WHERE
 time BETWEEN @scheduled_runtime - 15m AND @scheduled_runtime GROUP BY region, cell,
 silo, availability_zone, microservice_name",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/15 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "hc_unique_dimensions_pt15m",
            "TimeColumn": "time",
            "DimensionMappings": [
                {
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "cell",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "silo",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "availability_zone",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "microservice_name",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "count_multi",
```

```
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "numDataPoints",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

派生テーブルからの変数の計算

スケジュールされた計算が派生テーブル hc_unique_dimensions_pt15m の一意の値を事前にマテリアライズしたら、派生テーブルを使用してディメンションの一意の値を効率的に計算できます。以下は、一意の値を計算する方法と、これらの一意の値クエリで他の変数を述語として使用する方法に関するクエリの例です。

リージョン

```
SELECT DISTINCT region
FROM "derived"."hc_unique_dimensions_pt15m"
WHERE time > ago(1h)
ORDER BY 1
```

セル

```
SELECT DISTINCT cell
FROM "derived"."hc_unique_dimensions_pt15m"
WHERE time > ago(1h)
  AND region = '${region}'
ORDER BY 1
```

サイロ

```
SELECT DISTINCT silo
FROM "derived"."hc_unique_dimensions_pt15m"
WHERE time > ago(1h)
   AND region = '${region}' AND cell = '${cell}'
ORDER BY 1
```

マイクロサービス

```
SELECT DISTINCT microservice_name
FROM "derived"."hc_unique_dimensions_pt15m"
WHERE time > ago(1h)
   AND region = '${region}' AND cell = '${cell}'
ORDER BY 1
```

アベイラビリティーゾーン

```
SELECT DISTINCT availability_zone
FROM "derived"."hc_unique_dimensions_pt15m"
WHERE time > ago(1h)
   AND region = '${region}' AND cell = '${cell}' AND silo = '${silo}'
ORDER BY 1
```

遅延受信データの処理

の Timestream にデータが取り込まれた時間が、取り込まれた行に関連付けられたタイムスタンプと比較して LiveAnalytics 大幅に遅延するなど、データが大幅に遅れるシナリオがある場合があります。前の例では、@scheduled_runtime パラメータで定義された時間範囲を使用して、到着が遅れたデータを考慮する方法を見てきました。ただし、データが数時間または数日遅延する可能性があるユースケースがある場合は、派生テーブルの事前計算が、そのような遅延受信データを反映するように適切に更新されるように、別のパターンが必要になる場合があります。遅延受信データに関する一般的な情報については、「」を参照してくださいデータの作成 (挿入とアップサート) 。

以下に、この到着遅延データに対処する 2 つの異なる方法を示します。

- データ到着に予測可能な遅延がある場合は、別の「キャッチアップ」スケジュールされた計算を使用して、到着が遅れたデータの集計を更新できます。
- 予測不可能な遅延や時折の遅延到着データがある場合は、手動実行を使用して派生テーブルを更新できます。

このディスカッションでは、データ到着が遅れた場合のシナリオについて説明します。ただし、ソーステーブルのデータを変更し、派生テーブルの集計を更新する場合のデータ修正にも同じ原則が適用されます。

トピック

スケジュールされたキャッチアップクエリ

時間内に到着したデータを集計するクエリ

以下は、データ到着に予測可能な遅延が発生した場合に、自動方法を使用して集計を更新する方法を示すパターンです。以下のリアルタイムデータでスケジュールされた計算の例の 1 つを考えてみましょう。このスケジュールされた計算では、派生テーブルが 30 分に 1 回更新され、最大 1 時間の遅延データがすでに処理されています。

```
{
    "Name": "MultiPT30mPerHrPerTimeseriesDPCount",
    "QueryString": "SELECT region, cell, silo, availability_zone, microservice_name,
 instance_type, os_version, instance_name, process_name, jdk_version, bin(time,
 1h) as hour, SUM(CASE WHEN measure_name = 'metrics' THEN 20 ELSE 5 END) as
 numDataPoints FROM raw_data.devops WHERE time BETWEEN bin(@scheduled_runtime, 1h)
 - 1h AND @scheduled_runtime + 1h GROUP BY region, cell, silo, availability_zone,
 microservice_name, instance_type, os_version, instance_name, process_name,
 jdk_version, bin(time, 1h)",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/30 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "dp_per_timeseries_per_hr",
            "TimeColumn": "hour",
            "DimensionMappings": [
                {
```

```
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "cell",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "silo",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "availability_zone",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "microservice_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "instance_type",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "os_version",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "instance_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "process_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "jdk_version",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "numDataPoints",
                "MultiMeasureAttributeMappings": [
                    {
```

```
                        "SourceColumn": "numDataPoints",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

遅延到着データの集計を更新するキャッチアップクエリ

次に、データが約 12 時間遅延する可能性がある場合を考えてみましょう。以下は同じクエリのバリアントです。ただし、違いは、スケジュールされた計算がトリガーされるときと比較して、最大 12 時間遅延したデータの集計を計算することです。例えば、以下の例ではクエリが表示されています。このクエリがターゲットとしている時間範囲は、クエリがトリガーされる 2 時間前から 14 時間前までです。さらに、スケジュール式 cron(0 0,12 * * ? *) に気付いた場合、UTC毎日 00:00 UTC と 12:00 に計算がトリガーされます。したがって、クエリが 2021-12-01 00:00:00 にトリガーされると、クエリの更新は 2021-11-30 10:00:00 から 2021-11-30 22:00:00 の時間範囲で集計されます。スケジュールされたクエリは、 LiveAnalyticsの書き込みの Timestream に似たアップサートセマンティクスを使用します。このキャッチアップクエリでは、ウィンドウに遅延到着データがある場合、または新しい集計が見つかった場合 (例えば、元のスケジュールされた計算がトリガーされたときに存在しなかった新しいグループがこの集計に表示されます) 、新しい集計が派生テーブルに挿入されます。同様に、次のインスタンスが 2021-12-01 12:00:00 にトリガーされると、そのインスタンスは 2021-11-30 22:00:00 から 2021-12-01 10:00:00 の範囲の集計を更新します。

```
       {
    "Name": "MultiPT12HPerHrPerTimeseriesDPCountCatchUp",
    "QueryString": "SELECT region, cell, silo, availability_zone, microservice_name,
  instance_type, os_version, instance_name, process_name, jdk_version, bin(time, 1h)
  as hour, SUM(CASE WHEN measure_name = 'metrics' THEN 20 ELSE 5 END) as numDataPoints
  FROM raw_data.devops WHERE time BETWEEN bin(@scheduled_runtime, 1h) - 14h AND
  bin(@scheduled_runtime, 1h) - 2h GROUP BY region, cell, silo, availability_zone,
```

```
microservice_name, instance_type, os_version, instance_name, process_name,
jdk_version, bin(time, 1h)",
   "ScheduleConfiguration": {
       "ScheduleExpression": "cron(0 0,12 * * ? *)"
   },
   "NotificationConfiguration": {
       "SnsConfiguration": {
           "TopicArn": "******"
       }
   },
   "TargetConfiguration": {
       "TimestreamConfiguration": {
           "DatabaseName": "derived",
           "TableName": "dp_per_timeseries_per_hr",
           "TimeColumn": "hour",
           "DimensionMappings": [
               {
                   "Name": "region",
                   "DimensionValueType": "VARCHAR"
               },
               {
                   "Name": "cell",
                   "DimensionValueType": "VARCHAR"
               },
               {
                   "Name": "silo",
                   "DimensionValueType": "VARCHAR"
               },
               {
                   "Name": "availability_zone",
                   "DimensionValueType": "VARCHAR"
               },
               {
                   "Name": "microservice_name",
                   "DimensionValueType": "VARCHAR"
               },
               {
                   "Name": "instance_type",
                   "DimensionValueType": "VARCHAR"
               },
               {
                   "Name": "os_version",
                   "DimensionValueType": "VARCHAR"
               },
```

```
                {
                    "Name": "instance_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "process_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "jdk_version",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "numDataPoints",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "numDataPoints",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

この例は、遅延到着が 12 時間に制限されており、リアルタイムウィンドウより遅れて到着するデータについては、12 時間に 1 回派生テーブルを更新しても問題ないと仮定した図です。このパターンを適応させて、派生テーブルを 1 時間に 1 回更新し、派生テーブルが到着遅延データをより早く反映するようにすることができます。同様に、1 日や 1 週間以上など、12 時間以上経過するように時間範囲を調整して、予測可能な遅延受信データを処理できます。

予測不可能な到着遅延データの手動実行

予測不可能な遅延到着データがある場合や、ソースデータを変更して、事後にいくつかの値を更新した場合があります。このような場合はすべて、スケジュールされたクエリを手動でトリガーして、派生テーブルを更新できます。これを実現する方法の例を次に示します。

派生テーブル dp_per_timeseries_per_hr に計算が書き込まれるユースケースがあるとします。テーブル devops のベースデータは、2021-11-30 23:00:00 ~ 2021-12-01 00:00:00 の時間範囲で更新されました。この派生テーブルの更新に使用できるスケジュールされたクエリには、マルチ PT30mPerHrPerTimeseriesDPCount とマルチ の 2 種類があります PT12HPerHrPerTimeseriesDPCountCatchUp。Timestream で作成するスケジュールされた各計算 LiveAnalytics には、計算の作成時またはリストオペレーションの実行時に取得ARNする一意の があります。計算ARNには を使用し、このオペレーションを実行するには、クエリによって実行されたパラメータ @scheduled_runtime の値を使用できます。

マルチ PT30mPerHrPerTimeseriesDPCount の計算に ARN arn_1 があり、この計算を使用して派生テーブルを更新するとします。上記のスケジュールされた計算では、@scheduled_runtime 値の 1 時間前と 1 時間後に集計が更新されるため、@scheduled_runtime パラメータの 2021-12-01 00:00:00 の値を使用して、更新 (2021-11-30 23:00:00 ~ 2021-12-01 00:00:00) の時間範囲を力バーできます。を使用して ExecuteScheduledQuery API、この計算ARNの と時間パラメータ値を工ポック秒 (内UTC) で渡すことで、これを実現できます。以下は を使用する例AWSCLIであり、 の Timestream でSDKsサポートされている のいずれかを使用して、同じパターンをたどることができます LiveAnalytics。

```
aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638316800 --profile profile --region us-east-1
```

前の例では、プロファイルは、このAPI呼び出しを行うための適切な権限を持つAWSプロファイルであり、1638316800 は 2021-12-01 00:00:00 の工ポック秒に対応します。この手動トリガーは、システムが希望する期間にこの呼び出しをトリガーしたと仮定すると、自動トリガーとほぼ同じ動作をします。

より長い期間に更新があった場合は、ベースデータが 2021-11-30 23:00:00 ~ 2021-12-01 11:00:00 に更新されたとすると、前述のクエリを複数回トリガーして、この期間全体をカバーすることができます。例えば、次のように 6 つの異なる実行を実行できます。

```
aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638316800 --profile profile --region us-east-1
```

```
aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638324000 --profile profile --region us-east-1

aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638331200 --profile profile --region us-east-1

aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638338400 --profile profile --region us-east-1

aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638345600 --profile profile --region us-east-1

aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638352800 --profile profile --region us-east-1
```

前の 6 つのコマンドは、2021-12-01 00:00:00、2021-12-01 02:00:00、2021-12-01 04:0:00、2021-12-01 06:00:00、2021-12-01 08:00:00、および 2021-12-01 10:00 に呼び出されたスケジュールされた計算に対応します。

または、2021-12-01 13:00:00 にトリガーされた計算マルチ PT12HPerHrPerTimeseriesDPCountCatchUpを 1 回の実行に使用して、12 時間範囲全体の集計を更新することもできます。例えば、arn_2 がその計算ARNの である場合、 から次のコマンドを実行できますCLI。

```
aws timestream-query execute-scheduled-query --scheduled-query-arn arn_2 --invocation-
time 1638363600 --profile profile --region us-east-1
```

手動トリガーでは、自動トリガータイムスタンプと調整する必要のない呼び出し時パラメータにタイムスタンプを使用できます。例えば、前の例では、自動スケジュールがタイムスタンプ 2021-12-01 10:00:00、 12:00:00、および 2021-12-01 00:00:00 でのみトリガーする場合でも、 2021-12-01 13:00:00 2021-12-02 に計算をトリガーしました。Timestream for LiveAnalytics は、手動操作に必要な適切な値でトリガーする柔軟性を提供します。

を使用する際の重要な考慮事項を以下に示します ExecuteScheduledQuery API。

- これらの呼び出しの複数をトリガーする場合は、これらの呼び出しが重複する時間範囲で結果を生成しないようにする必要があります。例えば、前の例では、6 つの呼び出しがありました。各呼び出しは 2 時間の時間範囲をカバーするため、更新の重複を避けるために、呼び出しタイムスタンプはそれぞれ 2 時間分散されました。これにより、派生テーブル内のデータが、一致がソース

テーブルから集計される状態になります。重複しない時間範囲を確保できない場合は、これらの実行が順番にトリガーされていることを確認します。複数の実行を同時にトリガーし、その時間範囲が重複する場合、トリガーの失敗が表示され、これらの実行のエラーレポートにバージョン競合が表示される可能性があります。スケジュールされたクエリ呼び出しによって生成された結果には、呼び出しがトリガーされたタイミングに基づいてバージョンが割り当てられます。したがって、新しい呼び出しによって生成された行は、バージョンが高くなります。上位バージョンのレコードは、下位バージョンのレコードを上書きする可能性があります。自動的にトリガーされるスケジュールされたクエリの場合、 の LiveAnalytics Timestream はスケジュールを自動的に管理するため、後続の呼び出しの時間範囲が重複してもこれらの問題は表示されません。

- 前述のように、@scheduled_runtime の任意のタイムスタンプ値を使用して呼び出しをトリガーできます。したがって、ソーステーブルでデータが更新された範囲に対応する派生テーブルで適切な時間範囲が更新されるように、値を適切に設定するのはユーザーの責任です。
- これらの手動トリガーは、 DISABLED状態にあるスケジュールされたクエリにも使用できます。これにより、 DISABLED状態であるため、自動スケジュールで実行されない特別なクエリを定義できます。むしろ、手動トリガーを使用して、データ修正や遅延到着のユースケースを管理できます。

過去の計算のバックフィル

スケジュールされた計算を作成すると、 の Timestream はクエリの実行 LiveAnalytics を管理します。ここでは、更新が指定したスケジュール式によって管理されます。ソーステーブルの履歴データの量に応じて、履歴データに対応する集計を使用して派生テーブルを更新できます。前のロジックを手動トリガーに使用して、履歴集計をバックフィルできます。

例えば、派生テーブル per_timeseries_lastpoint_pt1d を考慮すると、スケジュールされた計算は過去 1 日 1 回更新されます。ソーステーブルに 1 年分のデータがある場合は、このスケジュールされた計算ARNに を使用し、1 歳までのすべての日に対して手動でトリガーして、派生テーブルにすべての履歴クエリが入力されるようにすることができます。手動トリガーのすべての注意事項がここに適用されることに注意してください。さらに、履歴取り込みが派生テーブル上のマグネティックストアに書き込むように派生テーブルが設定されている場合は、マグネティックストアへの書き込みのベストプラクティスと制限に注意してください。 https://docs.aws.amazon.com/timestream/latest/developerguide/ts-limits.html

スケジュールされたクエリの例

このセクションでは、 LiveAnalyticsのスケジュールされたクエリの Timestream を使用して、フリート全体の統計を視覚化してデバイスのフリートを効果的にモニタリングするときに、コストと

ダッシュボードのロード時間を最適化する方法の例を示します。の Timestream のスケジュールされたクエリ LiveAnalytics では、 の Timestream の全SQL表面領域を使用してクエリを表現できます LiveAnalytics。クエリには、1 つ以上のソーステーブルを含めることができ、 LiveAnalyticsの SQL言語に対して Timestream で許可される集計やその他のクエリを実行し、クエリの結果を の Timestream の別の宛先テーブルに格納できます LiveAnalytics。

このセクションでは、スケジュールされたクエリのターゲットテーブルを派生テーブル として参照します。

例として、複数のデプロイ (リージョン、セル、サイロなど) にデプロイされた多数のサーバーフリートをモニタリングし、複数のマイクロサービスを監視し、 の Timestream を使用してフリート全体の統計を追跡する DevOps アプリケーションを使用します LiveAnalytics。使用するスキーマの例については、スケジュールされたクエリのサンプルスキーマ を参照してください。

以下のシナリオについて説明します。

- ダッシュボードを変換し、取り込んだ生データから集計された統計を Timestream LiveAnalytics にプロットしてスケジュールされたクエリにし、事前に計算された集計を使用して集計統計を表示する新しいダッシュボードを作成する方法。
- スケジュールされたクエリを組み合わせて集計ビューと未加工の詳細なデータを取得し、詳細にドリルダウンする方法。これにより、スケジュールされたクエリを使用して一般的なフリート全体のオペレーションを最適化しながら、未加工データを保存および分析できます。
- スケジュールされたクエリを使用してコストを最適化するには、複数のダッシュボードでどの集計が使用され、同じスケジュールされたクエリが、同じまたは複数のダッシュボード内の複数のパネルに入力されているかを調べます。

トピック

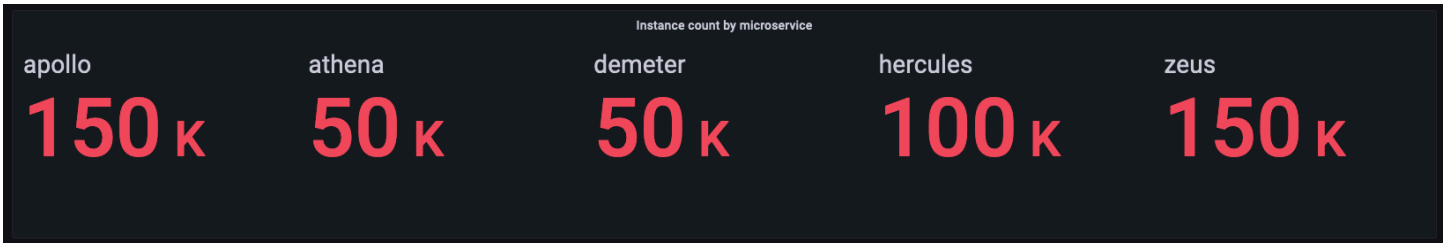
- 集計ダッシュボードをスケジュールされたクエリに変換する
- ドリルダウンにスケジュールされたクエリと未加工データを使用する
- ダッシュボード間でスケジュールされたクエリを共有してコストを最適化する
- ベーステーブルのクエリとスケジュールされたクエリ結果のクエリの比較

集計ダッシュボードをスケジュールされたクエリに変換する

5 つのマイクロサービスおよびサービスがデプロイされている 6 つのリージョンによって、フリートのホスト数などのフリート全体の統計を計算するとします。以下のスナップショットから、メト

リクスを出力するサーバーが 500K000 台あり、より大きなリージョン (us-east-1 など) の一部には >200,000 台のサーバーがあります。

これらの集計を計算すると、数百ギガバイトのデータにわたって異なるインスタンス名を計算する場合、データのスキャンコストに加えて、クエリレイテンシーが数十秒になる可能性があります。



元のダッシュボードクエリ

dasboard パネルに表示される集計は、以下のクエリを使用して生データから計算されます。クエリは、個別のカウントや複数の集計関数など、複数のSQLコンストラクトを使用します。

```
SELECT CASE WHEN microservice_name = 'apollo' THEN num_instances ELSE NULL END AS
 apollo,
    CASE WHEN microservice_name = 'athena' THEN num_instances ELSE NULL END AS athena,
    CASE WHEN microservice_name = 'demeter' THEN num_instances ELSE NULL END AS
 demeter,
    CASE WHEN microservice_name = 'hercules' THEN num_instances ELSE NULL END AS
 hercules,
    CASE WHEN microservice_name = 'zeus' THEN num_instances ELSE NULL END AS zeus
FROM (
    SELECT microservice_name, SUM(num_instances) AS num_instances
    FROM (
        SELECT microservice_name, COUNT(DISTINCT instance_name) as num_instances
        FROM "raw_data"."devops"
        WHERE time BETWEEN from_milliseconds(1636526171043) AND
 from_milliseconds(1636612571043)
            AND measure_name = 'metrics'
        GROUP BY region, cell, silo, availability_zone, microservice_name
    )
    GROUP BY microservice_name
)
```

スケジュールされたクエリへの変換

以前のクエリは、次のようにスケジュールされたクエリに変換できます。まず、リージョン、セル、サイロ、アベイラビリティーゾーン、マイクロサービス内の特定のデプロイ内の個別のホス

ト名を計算します。次に、ホストを合計して、マイクロサービスホスト数あたりの 1 時間あたりの を計算します。スケジュールされたクエリでサポートされている @scheduled_runtimeパラメータを使用すると、クエリが呼び出された過去 1 時間、再計算できます。内部クエリの bin(@scheduled_runtime, 1h)WHERE句の は、クエリが 1 時間の途中でスケジュールされていても、1 時間分のデータを取得できるようにします。

スケジュールされた計算設定でわかるように、クエリは時間単位の集計を計算しますが、派生テーブルの更新をより早く取得できるように、30 分ごとに更新するように設定されています。これを鮮度要件に基づいて調整できます。例えば、15 分ごとに集計を再計算したり、時間境界で再計算したりできます。

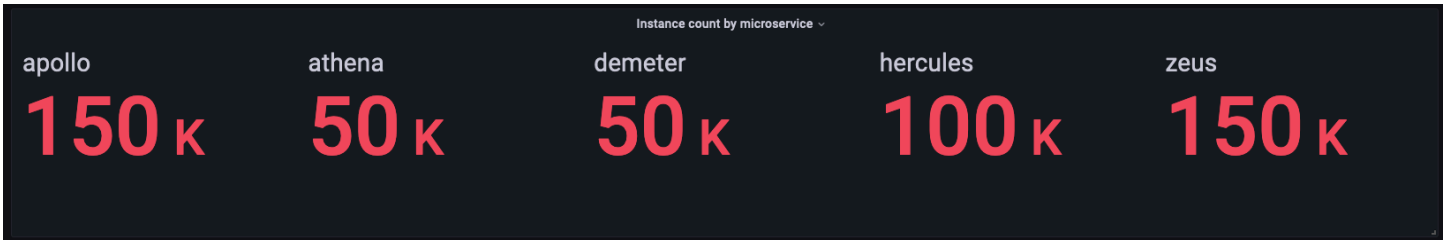
```
SELECT microservice_name, hour, SUM(num_instances) AS num_instances
FROM (
        SELECT microservice_name, bin(time, 1h) AS hour,
            COUNT(DISTINCT instance_name) as num_instances
       FROM raw_data.devops
       WHERE time BETWEEN bin(@scheduled_runtime, 1h) - 1h AND @scheduled_runtime

           AND measure_name = 'metrics'
       GROUP BY region, cell, silo, availability_zone, microservice_name, bin(time, 1h)

     )
GROUP BY microservice_name, hour
```

```
{
    "Name": "MultiPT30mHostCountMicroservicePerHr",
    "QueryString": "SELECT microservice_name, hour, SUM(num_instances) AS num_instances
    FROM (        SELECT microservice_name, bin(time, 1h) AS hour, COUNT(DISTINCT
 instance_name) as num_instances        FROM raw_data.devops        WHERE time BETWEEN
 bin(@scheduled_runtime, 1h) - 1h AND @scheduled_runtime            AND measure_name
 = 'metrics'        GROUP BY region, cell, silo, availability_zone, microservice_name,
 bin(time, 1h)    )    GROUP BY microservice_name, hour",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/30 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
```

```
            "DatabaseName": "derived",
            "TableName": "host_count_pt1h",
            "TimeColumn": "hour",
            "DimensionMappings": [
                {
                    "Name": "microservice_name",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "num_instances",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "num_instances",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

新しいダッシュボードで事前計算された結果を使用する

これで、作成したスケジュールされたクエリから派生テーブルを使用して集計ビューダッシュボードを作成する方法が表示されます。ダッシュボードスナップショットから、派生テーブルとベーステーブルから計算された集計も一致していることを検証することもできます。派生テーブルを使用してダッシュボードを作成すると、生データからこれらの集計を計算する場合と比較して、派生テーブルを使用するロード時間が著しく短縮され、コストが削減されます。以下は、事前計算されたデータを使用したダッシュボードのスナップショットと、テーブルに保存されている事前計算されたデータを使用したこのパネルのレンダリングに使用されるクエリです「derived」。「host_count_pt1h」。クエリの構造は、生データのダッシュボードで使用されたクエリと非常に似ていることに注意してください。ただし、このクエリが集計している個別のカウントを既に計算している派生テーブルを使用する点が異なります。




```
SELECT CASE WHEN microservice_name = 'apollo' THEN num_instances ELSE NULL END AS
 apollo,
    CASE WHEN microservice_name = 'athena' THEN num_instances ELSE NULL END AS athena,
    CASE WHEN microservice_name = 'demeter' THEN num_instances ELSE NULL END AS
 demeter,
    CASE WHEN microservice_name = 'hercules' THEN num_instances ELSE NULL END AS
 hercules,
    CASE WHEN microservice_name = 'zeus' THEN num_instances ELSE NULL END AS zeus
FROM (
    SELECT microservice_name, AVG(num_instances) AS num_instances
    FROM (
        SELECT microservice_name, bin(time, 1h), SUM(num_instances) as num_instances
        FROM "derived"."host_count_pt1h"
        WHERE time BETWEEN from_milliseconds(1636567785421) AND
 from_milliseconds(1636654185421)
            AND measure_name = 'num_instances'
        GROUP BY microservice_name, bin(time, 1h)
    )
    GROUP BY microservice_name
)
```

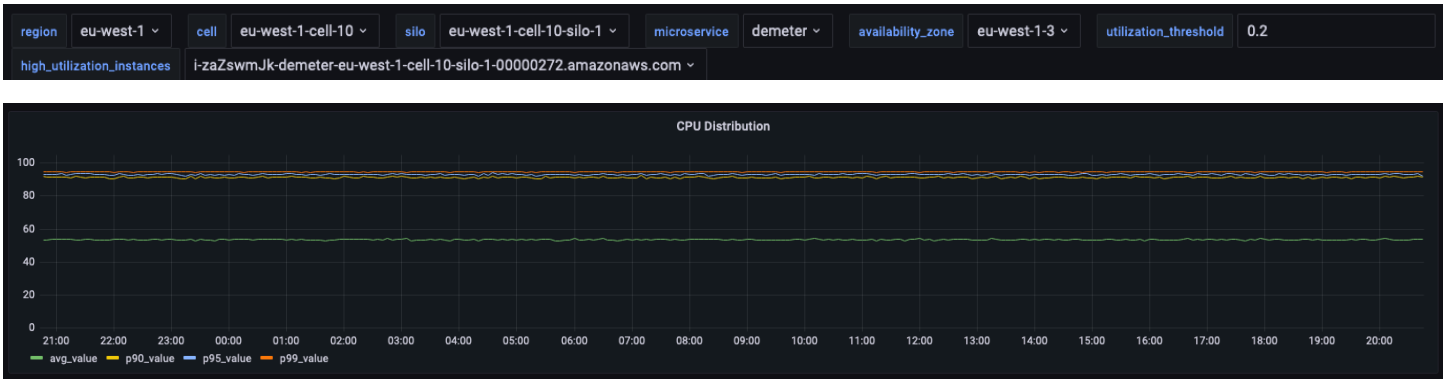
ドリルダウンにスケジュールされたクエリと未加工データを使用する

フリート全体の集計統計を使用してドリルダウンが必要な領域を特定し、未加工データを使用して詳細なデータをドリルダウンしてより深いインサイトを得ることができます。

この例では、集約ダッシュボードを使用して、他のデプロイと比較してCPU使用率が高いと思われるデプロイ (デプロイは特定のリージョン、セル、サイロ、アベイラビリティーゾーン内の特定のマイクロサービス用) を特定する方法を説明します。その後、ドリルダウンして、未加工データの使用をよりよく理解できます。これらのドリルダウンはまれであり、デプロイに関連するデータにのみアクセスする可能性があるため、この分析には raw データを使用できます。スケジュールされたクエリを使用する必要はありません。

デプロイごとのドリルダウン

以下のダッシュボードでは、特定のデプロイ内のより詳細なサーバーレベルの統計をドリルダウンできます。フリートのさまざまな部分にドリルダウンしやすくするために、このダッシュボードでは、リージョン、セル、サイロ、マイクロサービス、アベイラビリティーゾーンなどの変数を使用します。次に、そのデプロイの集計統計が表示されます。



以下のクエリでは、変数のドロップダウンで選択した値がクエリの WHERE句の述語として使用されていることがわかります。これにより、デプロイのデータにのみ焦点を合わせることができます。次に、パネルはそのデプロイ内のインスタンスの集計CPUメトリクスをプロットします。raw データを使用して、インタラクティブなクエリレイテンシーでこのドリルダウンを実行して、より深いインサイトを得ることができます。

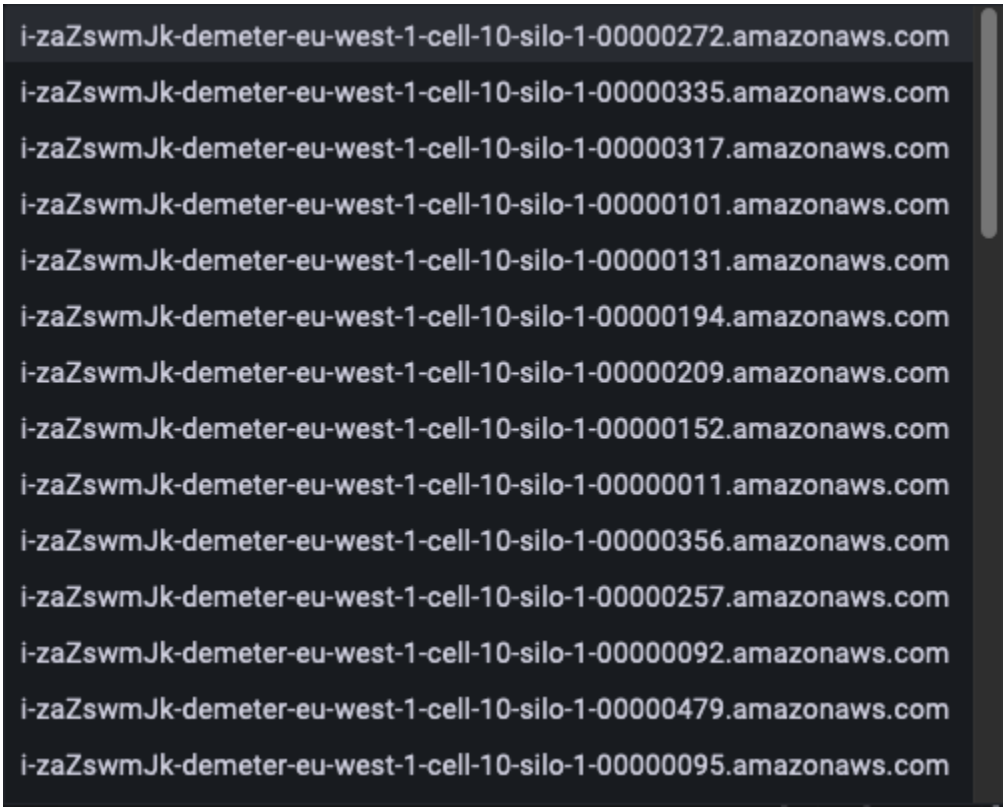
```
SELECT bin(time, 5m) as minute,
    ROUND(AVG(cpu_user), 2) AS avg_value,
    ROUND(APPROX_PERCENTILE(cpu_user, 0.9), 2) AS p90_value,
    ROUND(APPROX_PERCENTILE(cpu_user, 0.95), 2) AS p95_value,
    ROUND(APPROX_PERCENTILE(cpu_user, 0.99), 2) AS p99_value
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099476) AND
 from_milliseconds(1636613499476)
    AND region = 'eu-west-1'
    AND cell = 'eu-west-1-cell-10'
    AND silo = 'eu-west-1-cell-10-silo-1'
    AND microservice_name = 'demeter'
    AND availability_zone = 'eu-west-1-3'
    AND measure_name = 'metrics'
GROUP BY bin(time, 5m)
ORDER BY 1
```

インスタンスレベルの統計

このダッシュボードは、CPU使用率の高いサーバー/インスタンスを使用率の高い順にソートした別の変数をさらに計算します。この変数の計算に使用されるクエリを以下に示します。

```
WITH microservice_cell_avg AS (
    SELECT AVG(cpu_user) AS microservice_avg_metric
    FROM "raw_data"."devops"
    WHERE $__timeFilter
        AND measure_name = 'metrics'
        AND region = '${region}'
        AND cell = '${cell}'
        AND silo = '${silo}'
        AND availability_zone = '${availability_zone}'
        AND microservice_name = '${microservice}'
), instance_avg AS (
    SELECT instance_name,
        AVG(cpu_user) AS instance_avg_metric
    FROM "raw_data"."devops"
    WHERE $__timeFilter
        AND measure_name = 'metrics'
        AND region = '${region}'
        AND cell = '${cell}'
        AND silo = '${silo}'
        AND microservice_name = '${microservice}'
        AND availability_zone = '${availability_zone}'
    GROUP BY availability_zone, instance_name
)
SELECT i.instance_name
FROM instance_avg i CROSS JOIN microservice_cell_avg m
WHERE i.instance_avg_metric > (1 + ${utilization_threshold}) *
 m.microservice_avg_metric
ORDER BY i.instance_avg_metric DESC
```

前のクエリでは、他の変数に選択した値に応じて、変数が動的に再計算されます。デプロイに変数が入力されたら、リストから個々のインスタンスを選択して、そのインスタンスのメトリクスをさらに視覚化できます。以下のスナップショットに示すように、インスタンス名のドロップダウンからさまざまなインスタンスを選択できます。



前のパネルには、選択したインスタンスの統計が表示され、以下のクエリはこれらの統計の取得に使用されます。

```
SELECT BIN(time, 30m) AS time_bin,
    AVG(cpu_user) AS avg_cpu,
    ROUND(APPROX_PERCENTILE(cpu_user, 0.99), 2) as p99_cpu
```

```
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099477) AND
 from_milliseconds(1636613499477)
    AND measure_name = 'metrics'
    AND region = 'eu-west-1' AND cell = 'eu-west-1-cell-10' AND silo = 'eu-west-1-
cell-10-silo-1'
    AND availability_zone = 'eu-west-1-3' AND microservice_name = 'demeter'
    AND instance_name = 'i-zaZswmJk-demeter-eu-west-1-cell-10-
silo-1-00000272.amazonaws.com'
GROUP BY BIN(time, 30m)
ORDER BY time_bin desc
```

```
SELECT BIN(time, 30m) AS time_bin,
    AVG(memory_used) AS avg_memory,
    ROUND(APPROX_PERCENTILE(memory_used, 0.99), 2) as p99_memory
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099477) AND
 from_milliseconds(1636613499477)
    AND measure_name = 'metrics'
    AND region = 'eu-west-1' AND cell = 'eu-west-1-cell-10' AND silo = 'eu-west-1-
cell-10-silo-1'
    AND availability_zone = 'eu-west-1-3' AND microservice_name = 'demeter'
    AND instance_name = 'i-zaZswmJk-demeter-eu-west-1-cell-10-
silo-1-00000272.amazonaws.com'
GROUP BY BIN(time, 30m)
ORDER BY time_bin desc
```

```
SELECT COUNT(gc_pause)
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099477) AND
 from_milliseconds(1636613499478)
    AND measure_name = 'events'
    AND region = 'eu-west-1' AND cell = 'eu-west-1-cell-10' AND silo = 'eu-west-1-
cell-10-silo-1'
    AND availability_zone = 'eu-west-1-3' AND microservice_name = 'demeter'
    AND instance_name = 'i-zaZswmJk-demeter-eu-west-1-cell-10-
silo-1-00000272.amazonaws.com'
```

```
SELECT avg(gc_pause) as avg, round(approx_percentile(gc_pause, 0.99), 2) as p99
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099478) AND
 from_milliseconds(1636613499478)
```

```
    AND measure_name = 'events'
    AND region = 'eu-west-1' AND cell = 'eu-west-1-cell-10' AND silo = 'eu-west-1-
cell-10-silo-1'
    AND availability_zone = 'eu-west-1-3' AND microservice_name = 'demeter'
    AND instance_name = 'i-zaZswmJk-demeter-eu-west-1-cell-10-
silo-1-00000272.amazonaws.com'
```

```
SELECT BIN(time, 30m) AS time_bin,
    AVG(disk_io_reads) AS avg,
    ROUND(APPROX_PERCENTILE(disk_io_reads, 0.99), 2) as p99
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099478) AND
 from_milliseconds(1636613499478)
    AND measure_name = 'metrics'
    AND region = 'eu-west-1' AND cell = 'eu-west-1-cell-10' AND silo = 'eu-west-1-
cell-10-silo-1'
    AND availability_zone = 'eu-west-1-3' AND microservice_name = 'demeter'
    AND instance_name = 'i-zaZswmJk-demeter-eu-west-1-cell-10-
silo-1-00000272.amazonaws.com'
GROUP BY BIN(time, 30m)
ORDER BY time_bin desc
```

ダッシュボード間でスケジュールされたクエリを共有してコストを最適化する

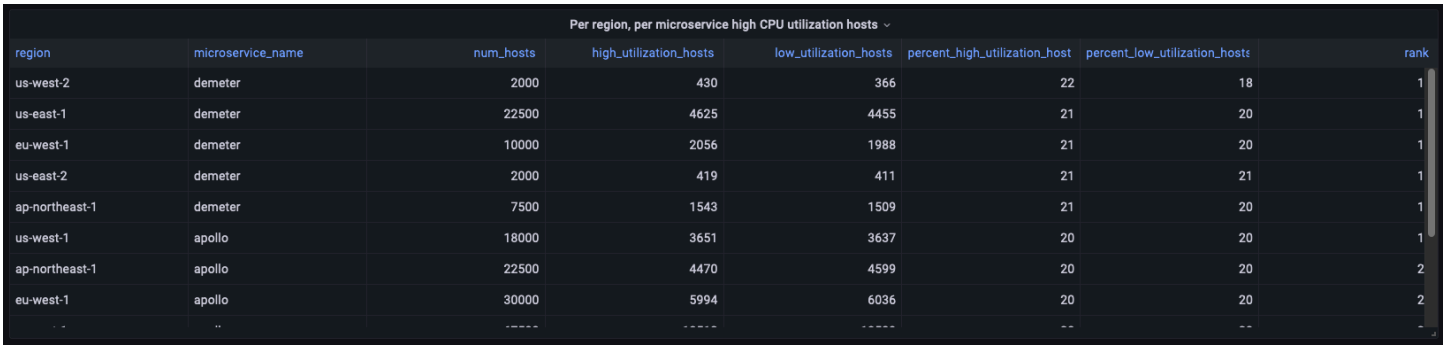
この例では、複数のダッシュボードパネルに類似情報のバリエーション (高いCPUホストとCPU使用率の高いフリートの割合を見つける) が表示され、同じスケジュールされたクエリを使用して結果を事前計算し、複数のパネルに入力する方法を示します。この再利用により、パネルごとに異なるスケジュールされたクエリを使用する代わりに、所有者のみを使用するコストがさらに最適化されます。

生データを含むダッシュボードパネル

CPU マイクロサービスあたりのリージョンあたりの使用率

最初のパネルは、リージョン、セル、サイロ、アベイラビリティーゾーン、マイクロサービス内の特定のデプロイについて、平均CPU使用率が上記のCPU使用率を下回るか上回るしきい値を持つインスタンスを計算します。次に、使用率の高いホストの割合が最も多いリージョンとマイクロサービスをソートします。これにより、特定のデプロイのサーバーがどの程度ホットであるかを特定し、その後ドリルダウンして問題をよりよく理解できます。

パネルのクエリは、一般的なテーブル式、ウィンドウ関数、結合などの複雑な分析タスクを がLiveAnalyticsSQLサポートする Timestream の柔軟性を示しています。



Per region, per microservice high CPU utilization hosts

region	microservice_name	num_hosts	high_utilization_hosts	low_utilization_hosts	percent_high_utilization_host	percent_low_utilization_hosts	rank
us-west-2	demeter	2000	430	366	22	18	1
us-east-1	demeter	22500	4625	4455	21	20	1
eu-west-1	demeter	10000	2056	1988	21	20	1
us-east-2	demeter	2000	419	411	21	21	1
ap-northeast-1	demeter	7500	1543	1509	21	20	1
us-west-1	apollo	18000	3651	3637	20	20	1
ap-northeast-1	apollo	22500	4470	4599	20	20	2
eu-west-1	apollo	30000	5994	6036	20	20	2

クエリ：

```
WITH microservice_cell_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, AVG(cpu_user) AS
 microservice_avg_metric
    FROM "raw_data"."devops"
    WHERE time BETWEEN from_milliseconds(1636526593876) AND
 from_milliseconds(1636612993876)
        AND measure_name = 'metrics'
    GROUP BY region, cell, silo, availability_zone, microservice_name
), instance_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, instance_name,
        AVG(cpu_user) AS instance_avg_metric
    FROM "raw_data"."devops"
    WHERE time BETWEEN from_milliseconds(1636526593876) AND
 from_milliseconds(1636612993876)
        AND measure_name = 'metrics'
    GROUP BY region, cell, silo, availability_zone, microservice_name, instance_name
), instances_above_threshold AS (
  SELECT i.*,
    CASE WHEN i.instance_avg_metric > (1 + 0.2) * m.microservice_avg_metric THEN 1 ELSE
 0 END AS high_utilization,
    CASE WHEN i.instance_avg_metric < (1 - 0.2) * m.microservice_avg_metric THEN 1 ELSE
 0 END AS low_utilization
  FROM instance_avg i INNER JOIN microservice_cell_avg m
    ON i.region = m.region AND i.cell = m.cell AND i.silo = m.silo AND
 i.availability_zone = m.availability_zone
      AND m.microservice_name = i.microservice_name
), per_deployment_high AS (
SELECT region, microservice_name, COUNT(*) AS num_hosts, SUM(high_utilization) AS
 high_utilization_hosts, SUM(low_utilization) AS low_utilization_hosts,
    ROUND(SUM(high_utilization) * 100.0 / COUNT(*), 0) AS
 percent_high_utilization_hosts,
    ROUND(SUM(low_utilization) * 100.0 / COUNT(*), 0) AS percent_low_utilization_hosts
```

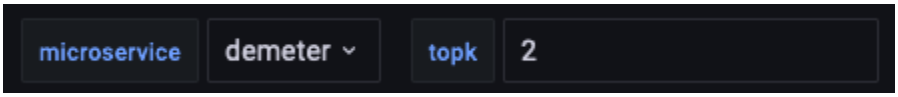
```
FROM instances_above_threshold
GROUP BY region, microservice_name
), per_region_ranked AS (
    SELECT *,
        DENSE_RANK() OVER (PARTITION BY region ORDER BY percent_high_utilization_hosts
 DESC, high_utilization_hosts DESC) AS rank
    FROM per_deployment_high
)
SELECT *
FROM per_region_ranked
WHERE rank <= 2
ORDER BY percent_high_utilization_hosts desc, rank asc
```

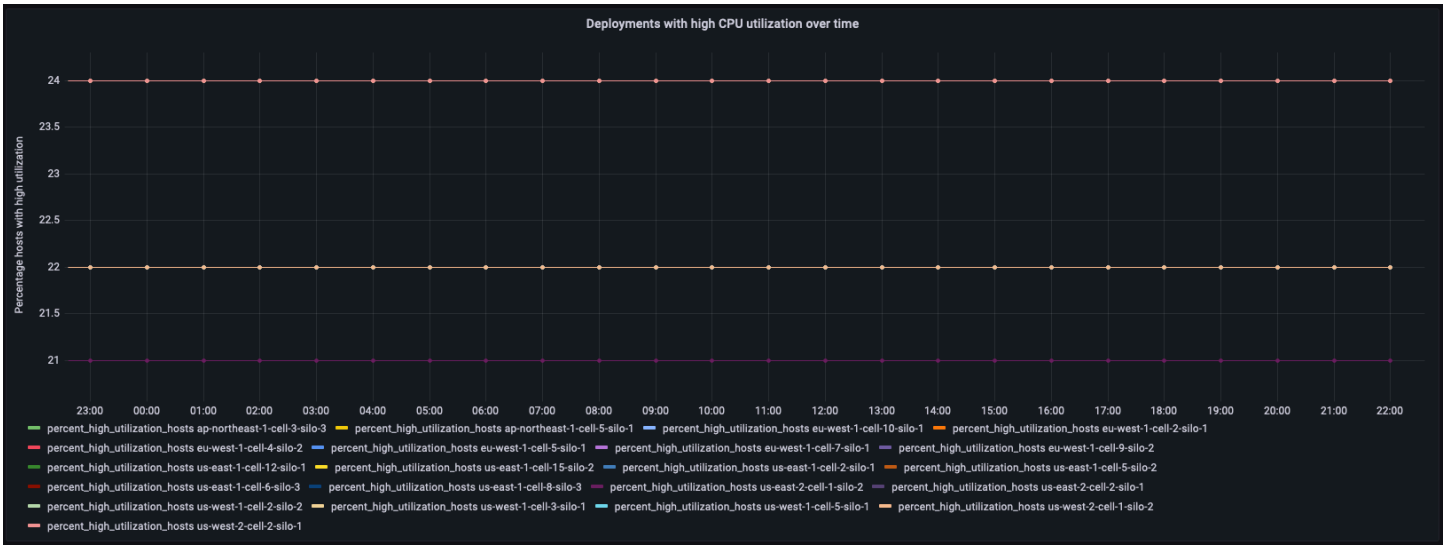
マイクロサービスにドリルダウンしてホットスポットを見つける

次のダッシュボードでは、マイクロサービスの 1 つをより深く掘り下げて、そのマイクロサービスの特定のリージョン、セル、サイロがCPU、使用率が高いフリートの何パーセントを実行しているかを調べることができます。例えば、フリート全体のダッシュボードでは、上位数位にマイクロサービスデメータが表示されていたため、このダッシュボードでは、そのマイクロサービスをより深く掘り下げたいと考えています。

このダッシュボードでは、変数を使用してドリルダウンするマイクロサービスを選択し、変数の値はディメンションの一意の値を使用して入力されます。マイクロサービスを選択すると、ダッシュボードの残りの部分が更新されます。

以下に示すように、最初のパネルは、デプロイ (マイクロサービスのリージョン、セル、サイロ) 内のホストの割合を経時的にプロットし、対応するクエリをダッシュボードのプロットに使用します。このプロット自体は、 が高いホストの割合が高い特定のデプロイを識別しますCPU。





クエリ：

```
WITH microservice_cell_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, bin(time, 1h) as
 hour, AVG(cpu_user) AS microservice_avg_metric
    FROM "raw_data"."devops"
    WHERE time BETWEEN from_milliseconds(1636526898831) AND
 from_milliseconds(1636613298831)
        AND measure_name = 'metrics'
        AND microservice_name = 'demeter'
    GROUP BY region, cell, silo, availability_zone, microservice_name, bin(time, 1h)
), instance_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, instance_name,
 bin(time, 1h) as hour,
        AVG(cpu_user) AS instance_avg_metric
    FROM "raw_data"."devops"
    WHERE time BETWEEN from_milliseconds(1636526898831) AND
 from_milliseconds(1636613298831)
        AND measure_name = 'metrics'
        AND microservice_name = 'demeter'
    GROUP BY region, cell, silo, availability_zone, microservice_name, instance_name,
 bin(time, 1h)
), instances_above_threshold AS (
  SELECT i.*,
    CASE WHEN i.instance_avg_metric > (1 + 0.2) * m.microservice_avg_metric THEN 1 ELSE
 0 END AS high_utilization
  FROM instance_avg i INNER JOIN microservice_cell_avg m
    ON i.region = m.region AND i.cell = m.cell AND i.silo = m.silo AND
 i.availability_zone = m.availability_zone
```

```
      AND m.microservice_name = i.microservice_name AND m.hour = i.hour
), high_utilization_percent AS (
    SELECT region, cell, silo, microservice_name, hour, COUNT(*) AS num_hosts,
 SUM(high_utilization) AS high_utilization_hosts,
        ROUND(SUM(high_utilization) * 100.0 / COUNT(*), 0) AS
 percent_high_utilization_hosts
    FROM instances_above_threshold
    GROUP BY region, cell, silo, microservice_name, hour
), high_utilization_ranked AS (
    SELECT region, cell, silo, microservice_name,
        DENSE_RANK() OVER (PARTITION BY region ORDER BY
 AVG(percent_high_utilization_hosts) desc, AVG(high_utilization_hosts) desc) AS rank
    FROM high_utilization_percent
    GROUP BY region, cell, silo, microservice_name
)
SELECT hup.silo, CREATE_TIME_SERIES(hour, hup.percent_high_utilization_hosts) AS
 percent_high_utilization_hosts
FROM high_utilization_percent hup INNER JOIN high_utilization_ranked hur
    ON hup.region = hur.region AND hup.cell = hur.cell AND hup.silo = hur.silo AND
 hup.microservice_name = hur.microservice_name
WHERE rank <= 2
GROUP BY hup.region, hup.cell, hup.silo
ORDER BY hup.silo
```

再利用を可能にする単一のスケジュールされたクエリへの変換

2 つのダッシュボードの異なるパネル間で同様の計算が行われることに注意してください。パネルごとに個別のスケジュールされたクエリを定義できます。ここでは、3 つのパネルをすべてレンダリングするために結果を使用できる 1 つのスケジュールされたクエリを定義することで、コストをさらに最適化する方法を説明します。

以下は、計算され、すべての異なるパネルに使用される集計をキャプチャするクエリです。このスケジュールされたクエリの定義には、いくつかの重要な側面があります。

- スケジュールされたクエリでサポートされるSQL表面積の柔軟性とパワー。一般的なテーブル式、結合、ケースステートメントなどを使用できます。
- スケジュールされたクエリを 1 つ使用して、特定のダッシュボードが必要とするよりも細かく統計を計算できます。また、ダッシュボードがさまざまな変数に使用するすべての値に対して統計を計算できます。例えば、集計はリージョン、セル、サイロ、マイクロサービス全体で計算されます。したがって、これらを組み合わせて、リージョンレベル、またはリージョン、マイクロサービスレベルの集計を作成できます。同様に、同じクエリは、すべてのリージョン、セル、サイロ、マ

イクロサービスの集計を計算します。これにより、これらの列にフィルターを適用して、値のサブセットの集計を取得できます。例えば、us-east-1 など、任意の 1 つのリージョンの集計を計算したり、任意の 1 つのマイクロサービスで、リージョン、セル、サイロ、マイクロサービス内の特定のデプロイを計測またはドリルダウンしたりできます。このアプローチにより、事前に計算された集計を維持するコストがさらに最適化されます。

```
WITH microservice_cell_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, bin(time, 1h) as
 hour, AVG(cpu_user) AS microservice_avg_metric
    FROM raw_data.devops
    WHERE time BETWEEN bin(@scheduled_runtime, 1h) - 1h AND bin(@scheduled_runtime, 1h)
 + 1h
        AND measure_name = 'metrics'
    GROUP BY region, cell, silo, availability_zone, microservice_name, bin(time, 1h)

), instance_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, instance_name,
 bin(time, 1h) as hour,
        AVG(cpu_user) AS instance_avg_metric
   FROM raw_data.devops
   WHERE time BETWEEN bin(@scheduled_runtime, 1h) - 1h AND bin(@scheduled_runtime, 1h)
 + 1h
       AND measure_name = 'metrics'
   GROUP BY region, cell, silo, availability_zone, microservice_name, instance_name,
 bin(time, 1h)
), instances_above_threshold AS (
    SELECT i.*,
        CASE WHEN i.instance_avg_metric > (1 + 0.2) * m.microservice_avg_metric THEN 1
 ELSE 0 END AS high_utilization,
        CASE WHEN i.instance_avg_metric < (1 - 0.2) * m.microservice_avg_metric THEN 1
 ELSE 0 END AS low_utilization
   FROM instance_avg i INNER JOIN microservice_cell_avg m
       ON i.region = m.region AND i.cell = m.cell AND i.silo = m.silo AND
 i.availability_zone = m.availability_zone
           AND m.microservice_name = i.microservice_name AND m.hour = i.hour
)
SELECT region, cell, silo, microservice_name, hour,
     COUNT(*) AS num_hosts, SUM(high_utilization) AS high_utilization_hosts,
 SUM(low_utilization) AS low_utilization_hosts
FROM instances_above_threshold GROUP BY region, cell, silo, microservice_name, hour
```

以下は、前のクエリのスケジュールされたクエリ定義です。スケジュール式は、30 分ごとに更新するように設定され、bin(@scheduled_runtime、1h) 構造を使用して最大 1 時間前にデータを更新します。アプリケーションの鮮度要件に応じて、頻繁に更新するように設定できます。time WHERE BETWEEN bin(@scheduled_runtime, 1h) - 1h AND bin(@scheduled_runtime, 1h) + 1h を使用すると、15 分に 1 回更新しても、現在の時刻と前の時刻の 1 時間分のデータを取得できます。

後で、3 つのパネルがテーブル deployment_cpu_stats_per_hr に書き込まれたこれらの集計を使用して、パネルに関連するメトリクスを視覚化する方法を説明します。

```
{
    "Name": "MultiPT30mHighCpuDeploymentsPerHr",
    "QueryString": "WITH microservice_cell_avg AS (    SELECT region, cell,
 silo, availability_zone, microservice_name, bin(time, 1h) as hour, AVG(cpu_user)
 AS microservice_avg_metric    FROM raw_data.devops    WHERE time BETWEEN
 bin(@scheduled_runtime, 1h) - 1h AND bin(@scheduled_runtime, 1h) + 1h    AND
 measure_name = 'metrics'    GROUP BY region, cell, silo, availability_zone,
 microservice_name, bin(time, 1h)    ), instance_avg AS (    SELECT region,
 cell, silo, availability_zone, microservice_name, instance_name, bin(time, 1h)
 as hour,    AVG(cpu_user) AS instance_avg_metric    FROM raw_data.devops
 WHERE time BETWEEN bin(@scheduled_runtime, 1h) - 1h AND bin(@scheduled_runtime,
 1h) + 1h    AND measure_name = 'metrics'    GROUP BY region, cell, silo,
 availability_zone, microservice_name, instance_name, bin(time, 1h)    ),
 instances_above_threshold AS (    SELECT i.*,    CASE WHEN i.instance_avg_metric >
 (1 + 0.2) * m.microservice_avg_metric THEN 1 ELSE 0 END AS high_utilization,    CASE
 WHEN i.instance_avg_metric < (1 - 0.2) * m.microservice_avg_metric THEN 1 ELSE 0 END
 AS low_utilization    FROM instance_avg i INNER JOIN microservice_cell_avg m    ON
 i.region = m.region AND i.cell = m.cell AND i.silo = m.silo AND i.availability_zone
 = m.availability_zone    AND m.microservice_name = i.microservice_name AND m.hour =
 i.hour    )    SELECT region, cell, silo, microservice_name, hour,         COUNT(*)
 AS num_hosts, SUM(high_utilization) AS high_utilization_hosts, SUM(low_utilization) AS
 low_utilization_hosts    FROM instances_above_threshold GROUP BY region, cell, silo,
 microservice_name, hour",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/30 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
```

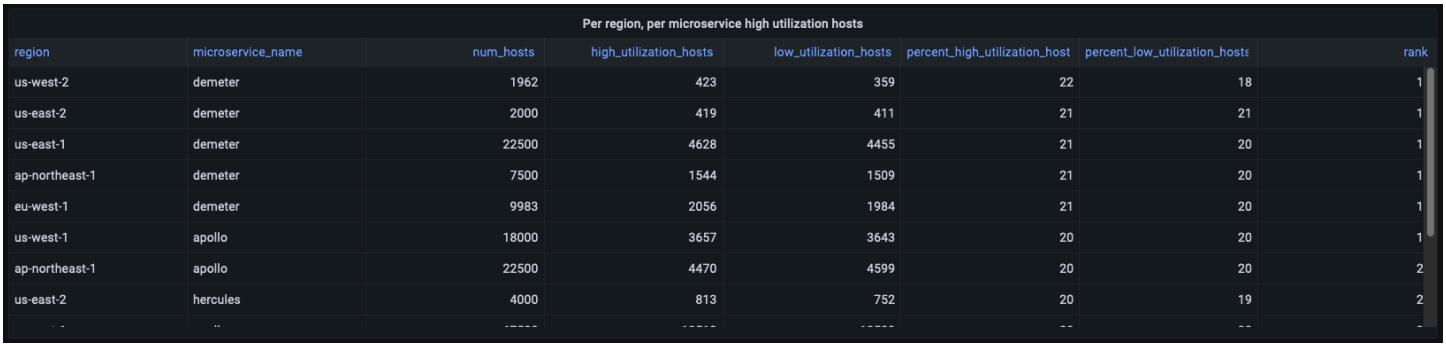
```
            "TableName": "deployment_cpu_stats_per_hr",
            "TimeColumn": "hour",
            "DimensionMappings": [
                {
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "cell",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "silo",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "microservice_name",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "cpu_user",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "num_hosts",
                        "MeasureValueType": "BIGINT"
                    },
                    {
                        "SourceColumn": "high_utilization_hosts",
                        "MeasureValueType": "BIGINT"
                    },
                    {
                        "SourceColumn": "low_utilization_hosts",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
```

```
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

事前に計算された結果からのダッシュボード

高CPU使用率ホスト

使用率の高いホストの場合、異なるパネルが deployment_cpu_stats_per_hr のデータを使用して、パネルに必要な異なる集計を計算する方法を確認できます。例えば、このパネルはリージョンレベルの情報を提供するため、リージョンやマイクロサービスをフィルタリングすることなく、リージョンやマイクロサービス別にグループ化された集計を報告します。



Per region, per microservice high utilization hosts

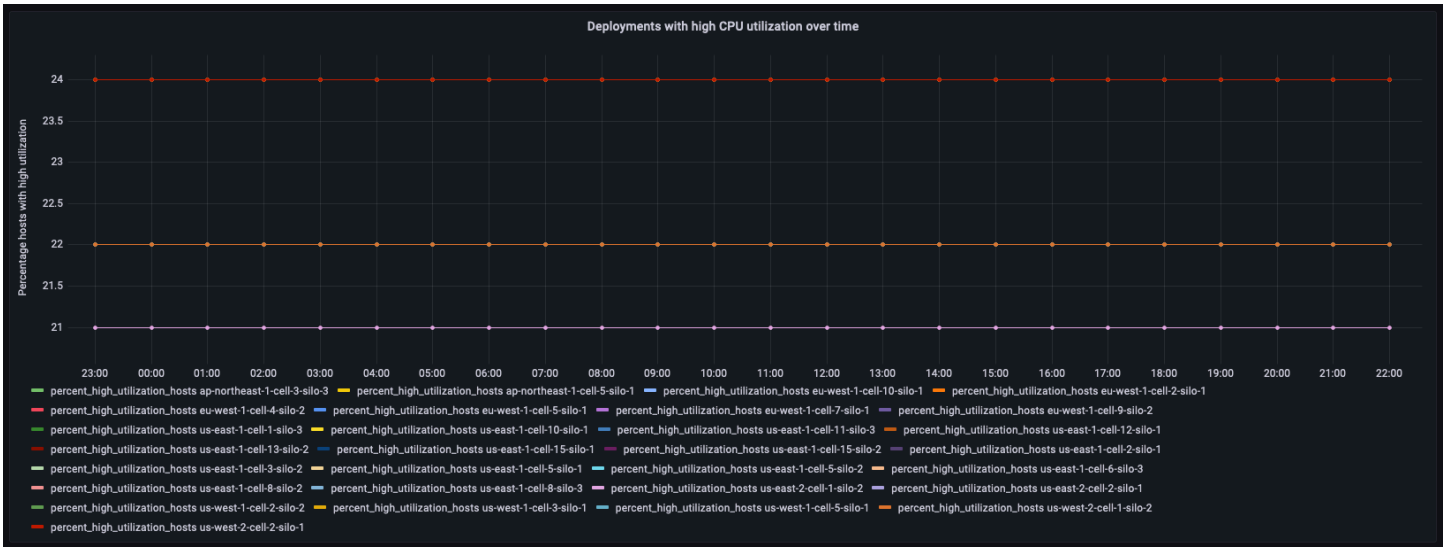
region	microservice_name	num_hosts	high_utilization_hosts	low_utilization_hosts	percent_high_utilization_host	percent_low_utilization_hosts	rank
us-west-2	demeter	1962	423	359	22	18	1
us-east-2	demeter	2000	419	411	21	21	1
us-east-1	demeter	22500	4628	4455	21	20	1
ap-northeast-1	demeter	7500	1544	1509	21	20	1
eu-west-1	demeter	9983	2056	1984	21	20	1
us-west-1	apollo	18000	3657	3643	20	20	1
ap-northeast-1	apollo	22500	4470	4599	20	20	2
us-east-2	hercules	4000	813	752	20	19	2

```
WITH per_deployment_hosts AS (
    SELECT region, cell, silo, microservice_name,
        AVG(num_hosts) AS num_hosts,
        AVG(high_utilization_hosts) AS high_utilization_hosts,
        AVG(low_utilization_hosts) AS low_utilization_hosts
    FROM "derived"."deployment_cpu_stats_per_hr"
    WHERE time BETWEEN from_milliseconds(1636567785437) AND
 from_milliseconds(1636654185437)
        AND measure_name = 'cpu_user'
    GROUP BY region, cell, silo, microservice_name
), per_deployment_high AS (
    SELECT region, microservice_name,
        SUM(num_hosts) AS num_hosts,
        ROUND(SUM(high_utilization_hosts), 0) AS high_utilization_hosts,
        ROUND(SUM(low_utilization_hosts),0) AS low_utilization_hosts,
        ROUND(SUM(high_utilization_hosts) * 100.0 / SUM(num_hosts)) AS
 percent_high_utilization_hosts,
        ROUND(SUM(low_utilization_hosts) * 100.0 / SUM(num_hosts)) AS
 percent_low_utilization_hosts
    FROM per_deployment_hosts
```

```
    GROUP BY region, microservice_name
),
per_region_ranked AS (
    SELECT *,
        DENSE_RANK() OVER (PARTITION BY region ORDER BY percent_high_utilization_hosts
 DESC, high_utilization_hosts DESC) AS rank
    FROM per_deployment_high
)
SELECT *
FROM per_region_ranked
WHERE rank <= 2
ORDER BY percent_high_utilization_hosts desc, rank asc
```

マイクロサービスにドリルダウンして、高CPU使用率のデプロイを見つける

次の例では、再び deployment_cpu_stats_per_hr 派生テーブルを使用しますが、特定のマイクロサービス (この例では、集計ダッシュボードで高使用率ホストが報告されているため、この例の深さ) にフィルターを適用します。このパネルは、高CPU使用率ホストの割合を経時的に追跡します。




```
WITH high_utilization_percent AS (
    SELECT region, cell, silo, microservice_name, bin(time, 1h) AS hour, MAX(num_hosts)
 AS num_hosts,
        MAX(high_utilization_hosts) AS high_utilization_hosts,
        ROUND(MAX(high_utilization_hosts) * 100.0 / MAX(num_hosts)) AS
 percent_high_utilization_hosts
    FROM "derived"."deployment_cpu_stats_per_hr"
    WHERE time BETWEEN from_milliseconds(1636525800000) AND
 from_milliseconds(1636612200000)
        AND measure_name = 'cpu_user'
```

```
        AND microservice_name = 'demeter'
    GROUP BY region, cell, silo, microservice_name, bin(time, 1h)
), high_utilization_ranked AS (
    SELECT region, cell, silo, microservice_name,
        DENSE_RANK() OVER (PARTITION BY region ORDER BY
 AVG(percent_high_utilization_hosts) desc, AVG(high_utilization_hosts) desc) AS rank
    FROM high_utilization_percent
    GROUP BY region, cell, silo, microservice_name
)
SELECT hup.silo, CREATE_TIME_SERIES(hour, hup.percent_high_utilization_hosts) AS
 percent_high_utilization_hosts
FROM high_utilization_percent hup INNER JOIN high_utilization_ranked hur
    ON hup.region = hur.region AND hup.cell = hur.cell AND hup.silo = hur.silo AND
 hup.microservice_name = hur.microservice_name
WHERE rank <= 2
GROUP BY hup.region, hup.cell, hup.silo
ORDER BY hup.silo
```

ベーステーブルのクエリとスケジュールされたクエリ結果のクエリの比較

この Timestream クエリ例では、次のスキーマ、クエリの例、出力を使用して、ベーステーブルのクエリとスケジュールされたクエリ結果の派生テーブルのクエリを比較します。適切に計画されたスケジュールされたクエリを使用すると、元のベーステーブルで可能になるよりも高速なクエリにつながる可能性のある行やその他の特性の派生テーブルを取得できます。

このシナリオを説明するビデオについては、「 の Amazon Timestream でスケジュールされたクエリを使用して、クエリのパフォーマンスを向上させ、コストを削減する LiveAnalytics」を参照してください。

この例では、次のシナリオを使用します。

- リージョン – us-east-1
- ベーステーブル – `"clickstream"."shopping"`
- 派生テーブル – `"clickstream"."aggregate"`

ベーステーブル

ベーステーブルのスキーマを以下に示します。

列	タイプ	LiveAnalytics 属性タイプのタイムストリーム
チャンネル	varchar	MULTI
説明	varchar	MULTI
イベント	varchar	DIMENSION
ip_address	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
product	varchar	MULTI
product_id	varchar	MULTI
数量	double	MULTI
query	varchar	MULTI
session_id	varchar	DIMENSION
user_group	varchar	DIMENSION
user_id	varchar	DIMENSION

ベーステーブルの測定値を次に示します。ベーステーブルは、スケジュールされたクエリが実行される Timestream のテーブルを指します。

- measure_name – `metrics`
- データ – マルチ
- ディメンション ：

```
[ ( user_group, varchar ),( user_id, varchar ),( session_id, varchar ),( ip_address,
 varchar ),( event, varchar ) ]
```

ベーステーブルでのクエリ

以下は、特定の時間範囲で 5 分間の集計でカウントを収集するアドホッククエリです。

```
SELECT BIN(time, 5m) as time,
channel,
product_id,
SUM(quantity) as product_quantity
FROM "clickstream"."shopping"
WHERE BIN(time, 5m) BETWEEN '2023-05-11 10:10:00.000000000' AND '2023-05-11
 10:30:00.000000000'
AND channel = 'Social media'
and product_id = '431412'
GROUP BY BIN(time, 5m),channel,product_id
```

出力:

```
duration:1.745 sec
Bytes scanned: 29.89 MB
Query Id: AEBQEANMHG7MHHBHCKJ3BSOE3QUGIDBGWCCP5I6J6YUW5CVJZ2M3JCJ27QRMM7A
Row count:5
```

スケジュールされたクエリ

以下は、5 分ごとに実行されるスケジュールされたクエリです。

```
SELECT BIN(time, 5m) as time, channel as measure_name, product_id, product,
SUM(quantity) as product_quantity
FROM "clickstream"."shopping"
WHERE time BETWEEN BIN(@scheduled_runtime, 5m) - 10m AND BIN(@scheduled_runtime, 5m) -
 5m
AND channel = 'Social media'
GROUP BY BIN(time, 5m), channel, product_id, product
```

派生テーブルのクエリ

以下は、派生テーブルのアドホッククエリです。派生テーブルは、スケジュールされたクエリの結果を含む Timestream テーブルを指します。

```
SELECT time, measure_name, product_id,product_quantity
FROM "clickstream"."aggregate"
```

```
WHERE time BETWEEN '2023-05-11 10:10:00.000000000' AND '2023-05-11 10:30:00.000000000'
AND measure_name = 'Social media'
and product_id = '431412'
```

出力:

```
duration: 0.2960 sec
Bytes scanned: 235.00 B
QueryID: AEBQEANMHHAAQU4FFTT6CFM6UYXTL4SMLZV22MFP4KV2Z7IRVOPLOMLDD6BR33Q
Row count: 5
```

比較

以下は、ベーステーブルのクエリと派生テーブルのクエリの結果の比較です。スケジュールされたクエリを通じて結果を集約した派生テーブルの同じクエリは、スキャンされたバイト数が少なくて、より速く完了します。

これらの結果は、スケジュールされたクエリを使用してデータを集約し、クエリを高速化する の値を示しています。

	ベーステーブルのクエリ	派生テーブルのクエリ
duration	1.745 秒	0.2960 秒
スキャンされたバイト数	29.89 MB	235 バイト
行数	5	5

UNLOAD を使用して、 の Timestream から S3 にクエリ結果をエクスポートする LiveAnalytics

Amazon Timestream for LiveAnalytics では、 UNLOADステートメントを使用して、費用対効果の高い安全な方法でクエリ結果を Amazon S3 にエクスポートできるようになりました。UNLOAD ステートメントを使用して、時系列データを Apache Parquet または Comma Separated Values (CSV) 形式で選択した S3 バケットにエクスポートできるようになりました。これにより、時系列データを他の のサービスで保存、結合、分析する柔軟性が得られます。UNLOAD ステートメントを使用すると、データを圧縮された方法でエクスポートできるため、転送されるデータと必要なストレージス

ペースが削減されます。 は、データをエクスポートするときに選択した属性に基づくパーティショニングUNLOADもサポートし、パフォーマンスを向上させ、データにアクセスするダウンストリームサービスの処理時間を短縮します。さらに、Amazon S3 マネージドキー (SSE-S3) または AWS Key Management Service (AWS KMS) マネージドキー (SSE-KMS) を使用して、エクスポートされたデータを暗号化できます。

Timestream UNLOADの利点 LiveAnalytics

UNLOAD ステートメントを使用する主な利点は次のとおりです。

- 運用の容易さ – UNLOADステートメントを使用すると、1 つのクエリリクエストで ギガバイトのデータを Apache Parquet または CSV 形式でエクスポートできるため、ダウンストリームの処理ニーズに最適な形式を柔軟に選択でき、データレイクの構築が容易になります。
- 安全で費用対効果が高い – UNLOADステートメントは、データを圧縮された方法で S3 バケットにエクスポートし、カスタマーマネージドキーを使用してデータを暗号化 (SSE- KMSまたは SSE_S3) し、データストレージコストを削減し、不正アクセスから保護する機能を提供します。
- パフォーマンス – UNLOADステートメントを使用して、S3 バケットにエクスポートするときにデータをパーティション化できます。データをパーティション化することで、ダウンストリームサービスはデータを並行して処理できるため、処理時間を短縮できます。さらに、ダウンストリームサービスは必要なデータのみを処理できるため、必要な処理リソースが減り、それに伴うコストも削減されます。

の Timestream UNLOADからの のユースケース LiveAnalytics

UNLOAD ステートメントを使用して、次の S3 バケットにデータを書き込みます。

- データウェアハウスの構築 – 数十ギガバイトのクエリ結果を S3 バケットにエクスポートし、時系列データをデータレイクに簡単に追加できます。Amazon Athena や Amazon Redshift などのサービスを使用して、時系列データと他の関連データを組み合わせて、複雑なビジネスインサイトを得ることができます。
- AI と ML データパイプラインの構築 – このUNLOADステートメントを使用すると、時系列データにアクセスする機械学習モデルのデータパイプラインを簡単に構築できるため、Amazon SageMaker や Amazon などのサービスで時系列データを簡単に使用できますEMR。
- ETL 処理の簡素化 - S3 バケットにデータをエクスポートすると、データに対して抽出、変換、ロード (ETL) オペレーションを実行するプロセスが簡素化され、 AWS Glue などのサードパーティーのツールや AWS サービスをシームレスに使用してデータを処理および変換できます。

UNLOAD 概念

構文

```
UNLOAD (SELECT statement)
 TO 's3://bucket-name/folder'
 WITH ( option = expression [, ...] )
```

ここでoption、 は です。

```
{ partitioned_by = ARRAY[ col_name[,…] ]
 | format = [ '{ CSV | PARQUET }' ]
 | compression = [ '{ GZIP | NONE }' ]
 | encryption = [ '{ SSE_KMS | SSE_S3 }' ]
 | kms_key = '<string>'
 | field_delimiter ='<character>'
 | escaped_by = '<character>'
 | include_header = ['{true, false}']
 | max_file_size = '<value>'
 | }
```

パラメータ

SELECT ステートメント

LiveAnalytics テーブルの 1 つ以上の Timestream からデータを選択および取得するために使用されるクエリステートメント。

```
(SELECT column 1, column 2, column 3 from database.table
      where measure_name = "ABC" and timestamp between ago (1d) and now() )
```

TO 句

```
TO 's3://bucket-name/folder'
```

または

```
TO 's3://access-point-alias/folder'
```

UNLOAD ステートメントの TO句は、クエリ結果の出力先を指定します。の Timestream LiveAnalytics が出力ファイルオブジェクトを書き込む Amazon S3 上のフォルダの場所を持つ Amazon S3 access-point-alias バケット名または Amazon S3 を含む完全なパスを指定する必要があります。S3 バケットは、同じアカウントと同じリージョンで所有されている必要があります。クエリ結果セットに加えて、 の Timestream はマニフェストファイルとメタデータファイルを指定された宛先フォルダに LiveAnalytics 書き込みます。

PARTITIONED_BY 句

```
partitioned_by = ARRAY [col_name[,…] , (default: none)
```

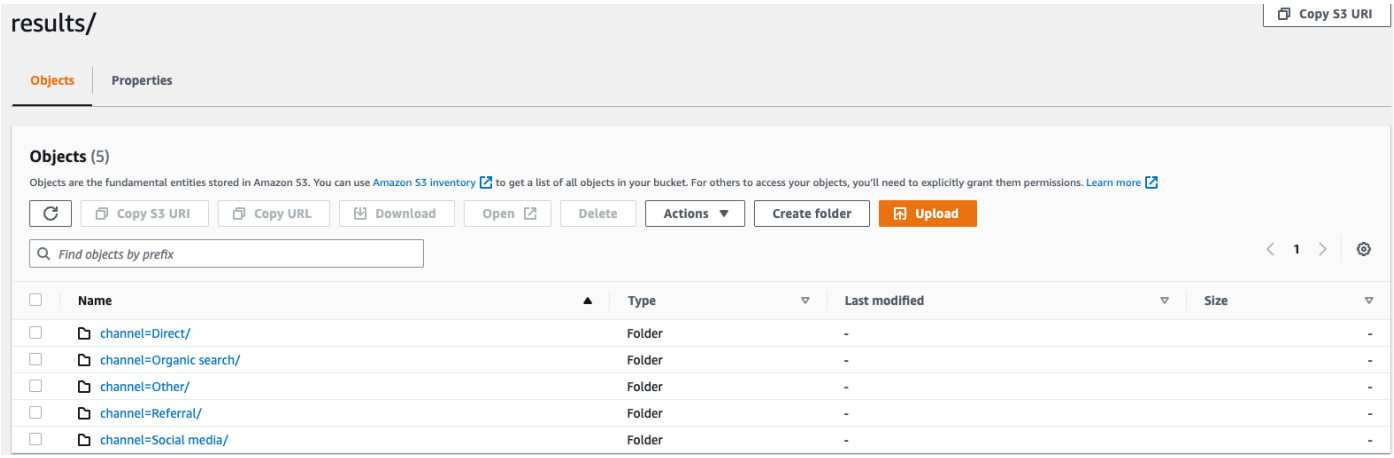
partitioned_by 句は、データをきめ細かなレベルでグループ化して分析するためのクエリに使用されます。クエリ結果を S3 バケットにエクスポートする場合、選択クエリの 1 つ以上の列に基づいてデータをパーティション化することを選択できます。データをパーティション化すると、エクスポートされたデータはパーティション列に基づいてサブセットに分割され、各サブセットは別のフォルダに保存されます。エクスポートされたデータを含む結果フォルダ内に、サブフォルダfolder/results/partition column = partition value/が自動的に作成されます。ただし、パーティション化された列は出力ファイルに含まれないことに注意してください。

partitioned_by は構文の必須句ではありません。パーティショニングなしでデータを エクスポートする場合は、構文で 句を除外できます。

Example

ウェブサイトのクリックストリームデータをモニタリングしていて、トラフィックのチャネルが direct、、Social Media、Organic SearchOther、 の 5 つあるとしますReferral。データを エクスポートするときは、列 を使用してデータをパーティション化することを選択できますChannel。データフォルダ 内にはs3://bucketname/results、それぞれがチャネル名を持つ 5 つのフォルダがあります。例えば、このフォルダs3://bucketname/results/channel=Social Media/.内には、Social Mediaチャネルを介してウェブサイトに進出したすべての顧客のデータが表示されます。同様に、残りのチャネルには他のフォルダがあります。

Channel 列でパーティション分割されたエクスポートされたデータ



FORMAT

```
format = [ '{ CSV | PARQUET }' , default: CSV
```

S3 バケットに書き込まれたクエリ結果の形式を指定するキーワード。データは、デフォルトの区切り文字としてカンマ (,CSV) を使用してカンマ区切り値 () としてエクスポートするか、分析用の効率的なオープン列ストレージ形式である Apache Parquet 形式でエクスポートできます。

COMPRESSION

```
compression = [ '{ GZIP | NONE }' ], default: GZIP
```

エクスポートされたデータは、圧縮アルゴリズムを使用して圧縮GZIPすることも、 NONEオプションを指定して圧縮解除することもできます。

ENCRYPTION

```
encryption = [ '{ SSE_KMS | SSE_S3 }' ], default: SSE_S3
```

Amazon S3 の出力ファイルは、選択した暗号化オプションを使用して暗号化されます。データに加えて、マニフェストファイルとメタデータファイルも、選択した暗号化オプションに基づいて暗号化されます。現在、SSE_S3 および SSE_KMS 暗号化をサポートしています。SSE_S3 は、Amazon S3 が 256 ビットの高度な暗号化標準 (AES) 暗号化を使用してデータを暗号化するサーバー側の暗号化です。SSE_KMS は、カスタマーマネージドキーを使用してデータを暗号化するサーバー側の暗号化です。

KMS_KEY

```
kms_key = '<string>'
```

KMS キーは、エクスポートされたクエリ結果を暗号化するためのカスタマー定義のキーです。KMS キーは AWS Key Management Service (AWS KMS) によって安全に管理され、Amazon S3 のデータファイルの暗号化に使用されます。

FIELD_DELIMITER

```
field_delimiter ='<character>' , default: (,)
```

データをCSV形式でエクスポートする場合、このフィールドは、パイプASCII文字 (|)、カンマ (,)、タブ (/t) などの出力ファイルのフィールドを区切るために使用される 1 つの文字を指定します。CSV ファイルのデフォルトの区切り文字はカンマ文字です。データ内の値に選択した区切り文字が含まれている場合、区切り文字は引用符で囲まれます。例えば、データ内の値に が含まれている場合`Time,stream`、この値はエクスポートされたデータ`"Time,stream"`内の として引用符で囲まれます。Timestream が に使用する引用符文字 LiveAnalytics は、二重引用符 (「) です。

にヘッダーを含める`FIELD_DELIMITER`場合は、キャリッジリターン文字 (ASCII 13、16 進0D数 、テキスト '\r') または改行文字 (ASCII 10、16 進数 0A、テキスト '\n') を として指定しないでください。これによりCSV、多くのパーサーが結果のCSV出力でヘッダーを正しく解析できなくなるためです。

ESCAPED_BY

```
escaped_by = '<character>', default: (\)
```

データを CSV 形式でエクスポートする場合、このフィールドは S3 バケットに書き込まれたデータファイルでエスケープ文字として扱う文字を指定します。エスケープは、次のシナリオで発生します。

1. 値自体に引用符文字 (「) が含まれている場合、エスケープ文字を使用してエスケープされます。例えば、値が でTime"stream、 (\) が設定されたエスケープ文字である場合、エスケープされます`Time\"stream`。
2. 値に設定されたエスケープ文字が含まれている場合、エスケープされます。例えば、値が の場合`Time\stream`、 としてエスケープされます`Time\\stream`。

> **Note**
>
> エクスポートされた出力に配列、行、時系列などの複雑なデータ型が含まれている場合、JSON文字列としてシリアル化されます。次に例を示します。

データ型	実際の値	値がエスケープされるCSV形式 [シリアル化されたJSON文字列〕
配列	`[ 23,24,25 ]`	`"[23,24,25]"`
行	`( x=23.0, y=hello )`	`"{\"x\":23.0,\"y\":\"hello\"}"`
時系列	`[ ( time=1970-01-01 00:00:00.000000010 , value=100.0 ), ( time=1970-01-01 00:00:00.000000012, value=120.0 ) ]`	`"[{\"time\":\"1970-01-01 00:00:00.000000010Z\",\"value\":100.0},{\"time\":\"1970-01-01 00:00:00.000000012Z\",\"value\":120.0}]"`

INCLUDE_HEADER

```
include_header = 'true' , default: 'false'
```

CSV データを形式でエクスポートする場合、このフィールドでは、エクスポートされたCSVデータファイルの最初の行として列名を含めることができます。

使用できる値は「true」と「false」で、デフォルト値は「false」です。escaped_by や などのテキスト変換オプションは、ヘッダーにもfield_delimiter適用されます。

> **Note**
>
> ヘッダーを含める場合、キャリッジリターン文字 (ASCII 13、16 進 0D、テキスト '\r') または改行文字 (ASCII 10、16 進 0A、テキスト '\n') を として選択しないことが重要です。これによりFIELD_DELIMITER、多くのパーサーが結果のCSV出力でヘッダーを正しく解析できなくなるためです。

MAX_FILE_SIZE

```
max_file_size = 'X[MB|GB]' , default: '78GB'
```

このフィールドは、UNLOADステートメントが Amazon S3 で作成するファイルの最大サイズを指定します。UNLOAD ステートメントは複数のファイルを作成できますが、Amazon S3 に書き込まれる各ファイルの最大サイズは、このフィールドで指定されたサイズとほぼ同じになります。

フィールドの値は 16 MB から 78 GB の間でなければなりません。などの整数、または 12GBなどの小数で指定できます0.5GB24.7MB。デフォルト値は 78 GB です。

実際のファイルサイズは、ファイルが書き込まれるときに概算されるため、実際の最大サイズが指定した数と完全に等しくない場合があります。

S3 バケットに書き込まれる内容

クエリが正常に実行されるたびにUNLOAD、 の Timestream はクエリ結果、メタデータファイル、マニフェストファイルを S3 バケットに LiveAnalytics 書き込みます。データをパーティション分割した場合は、結果フォルダにすべてのパーティションフォルダがあります。マニフェストファイルには、 UNLOAD コマンドによって書き込まれたファイルのリストが含まれています。メタデータファイルには、書き込まれたデータの特性、プロパティ、属性を説明する情報が含まれています。

エクスポートされたファイル名は何ですか？

エクスポートされたファイル名には 2 つのコンポーネントが含まれており、最初のコンポーネントは queryID、2 番目のコンポーネントは一意の識別子です。

CSV ファイル

```
S3://bucket_name/results/<queryid>_<UUID>.csv
S3://bucket_name/results/<partitioncolumn>=<partitionvalue>/<queryid>_<UUID>.csv
```

圧縮CSVファイル

```
S3://bucket_name/results/<partitioncolumn>=<partitionvalue>/<queryid>_<UUID>.gz
```

Parquet ファイル

```
S3://bucket_name/results/<partitioncolumn>=<partitionvalue>/<queryid>_<UUID>.parquet
```

メタデータファイルとマニフェストファイル

```
S3://bucket_name/<queryid>_<UUID>_manifest.json
S3://bucket_name/<queryid>_<UUID>_metadata.json
```

CSV 形式のデータはファイルレベルで保存されるため、S3 にエクスポートするときにデータを圧縮すると、ファイルには「.gz」拡張子が付きます。ただし、Parquet のデータは列レベルで圧縮されるため、エクスポート中にデータを圧縮しても、ファイルには .parquet 拡張子が残ります。

各ファイルにはどのような情報が含まれていますか？

マニフェストファイル

マニフェストファイルは、UNLOAD実行時にエクスポートされるファイルのリストに関する情報を提供します。マニフェストファイルは、 というファイル名で、提供された S3 バケットで使用できますs3://<bucket_name>/<queryid>_<UUID>_manifest.json。マニフェストファイルには、結果フォルダ内のファイルの URL、各ファイルのレコード数とサイズ、およびクエリメタデータ (クエリのために S3 にエクスポートされた合計バイト数と合計行数) が含まれます。

```
{
  "result_files": [
    {
        "url":"s3://my_timestream_unloads/ec2_metrics/
AEDAGANLHLBH4OLISD3CVOZZRWPX5GV2XCXRBKCVD554N6GWPWWXBP7LSG74V2Q_1448466917_szCL4YgVYzGXj2lS.gz"
        "file_metadata":
            {
                "content_length_in_bytes": 32295,
                "row_count": 10
            }
    },
    {
        "url":"s3://my_timestream_unloads/ec2_metrics/
AEDAGANLHLBH4OLISD3CVOZZRWPX5GV2XCXRBKCVD554N6GWPWWXBP7LSG74V2Q_1448466917_szCL4YgVYzGXj2lS.gz"
        "file_metadata":
            {
                "content_length_in_bytes": 62295,
                "row_count": 20
            }
    },
  ],
  "query_metadata":
    {
```

```
      "content_length_in_bytes": 94590,
      "total_row_count": 30,
      "result_format": "CSV",
      "result_version": "Amazon Timestream version 1.0.0"
    },
  "author": {
        "name": "Amazon Timestream",
        "manifest_file_version": "1.0"
  }
}
```

メタデータ

メタデータファイルには、列名、列タイプ、スキーマなどのデータセットに関する追加情報が表示されます。メタデータファイルは、提供された S3 バケットでファイル名 S3://bucket_name/<queryid>_<UUID>_metadata.json で使用できます。

メタデータファイルの例を次に示します。

```
{
    "ColumnInfo": [
        {
            "Name": "hostname",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "region",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "measure_name",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "cpu_utilization",
            "Type": {
                "TimeSeriesMeasureValueColumnInfo": {
```

```
                    "Type": {
                        "ScalarType": "DOUBLE"
                    }
                }
            }
        }
    ],
    "Author": {
        "Name": "Amazon Timestream",
        "MetadataFileVersion": "1.0"
    }
}
```

メタデータファイルで共有される列情報は、SELECTクエリのクエリAPIレスポンス
でColumnInfo送信されるものと同じ構造です。

結果

結果フォルダには、エクスポートしたデータが Apache Parquet または CSV形式で格納されます。

例

UNLOAD クエリ を介して次のようなクエリを送信するとAPI、

```
UNLOAD(SELECT user_id, ip_address, event, session_id, measure_name, time, query,
 quantity, product_id, channel
                    FROM sample_clickstream.sample_shopping WHERE time BETWEEN ago(2d)
 AND now())
                TO 's3://my_timestream_unloads/withoutpartition/' WITH ( format='CSV',
 compression='GZIP')
```

UNLOAD クエリレスポンスには 1 行 x 3 列があります。これら 3 つの列は次のとおりです。

- タイプの行 BIGINT - エクスポートされた行数を示します
- metadataFile タイプ VARCHAR - エクスポートされたURIメタデータファイルの S3
- manifestFile タイプ VARCHAR - エクスポートされたURIマニフェストファイルの S3

クエリ から次のレスポンスが表示されますAPI。

```
{
    "Rows": [
```

```
        {
            "Data": [
                {
                    "ScalarValue": "20" # No of rows in output across all files
                },
                {
                    "ScalarValue": "s3://my_timestream_unloads/withoutpartition/
AEDAAANGH3D7FYHOBQGQQMEAISCJ45B42OWWJMOT4N6RRJICZUA7R25VYVOHJIY_<UUID>_metadata.json"
 #Metadata file
                },
                {
                    "ScalarValue": "s3://my_timestream_unloads/withoutpartition/
AEDAAANGH3D7FYHOBQGQQMEAISCJ45B42OWWJMOT4N6RRJICZUA7R25VYVOHJIY_<UUID>_manifest.json"
 #Manifest file
                }
            ]
        }
    ],
    "ColumnInfo": [
        {
            "Name": "rows",
            "Type": {
                "ScalarType": "BIGINT"
            }
        },
        {
            "Name": "metadataFile",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "manifestFile",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        }
    ],
    "QueryId": "AEDAAANGH3D7FYHOBQGQQMEAISCJ45B42OWWJMOT4N6RRJICZUA7R25VYVOHJIY",
    "QueryStatus": {
        "ProgressPercentage": 100.0,
        "CumulativeBytesScanned": 1000,
        "CumulativeBytesMetered": 10000000
    }
```

```
}
```

データ型

UNLOAD ステートメントは、 timeおよび サポートされているデータ型を除く で説明されている LiveAnalyticsクエリ言語のすべてのデータタイプの Timestream をサポートしますunknown。

の Timestream UNLOADからの の前提条件 LiveAnalytics

以下は、 の Timestream UNLOADから を使用して S3 にデータを書き込むための前提条件です LiveAnalytics。

- UNLOAD コマンドで使用する LiveAnalytics テーブルの Timestream (複数可) からデータを読み取るアクセス許可が必要です。
- リソースの Timestream LiveAnalytics と同じ AWS リージョンに Amazon S3 バケットが必要です。
- 選択した S3 バケットについて、S3 バケットポリシーに Timestream がデータをエクスポートすることを許可 LiveAnalytics するアクセス許可も付与されていることを確認します。
- UNLOAD クエリの実行に使用される認証情報には、 の Timestream が S3 にデータを LiveAnalytics 書き込めるようにするために必要な AWS Identity and Access Management (IAM) アクセス許可が必要です。ポリシーの例は次のとおりです。

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "timestream:Select",
                "timestream:ListMeasures",
                "timestream:WriteRecords",
                "timestream:Unload"
            ],
            "Resource": "arn:aws:timestream:<region>:<account_id>:database/
<database_name>/table/<table_name>"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketAcl",
```

```
                "s3:PutObject",
                "s3:GetObjectMetadata",
                "s3:AbortMultipartUpload"
            ],
            "Resource": [
                "arn:aws:s3:::<S3_Bucket_Created>",
                "arn:aws:s3:::<S3_Bucket_Created>/*"
            ]
        }
    ]
}
```

これらの S3 書き込みアクセス許可に関する追加のコンテキストについては、Amazon Simple Storage Service ガイド を参照してください。エクスポートされたデータの暗号化に KMSキーを使用している場合は、必要な追加のIAMポリシーについては、以下を参照してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
            "kms:DescribeKey",
            "kms:Decrypt",
            "kms:GenerateDataKey*"
        ],
        "Resource": "<account_id>-arn:aws:kms:<region>:<account_id>:key/*",
        "Condition": {
            "ForAnyValue:StringLike": {
                "kms:ResourceAliases": "alias/<Alias_For_Generated_Key>"
            }
        }
    }, {
        "Effect": "Allow",
        "Action": [
            "kms:CreateGrant"
        ],
        "Resource": "<account_id>-arn:aws:kms:<region>:<account_id>:key/*",
        "Condition": {
            "ForAnyValue:StringEquals": {
                "kms:EncryptionContextKeys": "aws:timestream:<database_name>"
            },
            "Bool": {
```

```
                "kms:GrantIsForAWSResource": true
            },
            "StringLike": {
                "kms:ViaService": "timestream.<region>.amazonaws.com"
            },
            "ForAnyValue:StringLike": {
                "kms:ResourceAliases": "alias/<Alias_For_Generated_Key>"
            }
        }
    }
]
}
```

の Timestream UNLOAD からの のベストプラクティス LiveAnalytics

UNLOAD コマンドに関連するベストプラクティスを次に示します。

- UNLOAD コマンドを使用して S3 バケットにエクスポートできるデータ量は制限されません。ただし、クエリは 60 分でタイムアウトするため、1 つのクエリでエクスポートするデータは 60GB 以下にすることをお勧めします。60GB を超えるデータをエクスポートする必要がある場合は、ジョブを複数のクエリに分割します。
- S3 に数千件のリクエストを送信してデータをアップロードできますが、書き込みオペレーションを複数の S3 プレフィックスに並列化することをお勧めします。https://docs.aws.amazon.com/AmazonS3/latest/userguide/optimizing-performance.html「」のドキュメントを参照してください。複数のリーダー/ライターが同じフォルダにアクセスすると、S3 APIコールレートをスロットリングできます。
- プレフィックスの定義には S3 キー長の制限があるため、特に partitioned_by句を使用する場合は、バケット名とフォルダ名を 10～15 文字以内にすることをお勧めします。
- UNLOAD ステートメントを含むクエリに対して 4XX または 5XX を受信すると、部分的な結果が S3 バケットに書き込まれる可能性があります。の Timestream LiveAnalytics は、バケットからデータを削除しません。同じ S3 送信先で別のUNLOADクエリを実行する前に、失敗したクエリによって作成されたファイルを手動で削除することをお勧めします。失敗したクエリによって書き込まれたファイルは、対応する で識別できますQueryExecutionId。失敗したクエリの場合、 の Timestream LiveAnalytics はマニフェストファイルを S3 バケットにエクスポートしません。
- の Timestream LiveAnalytics は、マルチパートアップロードを使用してクエリ結果を S3 にエクスポートします。UNLOAD ステートメントを含むクエリ LiveAnalytics に対して Timestream から 4XX または 5XX を受け取ると、 の Timestream LiveAnalytics はマルチパートアップロードのベストエフォート中止を実行しますが、一部の不完全なパートが残されている可能性があります。

したがって、S3 バケットで未完了のマルチパートアップロードの自動クリーンアップを設定するには、https://aws.amazon.com/blogs/aws-cloud-financial-management/discovering-and-deleting-incomplete-multipart-uploads-to-lower-amazon-s3-costs/「」のガイドラインに従うことをお勧めします。

CSV パーサーを使用して CSV 形式でデータにアクセスするための推奨事項

- CSV パーサーでは、区切り文字、エスケープ、引用符の文字を同じ文字にすることはできません。
- 一部のCSVパーサーは、配列などの複雑なデータ型を解釈できないため、JSONデシリアライザーを使用して解釈することをお勧めします。

Parquet 形式でデータにアクセスするための推奨事項

1. ユースケースでスキーマ別名列名で UTF-8 文字のサポートが必要な場合は、Parquet-mr ライブラリ を使用することをお勧めします。
2. 結果のタイムスタンプは 12 バイトの整数 (INT96) で表されます。
3. 時系列は として表されarray<row<time, value>>、他のネストされた構造は Parquet 形式でサポートされている対応するデータ型を使用します。

partition_by 句の使用

- partitioned_by フィールドで使用される列は、選択クエリの最後の列である必要があります。partitioned_by フィールドで複数の列が使用されている場合、列は選択クエリの最後の列で、partition_byフィールドで使用されるのと同じ順序である必要があります。
- データのpartitioned_byパーティション化 (フィールド) に使用される列値には、 ASCII 文字のみを含めることができます。の Timestream LiveAnalytics では値に UTF-8 文字を使用できますが、S3 ではオブジェクトキーとしてASCII文字のみをサポートしています。

の Timestream UNLOAD からの のユースケースの例 LiveAnalytics

E コマースウェブサイトのユーザーセッションメトリクス、トラフィックソース、製品購入をモニタリングしているとします。の Timestream を使用して、ユーザーの動作、製品販売に関するリアルタイムのインサイトを取得し、顧客 LiveAnalytics をウェブサイトに誘導するトラフィックチャネル (組

織検索、ソーシャルメディア、ダイレクトトラフィック、有料キャンペーンなど) でマーケティング分析を実行します。

トピック

パーティションなしでデータをエクスポートする

過去 2 日間のデータを CSV 形式でエクスポートします。

```
UNLOAD(SELECT user_id, ip_address, event, session_id, measure_name, time,
query, quantity, product_id, channel
FROM sample_clickstream.sample_shopping
WHERE time BETWEEN ago(2d) AND now())
TO 's3://<bucket_name>/withoutpartition'
WITH (  format='CSV',
compression='GZIP')
```

チャンネルによるデータのパーティション分割

過去 2 日間のデータを CSV 形式でエクスポートしたいが、各トラフィックチャネルのデータを別のフォルダに保持したい。これを行うには、以下に示すように、 channel 列を使用してデータをパーティション化する必要があります。

```
UNLOAD(SELECT user_id, ip_address, event, session_id, measure_name, time,
query, quantity, product_id, channel
FROM sample_clickstream.sample_shopping
WHERE time BETWEEN ago(2d) AND now())
TO 's3://<bucket_name>/partitionbychannel/'
WITH (
partitioned_by = ARRAY ['channel'],
format='CSV',
compression='GZIP')
```

イベントによるデータのパーティショニング

過去 2 日間のデータを CSV 形式でエクスポートしたいが、各イベントのデータを別のフォルダに保持したい。これを行うには、以下に示すように、 event列を使用してデータをパーティション化する必要があります。

```
UNLOAD(SELECT user_id, ip_address, channel, session_id, measure_name, time,
query, quantity, product_id, event
FROM sample_clickstream.sample_shopping
WHERE time BETWEEN ago(2d) AND now())
TO 's3://<bucket_name>/partitionbyevent/'
WITH (
partitioned_by = ARRAY ['event'],
format='CSV',
compression='GZIP')
```

チャネルとイベントの両方でデータをパーティション化

過去 2 日間のデータを CSV 形式でエクスポートしたいが、各チャンネルのデータとチャンネル内のデータを各イベントを別のフォルダに格納したい。これを行うには、以下に示すように、channelと event 列の両方を使用してデータをパーティション化する必要があります。

```
UNLOAD(SELECT user_id, ip_address, session_id, measure_name, time,
query, quantity, product_id, channel,event
FROM sample_clickstream.sample_shopping
WHERE time BETWEEN ago(2d) AND now())
TO 's3://<bucket_name>/partitionbychannelevent/'
WITH (
partitioned_by = ARRAY ['channel','event'],
format='CSV',
compression='GZIP')
```

マニフェストファイルとメタデータファイル

マニフェストファイル

マニフェストファイルは、UNLOAD実行時にエクスポートされるファイルのリストに関する情報を提供します。マニフェストファイルは、 というファイル名の指定された S3 バケットで使用できますS3://bucket_name/<queryid>_<UUID>_manifest.json。マニフェストファイルには、結果フォルダ内のファイルの URL、各ファイルのレコード数とサイズ、およびクエリメタデータ (クエリのために S3 にエクスポートされた合計バイト数と合計行数) が含まれます。

```
{
  "result_files": [
    {
        "url":"s3://my_timestream_unloads/ec2_metrics/
AEDAGANLHLBH4OLISD3CVOZZRWPX5GV2XCXRBKCVD554N6GWPWWXBP7LSG74V2Q_1448466917_szCL4YgVYzGXj2lS.gz"
        "file_metadata":
            {
                "content_length_in_bytes": 32295,
                "row_count": 10
            }
    },
    {
        "url":"s3://my_timestream_unloads/ec2_metrics/
AEDAGANLHLBH4OLISD3CVOZZRWPX5GV2XCXRBKCVD554N6GWPWWXBP7LSG74V2Q_1448466917_szCL4YgVYzGXj2lS.gz"
        "file_metadata":
            {
                "content_length_in_bytes": 62295,
                "row_count": 20
            }
    },
  ],
  "query_metadata":
    {
      "content_length_in_bytes": 94590,
      "total_row_count": 30,
      "result_format": "CSV",
      "result_version": "Amazon Timestream version 1.0.0"
    },
  "author": {
        "name": "Amazon Timestream",
        "manifest_file_version": "1.0"
  }
}
```

メタデータ

メタデータファイルには、列名、列タイプ、スキーマなどのデータセットに関する追加情報が表示されます。メタデータファイルは、提供された S3 バケットでファイル名 S3://bucket_name/<queryid>_<UUID>_metadata.json で使用できます。

メタデータファイルの例を次に示します。

```
{
```

```
    "ColumnInfo": [
        {
            "Name": "hostname",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "region",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "measure_name",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "cpu_utilization",
            "Type": {
                "TimeSeriesMeasureValueColumnInfo": {
                    "Type": {
                        "ScalarType": "DOUBLE"
                    }
                }
            }
        }
  ],
  "Author": {
        "Name": "Amazon Timestream",
        "MetadataFileVersion": "1.0"
  }
}
```

メタデータファイルで共有される列情報は、SELECTクエリのクエリAPIレスポンスでColumnInfo送信されるものと同じ構造です。

Glue クローラーを使用して Glue Data Catalog を構築する

1. 次の検証のために、管理者認証情報を使用して アカウントにログインします。

2. ここに記載されているガイドラインを使用して、 Glue データベース用のクローラーを作成します。データソースに提供される S3 フォルダは、 などのUNLOAD結果フォルダである必要があることに注意してくださいs3://my_timestream_unloads/results。
3. https://docs.aws.amazon.com/glue/latest/ug/tutorial-add-crawler.html#tutorial-add-crawler-step2「」のガイドラインに従ってクローラーを実行します。
4. Glue テーブルを表示します。
   - AWS Glue → Tables に移動します。
   - クローラーの作成時にテーブルプレフィックスを指定して作成された新しいテーブルが表示されます。
   - テーブルの詳細ビューをクリックすると、スキーマとパーティションの情報を表示できます。

以下は、Glue Data Catalog を使用する AWS 他の AWS サービスやオープンソースプロジェクトです。

- Amazon Athena – 詳細については、Amazon Athena ユーザーガイドの「テーブル、データベース、データカタログについて」を参照してください。
- Amazon Redshift Spectrum – 詳細については、「Amazon Redshift データベースデベロッパーガイド」の「Amazon Redshift Spectrum を使用した外部データのクエリ」を参照してください。
- Amazon EMR – 詳細については、「Amazon マネジメントガイド」の AWS 「Glue Data Catalog への Amazon EMR アクセスにリソースベースのポリシーを使用する」を参照してください。EMR
- AWS Apache Hive メタストア用の Glue Data Catalog クライアント – この GitHubプロジェクトの詳細については、AWS 「Apache Hive メタストア用の Glue Data Catalog クライアント」を参照してください。

の Timestream UNLOADからの の制限 LiveAnalytics

UNLOAD コマンドに関連する制限を次に示します。

- UNLOAD ステートメントを使用するクエリの同時実行数は、1 秒あたり 1 クエリです (QPS) 。クエリレートを超えると、スロットリングが発生する可能性があります。
- UNLOAD ステートメントを含むクエリは、クエリごとに最大 100 個のパーティションをエクスポートできます。エクスポートされたデータをパーティション化するために、選択した列の個別の数を確認することをお勧めします。
- UNLOAD ステートメントを含むクエリは 60 分後にタイムアウトします。

- Amazon S3 でUNLOADステートメントが作成するファイルの最大サイズは 78 GB です。 Amazon S3

の Timestream のその他の制限については LiveAnalytics、「」を参照してください。 クォータ

クエリインサイトを使用して Amazon Timestream のクエリを最適化する

クエリインサイトは、クエリの最適化、パフォーマンスの向上、コスト削減に役立つパフォーマンス調整機能です。クエリインサイトを使用すると、クエリの時間的、時間的、空間的なパーティションキーベースのプルーニング効率を評価できます。クエリインサイトを使用して、クエリのパフォーマンスを向上させる改善すべき分野を特定することもできます。さらに、クエリインサイトを使用すると、クエリが時間ベースおよびパーティションキーベースのインデックス作成を使用してデータ取得を最適化する効果を評価できます。クエリのパフォーマンスを最適化するには、クエリの実行を管理する時間パラメータと空間パラメータの両方を微調整することが重要です。

トピック

- クエリインサイトの利点
- Amazon Timestream のデータアクセスの最適化
- Amazon Timestream でのクエリインサイトの有効化
- クエリインサイトレスポンスを使用したクエリの最適化

クエリインサイトの利点

クエリインサイトを使用する主な利点は次のとおりです。

- 非効率的なクエリの特定 – クエリインサイトは、クエリによってアクセスされるテーブルの時間ベースおよび属性ベースのプルーニングに関する情報を提供します。この情報は、最適ではないアクセス先のテーブルを特定するのに役立ちます。
- データモデルとパーティショニングの最適化 – クエリインサイト情報を使用して、データモデルとパーティショニング戦略にアクセスして微調整できます。
- クエリの調整 – クエリインサイトは、インデックスをより効果的に使用する機会を強調します。

Amazon Timestream のデータアクセスの最適化

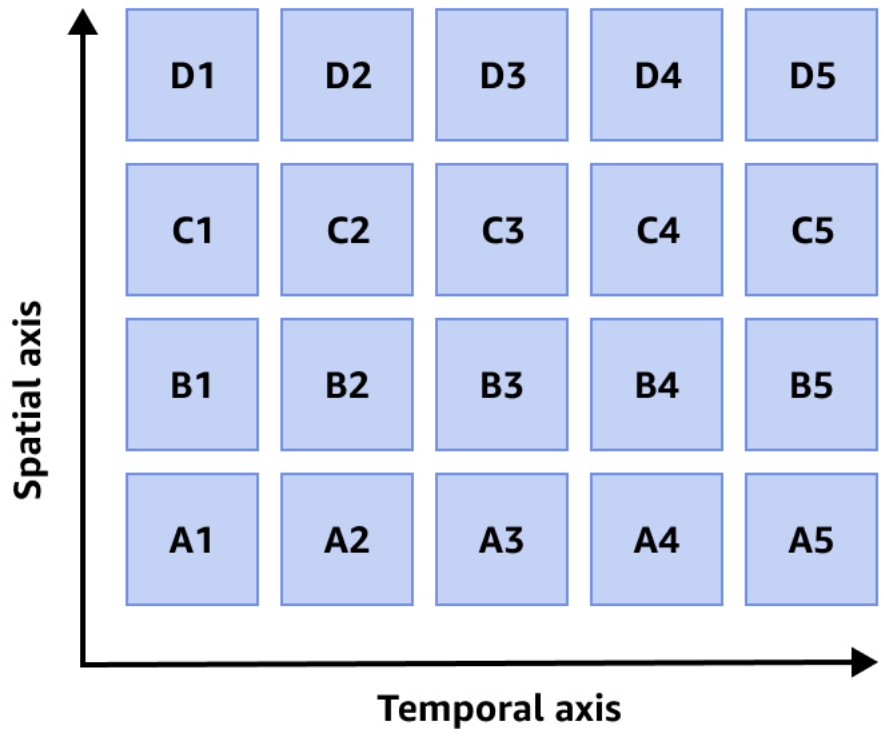
Timestream パーティショニングスキームまたはデータ組織手法を使用して、Amazon Timestream のデータアクセスパターンを最適化できます。

トピック

- タイムストリームパーティショニングスキーム
- データ組織

タイムストリームパーティショニングスキーム

Amazon Timestream は、各 Timestream テーブルに数百、数千、または数百万の独立したパーティションを持つことができる、非常にスケーラブルなパーティショニングスキームを使用します。可用性の高いパーティション追跡およびインデックス作成サービスは、パーティション化を管理し、障害の影響を最小限に抑え、システムの耐障害性を高めます。



データ組織

Timestream は、取り込んだ各データポイントを 1 つのパーティションに保存します。Timestream テーブルにデータを取り込むと、Timestream はデータ内のタイムスタンプ、パーティションキー、およびその他のコンテキスト属性に基づいてパーティションを自動的に作成します。Timestream は、データを時間どおりにパーティショニングする (一時パーティショニング) だけでなく、選択したパーティショニングキーやその他のディメンション (空間パーティショニング) に基づいてデータをパーティショニングします。このアプローチは、書き込みトラフィックを分散し、クエリのデータを効果的にプルーニングできるように設計されています。

クエリインサイト機能は、クエリのプルーニング効率に関する貴重なインサイトを提供します。これには、クエリ空間カバレッジとクエリ時間カバレッジが含まれます。

トピック

- QuerySpatialCoverage
- QueryTemporalCoverage

QuerySpatialCoverage

QuerySpatialCoverage メトリクスは、実行されたクエリの空間カバレッジと、最も非効率な空間プルーニングを持つテーブルに関するインサイトを提供します。この情報は、空間剪定を強化するためのパーティショニング戦略の改善分野を特定するのに役立ちます。QuerySpatialCoverage メトリクスの値は 0 ~ 1 の範囲です。メトリクスの値が低いほど、空間軸でのクエリプルーニングがより最適になります。例えば、0.1 の値は、クエリが空間軸の 10% をスキャンしていることを示します。1 の値は、クエリが空間軸の 100% をスキャンしていることを示します。

Example クエリインサイトを使用してクエリの空間カバレッジを分析する

気象データを保存する Timestream データベースがあるとします。米国のさまざまな州にある気象ステーションから、温度が 1 時間ごとに記録されると仮定します。カスタマー定義のパーティショニングキー (CDPK) Stateとして を選択して、データを状態別にパーティション化するとします。

クエリを実行して、特定の日の午後 2 時から午後 4 時の間、カリフォルニアのすべての気象ステーションの平均温度を取得するとします。次の例は、このシナリオのクエリを示しています。

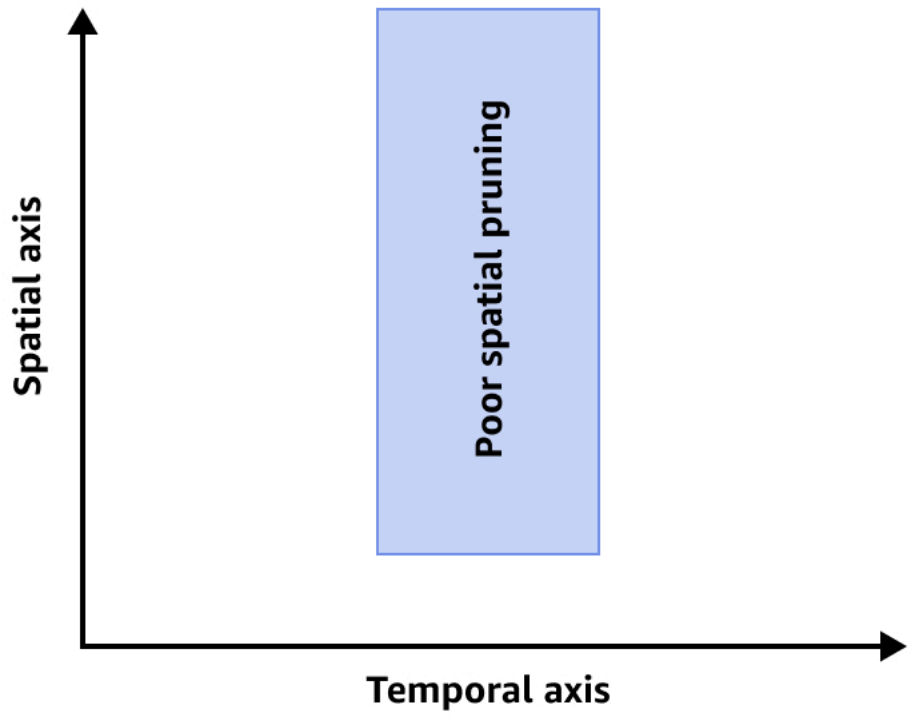
```
SELECT AVG(temperature)
FROM "weather_data"."hourly_weather"
WHERE time >= '2024-10-01 14:00:00' AND time < '2024-10-01 16:00:00'
```

```
  AND state = 'CA';
```

クエリインサイト機能を使用すると、クエリの空間カバレッジを分析できます。QuerySpatialCoverage メトリクスが 0.02 の値を返すとします。これは、クエリが空間軸の 2% のみをスキャンしたことを意味します。これは効率的です。この場合、クエリは空間範囲を効果的にプルーニングすることができ、カリフォルニアからデータを取得し、他の状態からデータを無視するだけです。

逆に、QuerySpatialCoverageメトリクスが 0.8 の値を返した場合、クエリが空間軸の 80% をスキャンしたことを示します。これは効率が悪くなります。これは、空間剪定を改善するためにパーティショニング戦略を改良する必要があることを示唆している可能性があります。例えば、パーティションキーは、状態ではなく都市またはリージョンとして選択できます。QuerySpatialCoverage メトリクスを分析することで、パーティショニング戦略を最適化し、クエリのパフォーマンスを向上させる機会を特定できます。

次の図は、空間プルーニングが不十分なことを示しています。



空間剪定効率を向上させるには、次のいずれかまたは両方を実行できます。

- measure_name、デフォルトのパーシショニングキーを追加するか、クエリでCDPK述語を使用します。
- 前のポイントで説明した属性を既に追加している場合は、 などのこれらの属性または句の周りの関数を削除しますLIKE。

QueryTemporalCoverage

QueryTemporalCoverage メトリクスは、スキャンされた時間範囲が最も大きいテーブルなど、実行されたクエリによってスキャンされた時間範囲に関するインサイトを提供します。QueryTemporalCoverage メトリクスの値は、ナノ秒で表される時間範囲です。このメトリクスの値が低いほど、時間範囲のクエリプルーニングがより最適になります。例えば、過去数分間のデータをスキャンするクエリは、テーブルの時間範囲全体をスキャンするクエリよりもパフォーマンスが高くなります。

Example

IoT IoT センサーデータを保存する Timestream データベースがあり、製造工場にあるデバイスから 1 分ごとに測定されているとします。でデータをパーティション化したとしますdevice_ID。

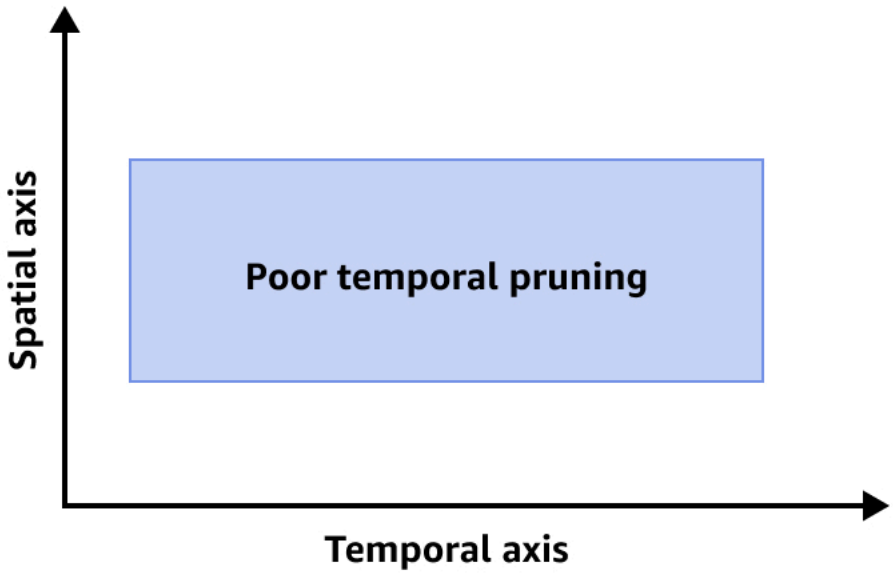
過去 30 分間にクエリを実行して、特定のデバイスの平均センサー読み取り値を取得するとします。次の例は、このシナリオのクエリを示しています。

```
SELECT AVG(sensor_reading)
FROM "sensor_data"."factory_1"
WHERE device_id = 'DEV_123'
  AND time >= NOW() - INTERVAL 30 MINUTE and time < NOW();
```

クエリインサイト機能を使用すると、クエリによってスキャンされた時間範囲を分析できます。QueryTemporalCoverage メトリクスが 1800000000000 ナノ秒 (30 分) の値を返すとします。これは、クエリが過去 30 分間のデータのみをスキャンしたことを意味します。これは比較的狭い時間範囲です。これは、クエリが時間パーティショニングを効果的にプルーニングし、要求されたデータのみを取得できたことを示す良い兆候です。

逆に、QueryTemporalCoverageメトリクスが 1 年の値をナノ秒単位で返した場合、クエリがテーブル内の 1 年間の時間範囲をスキャンしたことを示します。これは効率が悪くなります。これは、クエリが一時的なプルーニング用に最適化されていないことを示唆する場合があり、時間フィルターを追加することでクエリを改善できます。

次の図は、時間的なプルーニングが不十分なことを示しています。



時間剪定を改善するには、次のいずれかまたはすべてを実行することをお勧めします。

- クエリに欠落している時間述語を追加し、その時間述語が目的の時間枠をプルーニングしていることを確認します。
- 述語の前後に MAX( )などの関数を削除します。
- すべてのサブクエリにタイム述語を追加します。これは、サブクエリが大きなテーブルを結合したり、複雑なオペレーションを実行したりする場合に重要です。

Amazon Timestream でのクエリインサイトの有効化

クエリレスポンスを通じて直接配信されるインサイトを使用して、クエリのクエリインサイトを有効にできます。クエリインサイトを有効にすると、追加のインフラストラクチャや追加コストは必要ありません。クエリインサイトを有効にすると、クエリレスポンスの一部としてクエリ結果に加えて、クエリパフォーマンス関連のメタデータフィールドが返されます。この情報を使用してクエリを調整し、クエリのパフォーマンスを向上させ、クエリコストを削減できます。

クエリインサイトの有効化については、「」を参照してくださいクエリを実行する。

クエリインサイトを有効にして返されるレスポンスの例を表示するには、「スケジュールされたクエリの例」を参照してください。

> **Note**
>
> - クエリインサイトを有効にすると、レートはクエリを 1 秒あたり 1 クエリ () に制限しますQPS。パフォーマンスへの影響を避けるため、クエリインサイトを本番環境にデプロイする前に、クエリの評価段階でのみ有効にすることを強くお勧めします。
> - クエリインサイトで提供されるインサイトは、最終的に一貫性があります。つまり、新しいデータがテーブルに継続的に取り込まれると、それらが変化する可能性があります。

クエリインサイトレスポンスを使用したクエリの最適化

Amazon Timestream を使用して、さまざまな場所のエネルギー消費 LiveAnalytics をモニタリングしているとします。データベースに `raw-metrics`と という名前のテーブルが 2 つあるとしますaggregate-metrics。

この`raw-metrics`テーブルには、詳細なエネルギーデータがデバイスレベルで保存され、次の列が含まれます。

- タイムスタンプ
- 州、例えばワシントン
- デバイス ID
- エネルギー消費量

このテーブルのデータは収集され、 minute-by-minuteきめ細かに保存されます。テーブルは を `State`として使用しますCDPK。

この`aggregate-metrics`テーブルには、スケジュールされたクエリの結果が保存され、すべてのデバイス間でエネルギー消費データを 1 時間ごとに集計します。このテーブルには、次の列が含まれます。

- タイムスタンプ
- 州、例えばワシントン
- 総エネルギー消費量

この`aggregate-metrics`テーブルには、このデータが時間単位で保存されます。テーブルは を `State`として使用しますCDPK。

トピック

過去 24 時間のエネルギー消費量のクエリ

過去 24 時間にワシントンで消費された合計エネルギーを抽出するとします。このデータを見つけるには、 raw-metricsと の両方のテーブルの長所を活用できますaggregate-metrics。aggregate-metrics テーブルには過去 23 時間の時間単位のエネルギー消費データが表示され、raw-metricsテーブルには過去 1 時間の分粒度データが表示されます。両方のテーブルをクエリすることで、過去 24 時間のワシントンでのエネルギー消費の完全かつ正確な全体像を把握できます。

```
SELECT am.time, am.state, am.total_energy_consumption,
rm.time, rm.state, rm.device_id, rm.energy_consumption
FROM
 "metrics"."aggregate-metrics" am
 LEFT JOIN "metrics"."raw-metrics" rm ON am.state = rm.state
WHERE rm.time >= ago(1h) and rm.time < now()
```

このクエリ例は説明のみを目的としており、そのままは機能しない場合があります。これは概念を実証することを目的としていますが、特定のユースケースや環境に合わせて変更する必要がある場合があります。

このクエリを実行した後、クエリの応答時間が予想よりも遅いことに気付くかもしれません。このパフォーマンス問題の根本原因を特定するには、クエリインサイト機能を使用してクエリのパフォーマンスを分析し、実行を最適化できます。

次の例は、クエリインサイトレスポンスを示しています。

```
queryInsightsResponse={
                QuerySpatialCoverage: {
                    Max: {
                        Value: 1.0,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/raw-metrics,
                        PartitionKey: [State]
```

```
                    }
                },
                QueryTemporalRange: {
                    Max: {
                        Value:31540000000000000 //365 days,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/aggregate-metrics
                    }
                },
                QueryTableCount: 2,
                OutputRows: 83,
                OutputBytes: 590
```

クエリインサイトレスポンスには、次の情報が含まれます。

- 時間範囲 : クエリはaggregate-metricsテーブルの過剰な 365 日間の時間範囲をスキャンしました。これは、時間フィルタリングの非効率的な使用を示します。
- 空間カバレッジ : クエリはraw-metricsテーブルの空間範囲全体 (100%) をスキャンしました。これは、空間フィルタリングが効果的に活用されていないことを示唆しています。

クエリが複数のテーブルにアクセスする場合、クエリインサイトはテーブルのメトリクスを最も最適ではないアクセスパターンで提供します。

時間範囲のクエリの最適化

クエリインサイトレスポンスに基づいて、次の例に示すように、時間範囲のクエリを最適化できます。

```
SELECT am.time, am.state, am.total_energy_consumption,
rm.time, rm.state, rm.device_id, rm.energy_consumption
FROM
  "metrics"."aggregate-metrics" am
  LEFT JOIN "metrics"."raw-metrics" rm ON am.state = rm.state
WHERE
  am.time >=  ago(23h) and am.time < now()
  AND rm.time >=  ago(1h) and rm.time < now()
  AND rm.state = 'Washington'
```

QueryInsights コマンドを再度実行すると、次のレスポンスが返されます。

```
queryInsightsResponse={
```

```
                QuerySpatialCoverage: {
                    Max: {
                        Value: 1.0,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/aggregate-metrics,
                        PartitionKey: [State]
                    }
                },
                QueryTemporalRange: {
                    Max: {
                        Value: 82800000000000 //23 hours,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/aggregate-metrics
                    }
                },
                QueryTableCount: 2,
                OutputRows: 83,
                OutputBytes: 590
```

このレスポンスは、aggregate-metricsテーブルの空間カバレッジがまだ 100% であり、非効率であることを示しています。次のセクションでは、空間カバレッジのクエリを最適化する方法を示します。

空間カバレッジのクエリの最適化

クエリインサイトレスポンスに基づいて、次の例に示すように、空間カバレッジのクエリを最適化できます。

```
SELECT am.time, am.state, am.total_energy_consumption,
rm.time, rm.state, rm.device_id, rm.energy_consumption
FROM
  "metrics"."aggregate-metrics" am
  LEFT JOIN "metrics"."raw-metrics" rm ON am.state = rm.state
WHERE
  am.time >=  ago(23h) and am.time < now()
  AND am.state ='Washington'
  AND rm.time >=  ago(1h) and rm.time < now()
  AND rm.state = 'Washington'
```

QueryInsights コマンドを再度実行すると、次のレスポンスが返されます。

```
queryInsightsResponse={
```

```
                QuerySpatialCoverage: {
                    Max: {
                        Value: 0.02,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/aggregate-metrics,
                        PartitionKey: [State]
                    }
                },
                QueryTemporalRange: {
                    Max: {
                        Value: 82800000000000 //23 hours,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/aggregate-metrics
                    }
                },
                QueryTableCount: 2,
                OutputRows: 83,
                OutputBytes: 590
```

クエリパフォーマンスの向上

クエリを最適化すると、クエリインサイトは次の情報を提供します。

- aggregate-metrics テーブルの一時プルーニングは 23 時間です。これは、時間範囲の 23 時間のみがスキャンされることを示します。
- aggregate-metrics テーブルの空間プルーニングは 0.02 です。これは、テーブルの空間範囲データの 2% のみがスキャンされていることを示します。このクエリは、テーブルのごく一部をスキャンし、高速なパフォーマンスとリソース使用率の低下につながります。プルーニング効率が向上したことは、クエリのパフォーマンスが最適化されるようになったことを示しています。

の使用 AWS Backup

Amazon Timestream for のデータ保護機能は LiveAnalytics 、規制コンプライアンスと事業継続性要件を満たすのに役立つフルマネージドソリューションです。この機能は、バックアップの作成 AWS Backup、移行、復元、削除を簡素化すると同時に、レポートと監査を改善できるように設計された統合バックアップサービスである とのネイティブ統合によって有効化されます。との統合により、フルマネージド型のポリシー駆動型の集中型データ保護ソリューション AWS Backupを使用して、不変のバックアップを作成し、Timestream および でサポートされている他のAWSサービスにまたがるアプリケーションデータのデータ保護を一元管理できます AWS Backup。

機能を使用するには、 AWS Backup が Timestream リソースを保護するようにオプトインする必要があります。オプトインの選択は特定のアカウントと AWS リージョンに適用されるため、同じアカウントを使用して複数のリージョンにオプトインする必要がある場合があります。Backup の詳細については、 AWS AWS Backup 「 デベロッパーガイド」を参照してください。

から利用できるデータ保護機能 AWS Backup には、以下が含まれます。

スケジュールされたバックアップ — バックアッププランを使用して LiveAnalytics 、テーブルの Timestream の定期的なスケジュールされたバックアップを設定できます。

クロスアカウントコピーとクロスリージョンコピー — バックアップを別の AWS リージョンまたはアカウントの別のバックアップボールトに自動的にコピーできるため、データ保護要件をサポートできます。

コールドストレージの階層化 — バックアップを設定してライフサイクルルールを実装し、バックアップを削除したり、よりコールドストレージに移行したりできます。これにより、バックアップコストを最適化できます。

タグ — 請求とコスト配分の目的で、バックアップに自動的にタグ付けできます。

暗号化 — バックアップデータはボールトに AWS Backup 保存されます。これにより、Timestream から独立したキーを使用して AWS KMS LiveAnalytics 、テーブル暗号化キーを暗号化して保護できます。

WORM モデルを使用してバックアップを保護する — ボールトロックを使用して AWS Backup バックアップの (WORM) 設定を有効に write-once-read-manyできます。 AWS Backup ボールトロックを使用すると、バックアップを不注意または悪意のある削除オペレーション、バックアップ保持期間の変更、ライフサイクル設定の更新から保護する追加の防御レイヤーを追加できます。詳細については、AWS Backup Vault Lock を参照してください。

データ保護機能はすべてのリージョンで利用できます。機能の詳細については、AWS Backup 「 デベロッパーガイド」を参照してください。

Timestream テーブルのバックアップと復元: 仕組み

Amazon Timestream テーブルのバックアップを作成できます。このセクションでは、バックアップおよび復元プロセス中の概要について示します。

トピック

- バックアップ
- 復元

バックアップ

オンデマンドバックアップ機能を使用して、 LiveAnalytics テーブルの Amazon Timestream の完全なバックアップを作成できます。このセクションでは、バックアップおよび復元プロセス中の概要について示します。

Timestream データのバックアップは、テーブルの粒度で作成できます。選択したテーブルのバックアップは、Timestream コンソール、コンソール、、 AWS Backup SDKまたは を使用して開始できますCLI。バックアップは非同期で作成され、バックアップの開始時刻がバックアップに含まれるまで、テーブル内のすべてのデータが作成されます。ただし、バックアップの進行中にテーブルに取り込まれたデータの一部がバックアップに含まれている可能性があります。データを保護するために、1 回限りのオンデマンドバックアップを作成するか、テーブルの定期的なバックアップをスケジュールできます。

バックアップの実行中は、次の操作を行うことはできません。

- バックアップオペレーションの一時停止またはキャンセル。
- バックアップのソーステーブルの削除。
- テーブルのバックアップ中におけるテーブルのバックアップの無効化。

設定が完了すると、 は自動バックアップスケジュール、保持管理、ライフサイクル管理 AWS Backup を提供し、カスタムスクリプトや手動プロセスが不要になります。詳細については、AWS Backup 「 デベロッパーガイド」を参照してください。

LiveAnalytics バックアップのすべての Timestream は本質的に増分的であり、テーブルの最初のバックアップがフルバックアップであり、同じテーブルの後続のすべてのバックアップが増分バックアップであり、最後のバックアップ以降のデータへの変更のみをコピーすることを意味します。の Timestream のデータはパーティションのコレクションに保存 LiveAnalytics されるため、新しいデータの取り込みまたは最後のバックアップ以降の既存のデータの更新によって変更されたすべてのパーティションは、その後のバックアップ中にコピーされます。

LiveAnalytics コンソールに Timestream を使用している場合、アカウント内のすべてのリソース用に作成されたバックアップが Backups タブに表示されます。さらに、バックアップはテーブルの詳細にも一覧表示されます。

復元

LiveAnalytics コンソールの Timestream、または AWS Backup コンソール、、 SDKからテーブルを復元できます AWS CLI。バックアップからデータ全体を復元するか、選択したデータを復元するようにテーブル保持設定を設定できます。復元を開始するときは、次のテーブル設定を設定できます。

- Database Name
- テーブル名
- メモリストアの保持
- マグネティックストアの保持
- マグネティックストレージの書き込みを有効にする
- S3 エラーログの場所 (オプション)
- IAM バックアップの復元時に AWS Backup 引き受ける ロール

上記の設定はソーステーブルとは無関係です。バックアップ内のすべてのデータを復元するには、メモリストアの保持期間とマグネティックストアの保持期間の合計が、最も古いタイムスタンプと現在のタイムスタンプの差よりも大きくなるように、新しいテーブル設定を設定することをお勧めします。復元する増分バックアップを選択すると、すべてのデータ (増分 + 基礎となるフルデータ) が復元されます。復元が成功すると、テーブルはアクティブ状態になり、復元されたテーブルに対して取り込みおよび/またはクエリ操作を実行できます。ただし、復元の進行中にこれらの操作を実行することはできません。復元されると、テーブルはアカウントの他のテーブルと似ています。

Example バックアップからすべてのデータを復元する

この例では、次の前提があります。

最も古いタイムスタンプ —`August 1, 2021 0:00:00`

- 現在 —`November 9, 2022 0:00:00`

バックアップからすべてのデータを復元するには、次のように値を入力して比較します。

1. Memory store retention と Magnetic store retention を入力します。例えば、これらの値を仮定します。
   - メモリストアの保持 — 12 時間
   - マグネティックストアの保持 — 500 日
2. メモリストアの保持とマグネティックストアの保持の合計を検索します。

```
12 hours + (500 * 24 hours) =
12 hours + 12,000 hours =
12,012 hours
```

3. Oldest timestamp と Now の違いを見つけます。

```
November 9, 2022 0:00:00 - August 1, 2021 0:00:00 =
465 days =
465 * 24 hours =
11,160 hours
```

4. 2 番目のステップの保持値の合計が 3 番目のステップの時間の差より大きいことを確認します。必要に応じて保持時間を調整します。

```
12,012 > 11,160
true
```

Example バックアップから選択データを復元する

この例では、次の前提があります。

- 現在 —November 9, 2022 0:00:00

バックアップから選択したデータのみを復元するには、次のように値を入力して比較します。

1. 必要な最も早いタイムスタンプを決定します。例えば、 を想定しますDecember 4, 2021 0:00:00。
2. 必要な最も早いタイムスタンプと現在との違いを見つけます。

```
November 9, 2022 0:00:00 - December 4, 2021 0:00:00 =
340 days =
340 * 24 hours =
8,160 hours
```

3. メモリストアの保持に必要な値を入力します。例えば、12 時間を入力します。
4. 2 番目のステップの差から値を減算します。

```
8,160 hours - 12 hours =
```

```
8148 hours
```

5. 磁気ストア保持 にその値を入力します。

LiveAnalytics テーブルデータの Timestream のバックアップを別の AWS リージョンにコピーし、その新しいリージョンに復元できます。 AWS 商用リージョンと AWS GovCloud （ 米国) リージョン間でバックアップをコピーして復元できます。送信元リージョンからコピーしたデータと、送信先リージョンの新しいテーブルに復元したデータに対してのみ料金が発生します。

テーブルが復元されたら、復元されたテーブルに以下を手動で設定する必要があります。

- AWS Identity and Access Management (IAM) ポリシー
- タグ
- スケジュールされたクエリ

復元時間は、テーブルの設定に直接関係します。これには、テーブルのサイズ、基盤となるパーティションの数、メモリストアに復元されたデータの量、およびその他の変数が含まれます。ディザスタリカバリを計画する際のベストプラクティスは、平均復元完了時間を定期的に文書化し、これらの時間が全体的な目標復旧時間にどのように影響するかを確認することです (RTO)。

すべてのバックアップおよび復元コンソールとAPIアクションは、ログ記録、継続的なモニタリング、監査のためにキャプチャおよび記録 AWS CloudTrailされます。

Amazon Timestream テーブルのバックアップの作成

このセクションでは、Amazon Timestream のオンデマンド AWS Backup バックアップとスケジュールされたバックアップを有効にして作成する方法について説明します。

トピック

- AWS Backup データの Timestream の保護を有効にする LiveAnalytics
- オンデマンドバックアップの作成
- スケジュールバックアップ

AWS Backup データの Timestream の保護を有効にする LiveAnalytics

AWS Backup で Timestream で使用するには、 を有効にする必要があります LiveAnalytics。

LiveAnalytics コンソールの Timestream AWS Backup で を有効にするには、次の手順を実行します。

1. AWS マネジメントコンソールにサインインします。
2. データの Timestream をサポートできるように、 LiveAnalytics ダッシュボードの Timestream ページの上部 AWS Backup にポップアップバナーが表示されます LiveAnalytics 。それ以外の場合は、ナビゲーションペインからバックアップ を選択します。
3. Backup ウィンドウで、 を有効にするバナーが表示されます AWS Backup。[Enable (有効化)] を選択します。

   LiveAnalytics テーブルの Timestream で AWS Backup によるデータ保護が利用可能になりました。

から を有効にするには AWS Backup、 コンソールを介してプログラムで を有効にする AWS Backup ドキュメントを参照してください。

有効にした後で LiveAnalytics Timestream のデータの保護 AWS Backup を無効にする場合は、 AWS Backup コンソールからログインし、トグルを左に移動します。

AWS Backup 機能を有効または無効にできない場合は、 AWS 管理者がこれらのアクションを実行する必要がある場合があります。

オンデマンドバックアップの作成

LiveAnalytics テーブルの Timestream のオンデマンドバックアップを作成するには、次の手順に従います。

1. AWS マネジメントコンソールにサインインします。
2. コンソールの左側のナビゲーションペインで、[Backups] (バックアップ) を選択します。
3. [オンデマンドバックアップを作成] を選択します。
4. バックアップウィンドウで設定の選択を続行します。
5. 今すぐバックアップを作成するか、すぐにバックアップを開始するか、バックアップウィンドウを選択してバックアップを開始できます。
6. バックアップのライフサイクル管理ポリシーを選択します。バックアップデータをコールドストレージに移行し、バックアップを最低 90 日間保持する必要があります。バックアップに必要な保持期間を設定できます。既存のボールトを選択するか、新しいバックアップボールトを作成す

るを選択して AWS Backup コンソールに移動し、新しいバックアップボールトを作成します。< ここで新しいバックアップボールトを作成する際のドキュメントリンク >

7. 適切なIAMロールを選択します。
8. オンデマンドバックアップに 1 つ以上のタグを割り当てる場合は、[キー] とオプションの [値]] を入力して、[タグを追加] を選択します。
9. オンデマンドバックアップの作成を選択します。これにより、バックアップページが表示され、ジョブのリストが表示されます。
10. バックアップするリソースの [バックアップジョブ ID] を選択すると、そのジョブの詳細が表示されます。

スケジュールバックアップ

バックアップをスケジュールするには、「スケジュールされたバックアップの作成」を参照してください。

Amazon Timestream テーブルのバックアップの復元

このセクションでは、Amazon Timestream テーブルのバックアップを復元する方法について説明します。

トピック

- から LiveAnalytics テーブルの Timestream を復元する AWS Backup
- LiveAnalytics テーブルの Timestream を別のリージョンまたはアカウントに復元する

から LiveAnalytics テーブルの Timestream を復元する AWS Backup

Timestream for LiveAnalytics コンソール AWS Backup の使用から LiveAnalytics テーブルの Timestream を復元するには、次の手順に従います。

1. AWS マネジメントコンソールにサインインします。
2. コンソールの左側のナビゲーションペインで、[Backups] (バックアップ) を選択します。
3. リソースを復元するには、リソースのリカバリポイント ID の横にあるラジオボタンを選択します。ペインの右上隅にある [復元] を選択します。
4. テーブル設定、つまりデータベース名とテーブル名 を入力します。復元されたテーブル名は、元のソーステーブル名とは異なる必要があります。
5. メモリとマグネティックストアの保持設定を構成します。

6. 復元ロール では、この復元に引き受け AWS Backup るIAMロールを選択します。
7. [バックアップを復元] を選択します。ページ上部のメッセージには、復元ジョブに関する情報が表示されます。

> **Note**
>
> 設定されたメモリとマグネティックストアの保持期間に関係なく、バックアップ全体を復元すると料金が発生します。ただし、復元が完了すると、復元されたテーブルには、設定された保持期間内のデータのみが含まれます。

LiveAnalytics テーブルの Timestream を別のリージョンまたはアカウントに復元する

LiveAnalytics テーブルの Timestream を別のリージョンまたはアカウントに復元するには、まずその新しいリージョンまたはアカウントにバックアップをコピーする必要があります。別のアカウントにコピーするには、まずそのアカウントからアクセス許可を付与される必要があります。Timestream を新しいリージョンまたはアカウントに LiveAnalytics バックアップするためにコピーしたら、前のセクションのプロセスで復元できます。

Amazon Timestream テーブルのバックアップのコピー

現在のバックアップのコピーを作成できます。バックアップは、オンデマンドで複数の AWS アカウントまたは AWS リージョンにコピーすることも、スケジュールされたバックアッププランの一部として自動的にコピーすることもできます。クロスリージョンレプリケーションは、ビジネス継続性またはコンプライアンス要件があり、本番稼働用データから最小限の距離だけ離してバックアップを保存する場合に特に役立ちます。

クロスアカウントバックアップは、運用上またはセキュリティ上の理由からバックアップを組織内の 1 つまたは複数の AWS アカウントに確実にコピーする必要がある場合に便利です。元のバックアップを誤って削除してしまった場合は、コピー先のアカウントからコピー元のアカウントにバックアップをコピーし復元できます。これを行うには、Organizations サービスに同じ組織に属し、アカウントに必要なアクセス許可を持つ 2 つのアカウントが必要です。増分バックアップを別のアカウントまたはリージョンにコピーすると、関連付けられたフルバックアップもコピーされます。

特に指定しない限り、コピーはソースバックアップの設定を継承します。1 つ例外があります。新しいコピーを「Never」に指定すると、有効期限が切れます。この設定では、新しいコピーはソースの有効期限を継承します。新しいバックアップコピーを永続的なコピーにする場合は、ソースバック

アップを期限切れにならないように設定するか、新しいコピーの作成から 100 年後を有効期限に指定します。

Timestream コンソールからバックアップをコピーするには、次の手順に従います。

1. AWS マネジメントコンソールにサインインします。
2. コンソールの左側のナビゲーションペインで、[Backups] (バックアップ) を選択します。
3. リソースのリカバリポイント ID の横にあるラジオボタンを選択します。ペインの右上隅で、アクションを選択し、コピー を選択します。
4. AWS バックアップに進む を選択し、クロスアカウントバックアップ のステップに従います。

アカウントとリージョン間でのオンデマンドバックアップとスケジュールされたバックアップのコピーは、現在 LiveAnalytics 、コンソールの Timestream ではネイティブにサポートされていないため、 に移動 AWS Backup してオペレーションを実行する必要があります。

バックアップの削除

このセクションでは、 LiveAnalytics テーブルの Timestream のバックアップを削除する方法について説明します。

Timestream コンソールからバックアップを削除するには、次の手順に従います。

1. AWS マネジメントコンソールにサインインします。
2. コンソールの左側のナビゲーションペインで、[Backups] (バックアップ) を選択します。
3. リソースのリカバリポイント ID の横にあるラジオボタンを選択します。ペインの右上隅で、アクション を選択し、削除 を選択します。
4. Continue to AWS Backup を選択し、「Delete backups」のステップに従ってバックアップを削除しますhttps://docs.aws.amazon.com/aws-backup/latest/devguide/deleting-backups.html。

> **Note**
>
> 増分バックアップを削除すると、増分バックアップのみが削除され、基盤となるフルバックアップは削除されません。

クォータと制限

AWS Backup は、バックアップをリソースごとに 1 つの同時バックアップに制限します。したがって、リソースに対する追加のスケジュールされたバックアップリクエストまたはオンデマンドバックアップリクエストはキューに入れられ、既存のバックアップジョブが完了した後にのみ開始されます。バックアップウィンドウ内でバックアップジョブを開始または完了しない場合、リクエストは失敗します。 AWS Backup 制限の詳細については、AWS 「Backup デベロッパーガイド」の「Backup Limits」を参照してください。 AWS

バックアップを作成するときは、アカウントごとに最大 4 つの同時バックアップを実行できます。同様に、アカウントごとに 1 つの同時復元を実行できます。4 つ以上のバックアップジョブを同時に開始すると、4 つのバックアップジョブのみが開始され、残りのジョブは定期的に再試行されます。起動すると、バックアップジョブが設定されたバックアップウィンドウの期間内に完了しない場合、バックアップジョブは失敗します。失敗したバックアップジョブがオンデマンドバックアップである場合は、バックアップを再試行できます。スケジュールされたバックアップの場合は、次のスケジュールでジョブが試行されます。

ユーザー定義パーティションキー

Amazon Timestream for LiveAnalytics Customer-Defined パーティションキーは、お客様がテーブルに独自のパーティションキーを定義 LiveAnalytics できるようにする の Timestream の機能です。パーティション分割は、複数の物理ストレージユニットにデータを分散するために使用される手法であり、データ取得を迅速かつ効率的に行うことができます。ユーザー定義のパーティションキーを使用すると、クエリパターンとユースケースにより適したパーティショニングスキーマを作成できます。

LiveAnalytics ユーザー定義のパーティションキーの Timestream を使用すると、テーブルのパーティションキーとして 1 つのディメンション名を選択できます。これにより、データのパーティショニングスキーマをより柔軟に定義できます。適切なパーティションキーを選択することで、お客様はデータモデルを最適化し、クエリのパフォーマンスを向上させ、クエリのレイテンシーを短縮できます。

トピック

- ユーザー定義パーティションキーの使用
- ユーザー定義パーティションキーの開始方法
- パーティショニングスキーマ設定の確認
- パーティショニングスキーマ設定の更新

ユーザー定義パーティションキーの使用

カーディナリティのディメンションが高く、クエリレイテンシーが低い、明確に定義されたクエリパターンがある場合、 LiveAnalytics カスタマー定義パーティションキーの Timestream はデータモデルを強化するのに役立つツールになります。例えば、ウェブサイトでカスタマーインタラクションを追跡している小売企業の場合、主なアクセスパターンは、おそらくカスタマー ID とタイムスタンプによるものです。カスタマー ID をパーティションキーとして定義することで、データを均等に分散できるため、レイテンシーが短縮され、最終的にユーザーエクスペリエンスが向上します。

もう 1 つの例は、ウェアラブルデバイスがセンサーデータを収集して患者のバイタルサインを追跡する医療業界です。主なアクセスパターンはデバイス ID とタイムスタンプで、両方のディメンションでカーディナリティが高くなります。デバイス ID をパーティションキーとして定義することで、はクエリの実行を最適化し、長期的なクエリパフォーマンスを維持できます。

要約すると、 LiveAnalytics ユーザー定義パーティションキーの Timestream は、明確なクエリパターン、高いカーディナリティディメンション、およびクエリの低レイテンシーが必要な場合に最も役立ちます。クエリパターンに沿ったパーティションキーを定義することで、クエリの実行を最適化し、長期的なパフォーマンスクエリのパフォーマンスを維持できます。

ユーザー定義パーティションキーの開始方法

コンソールからテーブルを選択し、新しいテーブルを作成します。を使用して `CreateTable`アクションSDKにアクセスして、ユーザー定義のパーティションキーを含めることができる新しいテーブルを作成することもできます。

ディメンションタイプのパーティションキーを使用してテーブルを作成する

次のコードスニペットを使用して、ディメンションタイプのパーティションキーを持つテーブルを作成できます。

Java

```
public void createTableWithDimensionTypePartitionKeyExample() {
        System.out.println("Creating table");
        CreateTableRequest createTableRequest = new CreateTableRequest();
        createTableRequest.setDatabaseName(DATABASE_NAME);
        createTableRequest.setTableName(TABLE_NAME);
        final RetentionProperties retentionProperties = new RetentionProperties()
                .withMemoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                .withMagneticStoreRetentionPeriodInDays(CT_TTL_DAYS);
        createTableRequest.setRetentionProperties(retentionProperties);

        // Can specify enforcement level with OPTIONAL or REQUIRED
        final List<PartitionKey> partitionKeyWithDimensionAndOptionalEnforcement =
 Collections.singletonList(new PartitionKey()
            .withName(COMPOSITE_PARTITION_KEY_DIM_NAME)
            .withType(PartitionKeyType.DIMENSION)
            .withEnforcementInRecord(PartitionKeyEnforcementLevel.OPTIONAL));
        Schema schema = new Schema();

 schema.setCompositePartitionKey(partitionKeyWithDimensionAndOptionalEnforcement);
        createTableRequest.setSchema(schema);

        try {
            writeClient.createTable(createTableRequest);
            System.out.println("Table [" + TABLE_NAME + "] successfully created.");
        } catch (ConflictException e) {
            System.out.println("Table [" + TABLE_NAME + "] exists on database [" +
 DATABASE_NAME + "] . Skipping database creation");
        }
    }
```

Java v2

```
public void createTableWithDimensionTypePartitionKeyExample() {
        System.out.println("Creating table");
        final RetentionProperties retentionProperties =
 RetentionProperties.builder()
                .memoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                .magneticStoreRetentionPeriodInDays(CT_TTL_DAYS)
                .build();
        // Can specify enforcement level with OPTIONAL or REQUIRED
```

```
        final List<PartitionKey> partitionKeyWithDimensionAndOptionalEnforcement =
 Collections.singletonList(PartitionKey
                .builder()
                .name(COMPOSITE_PARTITION_KEY_DIM_NAME)
                .type(PartitionKeyType.DIMENSION)
                .enforcementInRecord(PartitionKeyEnforcementLevel.OPTIONAL)
                .build());
        final Schema schema = Schema.builder()

 .compositePartitionKey(partitionKeyWithDimensionAndOptionalEnforcement).build();
        final CreateTableRequest createTableRequest = CreateTableRequest.builder()
                .databaseName(DATABASE_NAME)
                .tableName(TABLE_NAME)
                .retentionProperties(retentionProperties)
                .schema(schema)
                .build();

        try {
            writeClient.createTable(createTableRequest);
            System.out.println("Table [" + TABLE_NAME + "] successfully created.");
        } catch (ConflictException e) {
            System.out.println("Table [" + TABLE_NAME + "] exists on database [" +
 DATABASE_NAME + "] . Skipping database creation");
        }
    }
```

Go v1

```
func createTableWithDimensionTypePartitionKeyExample(){
        // Can specify enforcement level with OPTIONAL or REQUIRED
        partitionKeyWithDimensionAndOptionalEnforcement :=
 []*timestreamwrite.PartitionKey{
                {
                    Name:                aws.String(CompositePartitionKeyDimName),
                    EnforcementInRecord: aws.String("OPTIONAL"),
                    Type:                aws.String("DIMENSION"),
                },
        }
        createTableInput := &timestreamwrite.CreateTableInput{
             DatabaseName: aws.String(*databaseName),
             TableName:    aws.String(*tableName),
             // Enable MagneticStoreWrite for Table
```

```
            MagneticStoreWriteProperties:
&timestreamwrite.MagneticStoreWriteProperties{
                EnableMagneticStoreWrites: aws.Bool(true),
                // Persist MagneticStoreWrite rejected records in S3
                MagneticStoreRejectedDataLocation:
&timestreamwrite.MagneticStoreRejectedDataLocation{
                    S3Configuration: &timestreamwrite.S3Configuration{
                        BucketName:       aws.String("timestream-sample-bucket"),
                        ObjectKeyPrefix:  aws.String("TimeStreamCustomerSampleGo"),
                        EncryptionOption: aws.String("SSE_S3"),
                    },
                },
            },
            Schema: &timestreamwrite.Schema{
                CompositePartitionKey:
partitionKeyWithDimensionAndOptionalEnforcement,
            }
        }
        _, err := writeSvc.CreateTable(createTableInput)
   }
```

Go v2

```
 func (timestreamBuilder TimestreamBuilder)
CreateTableWithDimensionTypePartitionKeyExample() error {
        partitionKeyWithDimensionAndOptionalEnforcement := []types.PartitionKey{
            {
                Name:                aws.String(CompositePartitionKeyDimName),
                EnforcementInRecord: types.PartitionKeyEnforcementLevelOptional,
                Type:                types.PartitionKeyTypeDimension,
            },
        }
        _, err := timestreamBuilder.WriteSvc.CreateTable(context.TODO(),
&timestreamwrite.CreateTableInput{
            DatabaseName: aws.String(databaseName),
            TableName:    aws.String(tableName),
            MagneticStoreWriteProperties: &types.MagneticStoreWriteProperties{
                EnableMagneticStoreWrites: aws.Bool(true),
                // Persist MagneticStoreWrite rejected records in S3
                MagneticStoreRejectedDataLocation:
&types.MagneticStoreRejectedDataLocation{
                    S3Configuration: &types.S3Configuration{
                        BucketName:       aws.String(s3BucketName),
```

```
                        EncryptionOption: "SSE_S3",
                    },
                },
            },
            Schema: &types.Schema{
                CompositePartitionKey:
 partitionKeyWithDimensionAndOptionalEnforcement,
            },
        })

        if err != nil {
            fmt.Println("Error:")
            fmt.Println(err)
        } else {
            fmt.Println("Create table is successful")
        }
        return err
    }
```

Python

```
def create_table_with_measure_name_type_partition_key(self):
        print("Creating table")
        retention_properties = {
            'MemoryStoreRetentionPeriodInHours': HT_TTL_HOURS,
            'MagneticStoreRetentionPeriodInDays': CT_TTL_DAYS
        }
        partitionKey_with_measure_name = [
            {'Type': 'MEASURE'}
        ]
        schema = {
            'CompositePartitionKey': partitionKey_with_measure_name
        }
        try:
            self.client.create_table(DatabaseName=DATABASE_NAME,
 TableName=TABLE_NAME,
                                     RetentionProperties=retention_properties,
 Schema=schema)
            print("Table [%s] successfully created." % TABLE_NAME)
        except self.client.exceptions.ConflictException:
            print("Table [%s] exists on database [%s]. Skipping table creation" % (
                TABLE_NAME, DATABASE_NAME))
        except Exception as err:
```

```
            print("Create table failed:", err)
```

パーティショニングスキーマ設定の確認

スキーマをパーティショニングするためのテーブル設定は、いくつかの方法で確認できます。コンソールからデータベースを選択し、確認するテーブルを選択します。を使用して DescribeTableアクションSDKにアクセスすることもできます。

パーティションキーを使用してテーブルを記述する

次のコードスニペットを使用して、パーティションキーを持つテーブルを記述できます。

Java

```
   public void describeTable() {
       System.out.println("Describing table");
       final DescribeTableRequest describeTableRequest = new
DescribeTableRequest();
       describeTableRequest.setDatabaseName(DATABASE_NAME);
       describeTableRequest.setTableName(TABLE_NAME);
       try {
           DescribeTableResult result =
amazonTimestreamWrite.describeTable(describeTableRequest);
           String tableId = result.getTable().getArn();
           System.out.println("Table " + TABLE_NAME + " has id " + tableId);
           // If table is created with composite partition key, it can be described
with
           //
System.out.println(result.getTable().getSchema().getCompositePartitionKey());
       } catch (final Exception e) {
           System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
           throw e;
       }
   }
```

以下に出力例を示します。

1. テーブルにディメンションタイプのパーティションキーがある

```
[{Type: DIMENSION,Name: hostId,EnforcementInRecord: OPTIONAL}]
```

2. テーブルにメジャー名タイプのパーティションキーがある

```
[{Type: MEASURE,}]
```

3. 複合パーティションキーを指定せずに作成されたテーブルから複合パーティションキーを取得する

```
[{Type: MEASURE,}]
```

Java v2

```
 public void describeTable() {
       System.out.println("Describing table");
       final DescribeTableRequest describeTableRequest =
DescribeTableRequest.builder()
               .databaseName(DATABASE_NAME).tableName(TABLE_NAME).build();
       try {
           DescribeTableResponse response =
writeClient.describeTable(describeTableRequest);
           String tableId = response.table().arn();
           System.out.println("Table " + TABLE_NAME + " has id " + tableId);
           // If table is created with composite partition key, it can be described
with
           //
System.out.println(response.table().schema().compositePartitionKey());
       } catch (final Exception e) {
           System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
           throw e;
       }
   }
```

以下に出力例を示します。

1. テーブルにディメンションタイプのパーティションキーがある

```
[PartitionKey(Type=DIMENSION, Name=hostId, EnforcementInRecord=OPTIONAL)]
```

2. テーブルにメジャー名タイプのパーティションキーがある

```
[PartitionKey(Type=MEASURE)]
```

3. 複合パーティションキーを指定せずに作成されたテーブルから複合パーティションキーを取得すると、 が返されます。

```
[PartitionKey(Type=MEASURE)]
```

Go v1

```
    <tablistentry>
     <tabname> Go </tabname>
     <tabcontent>
      <programlisting language="go"></programlisting>
     </tabcontent>
    </tablistentry>
```

以下に出力例を示します。

```
{
  Table: {
    Arn: "arn:aws:timestream:us-west-2:533139590831:database/devops/table/
host_metrics_dim_pk_1",
    CreationTime: 2023-05-31 01:52:00.511 +0000 UTC,
    DatabaseName: "devops",
    LastUpdatedTime: 2023-05-31 01:52:00.511 +0000 UTC,
    MagneticStoreWriteProperties: {
      EnableMagneticStoreWrites: true,
      MagneticStoreRejectedDataLocation: {
        S3Configuration: {
          BucketName: "timestream-sample-bucket-west",
          EncryptionOption: "SSE_S3",
          ObjectKeyPrefix: "TimeStreamCustomerSampleGo"
        }
      }
    },
    RetentionProperties: {
      MagneticStoreRetentionPeriodInDays: 73000,
      MemoryStoreRetentionPeriodInHours: 6
    },
    Schema: {
      CompositePartitionKey: [{
          EnforcementInRecord: "OPTIONAL",
          Name: "hostId",
```

```
          Type: "DIMENSION"
        }]
    },
    TableName: "host_metrics_dim_pk_1",
    TableStatus: "ACTIVE"
  }
}
```

Go v2

```
func (timestreamBuilder TimestreamBuilder) DescribeTable()
(*timestreamwrite.DescribeTableOutput, error) {
        describeTableInput := &timestreamwrite.DescribeTableInput{
            DatabaseName: aws.String(databaseName),
            TableName:    aws.String(tableName),
        }
       describeTableOutput, err :=
timestreamBuilder.WriteSvc.DescribeTable(context.TODO(), describeTableInput)

       if err != nil {
           fmt.Printf("Failed to describe table with Error: %s", err.Error())
       } else {
           fmt.Printf("Describe table is successful : %s\n",
JsonMarshalIgnoreError(*describeTableOutput))
           // If table is created with composite partition key, it will be included
in the output
       }

       return describeTableOutput, err
   }
```

以下に出力例を示します。

```
{
  "Table": {
    "Arn":"arn:aws:timestream:us-east-1:351861611069:database/cdpk-wr-db/table/
host_metrics_dim_pk",
    "CreationTime":"2023-05-31T22:36:10.66Z",
    "DatabaseName":"cdpk-wr-db",
    "LastUpdatedTime":"2023-05-31T22:36:10.66Z",
    "MagneticStoreWriteProperties":{
      "EnableMagneticStoreWrites":true,
      "MagneticStoreRejectedDataLocation":{
```

```
        "S3Configuration":{
          "BucketName":"error-configuration-sample-s3-bucket-cq8my",
          "EncryptionOption":"SSE_S3",
          "KmsKeyId":null,"ObjectKeyPrefix":null
        }
      }
    },
    "RetentionProperties":{
      "MagneticStoreRetentionPeriodInDays":73000,
      "MemoryStoreRetentionPeriodInHours":6
    },
    "Schema":{
      "CompositePartitionKey":[{
        "Type":"DIMENSION",
        "EnforcementInRecord":"OPTIONAL",
        "Name":"hostId"
      }]
    },
    "TableName":"host_metrics_dim_pk",
    "TableStatus":"ACTIVE"
  },
  "ResultMetadata":{}
}
```

Python

```
 def describe_table(self):
       print('Describing table')
       try:
           result = self.client.describe_table(DatabaseName=DATABASE_NAME,
TableName=TABLE_NAME)
           print("Table [%s] has id [%s]" % (TABLE_NAME, result['Table']['Arn']))
           # If table is created with composite partition key, it can be described
with
           # print(result['Table']['Schema'])
       except self.client.exceptions.ResourceNotFoundException:
           print("Table doesn't exist")
       except Exception as err:
           print("Describe table failed:", err)
```

以下に出力例を示します。

1. テーブルにディメンションタイプのパーティションキーがある

```
[{'CompositePartitionKey': [{'Type': 'DIMENSION', 'Name': 'hostId',
 'EnforcementInRecord': 'OPTIONAL'}]}]
```

2. テーブルにメジャー名タイプのパーティションキーがある

```
[{'CompositePartitionKey': [{'Type': 'MEASURE'}]}]
```

3. 複合パーティションキーを指定せずに作成されたテーブルから複合パーティションキーを取得する

```
[{'CompositePartitionKey': [{'Type': 'MEASURE'}]}]
```

パーティショニングスキーマ設定の更新

UpdateTable アクションSDKにアクセスできる を使用して、スキーマをパーティショニングするためのテーブル設定を更新できます。

パーティションキーを使用してテーブルを更新する

次のコードス二ペットを使用して、パーティションキーを使用してテーブルを更新できます。

Java

```
   public void updateTableCompositePartitionKeyEnforcement() {
       System.out.println("Updating table");

       UpdateTableRequest updateTableRequest = new UpdateTableRequest();
       updateTableRequest.setDatabaseName(DATABASE_NAME);
       updateTableRequest.setTableName(TABLE_NAME);

       // Can update enforcement level for dimension type partition key with
OPTIONAL or REQUIRED enforcement
       final List<PartitionKey> partitionKeyWithDimensionAndRequiredEnforcement =
Collections.singletonList(new PartitionKey()
           .withName(COMPOSITE_PARTITION_KEY_DIM_NAME)
           .withType(PartitionKeyType.DIMENSION)
           .withEnforcementInRecord(PartitionKeyEnforcementLevel.REQUIRED));
       Schema schema = new Schema();

schema.setCompositePartitionKey(partitionKeyWithDimensionAndRequiredEnforcement);
```

```
        updateTableRequest.withSchema(schema);

        writeClient.updateTable(updateTableRequest);
        System.out.println("Table updated");
```

Java v2

```
   public void updateTableCompositePartitionKeyEnforcement() {
        System.out.println("Updating table");
        // Can update enforcement level for dimension type partition key with
OPTIONAL or REQUIRED enforcement
        final List<PartitionKey> partitionKeyWithDimensionAndRequiredEnforcement =
Collections.singletonList(PartitionKey
            .builder()
            .name(COMPOSITE_PARTITION_KEY_DIM_NAME)
            .type(PartitionKeyType.DIMENSION)
            .enforcementInRecord(PartitionKeyEnforcementLevel.REQUIRED)
            .build());
        final Schema schema = Schema.builder()

.compositePartitionKey(partitionKeyWithDimensionAndRequiredEnforcement).build();
        final UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()

.databaseName(DATABASE_NAME).tableName(TABLE_NAME).schema(schema).build();

        writeClient.updateTable(updateTableRequest);
        System.out.println("Table updated");
```

Go v1

```
// Update table partition key enforcement attribute
   updateTableInput := &timestreamwrite.UpdateTableInput{
        DatabaseName: aws.String(*databaseName),
        TableName:    aws.String(*tableName),
        // Can update enforcement level for dimension type partition key with
OPTIONAL or REQUIRED enforcement
        Schema: &timestreamwrite.Schema{
            CompositePartitionKey: []*timestreamwrite.PartitionKey{
                {
                        Name:
aws.String(CompositePartitionKeyDimName),
                        EnforcementInRecord: aws.String("REQUIRED"),
                        Type:               aws.String("DIMENSION"),
```

```
                },
            }},
    }
    updateTableOutput, err := writeSvc.UpdateTable(updateTableInput)
        if err != nil {
            fmt.Println("Error:")
            fmt.Println(err)
        } else {
            fmt.Println("Update table is successful, below is the output:")
            fmt.Println(updateTableOutput)
        }
```

Go v2

```
// Update table partition key enforcement attribute
        updateTableInput := &timestreamwrite.UpdateTableInput{
            DatabaseName: aws.String(*databaseName),
            TableName:    aws.String(*tableName),
            // Can update enforcement level for dimension type partition key with
OPTIONAL or REQUIRED enforcement
            Schema: &types.Schema{
                CompositePartitionKey: []types.PartitionKey{
                    {
                        Name:
aws.String(CompositePartitionKeyDimName),
                        EnforcementInRecord:
types.PartitionKeyEnforcementLevelRequired,
                        Type:               types.PartitionKeyTypeDimension,
                    },
                }},
        }
        updateTableOutput, err :=
timestreamBuilder.WriteSvc.UpdateTable(context.TODO(), updateTableInput)
        if err != nil {
            fmt.Println("Error:")
            fmt.Println(err)
        } else {
            fmt.Println("Update table is successful, below is the output:")
            fmt.Println(updateTableOutput)
        }
```

Python

```
   def update_table(self):
       print('Updating table')
       try:
           # Can update enforcement level for dimension type partition key with
OPTIONAL or REQUIRED enforcement
           partition_key_with_dimension_and_required_enforcement = [
               {
                   'Type': 'DIMENSION',
                   'Name': COMPOSITE_PARTITION_KEY_DIM_NAME,
                   'EnforcementInRecord': 'REQUIRED'
               }
           ]
           schema = {
               'CompositePartitionKey':
partition_key_with_dimension_and_required_enforcement
           }
           self.client.update_table(DatabaseName=DATABASE_NAME,
TableName=TABLE_NAME,
                                    Schema=schema)
           print('Table updated.')
       except Exception as err:
           print('Update table failed:', err)
```

ユーザー定義パーティションキーの利点

クエリパフォーマンスの向上: ユーザー定義のパーティションキーを使用すると、クエリの実行を最適化し、全体的なクエリパフォーマンスを向上させることができます。クエリパターンに沿ったパーティションキーを定義することで、データスキャンを最小限に抑え、データプルーニングを最適化できるため、クエリのレイテンシーが短縮されます。

長期パフォーマンスの予測可能性の向上: ユーザー定義のパーティションキーにより、お客様はパーティション間でデータを均等に分散できるため、データ管理の効率が向上します。これにより、保存されたデータが時間の経過とともにスケールされるにつれて、クエリのパフォーマンスが安定します。

ユーザー定義パーティションキーの制限

LiveAnalytics ユーザーの Timestream として、カスタマーパーティションキーに関する制限に留意することが重要です。まず、ワークロードとクエリのパターンをよく理解する必要があります。つま

り、どのディメンションがクエリのメインフィルタリング条件として最も頻繁に使用されるかを明確に把握し、パーティションキーを最も効果的に使用できるようにカーディナリティが高い必要があります。

次に、パーティションキーはテーブルの作成時に定義する必要があり、既存のテーブルに追加することはできません。つまり、テーブルを作成する前に、パーティション戦略を慎重に検討して、ビジネスニーズに確実に一致させる必要があります。

最後に、テーブルが作成されると、後でパーティションキーを変更できないことに注意してください。つまり、パーティショニング戦略にコミットする前に、パーティショニング戦略を徹底的にテストして評価する必要があります。これらの制限を考慮して、Timestream のユーザー定義パーティションキーはクエリパフォーマンスと長期的な満足度を大幅に向上させることができます。

ユーザー定義のパーティションキーと低カーディナリティディメンション

特定のリージョンや状態など、カーディナリティが非常に低いパーティションキーを使用することにした場合、`customerID`、、 `ProductCategory`などの他のエンティティの のデータは、データがほとんどない、またはまったくないときに、多すぎるパーティションに分散される可能性があることに注意してください。これにより、クエリの実行が非効率になり、パフォーマンスが低下する可能性があります。

これを回避するには、キーフィルタリング条件の一部であるだけでなく、カーディナリティが高いディメンションを選択することをお勧めします。これにより、データがパーティション全体に均等に分散され、クエリのパフォーマンスが向上します。

既存のテーブルのパーティションキーの作成

の Timestream に既にテーブルがあり、ユーザー定義のパーティションキー LiveAnalytics を使用する場合は、目的のパーティションスキーマ定義を持つ新しいテーブルにデータを移行する必要があります。これは、S3 へのエクスポートとバッチロードを併用して行うことができます。これには、既存のテーブルから S3 へのデータのエクスポート、パーティションキー (必要に応じて) を含めるようにデータの変更、CSVファイルにヘッダーの追加、目的のパーティショニングスキーマが定義された新しいテーブルへのデータのインポートが含まれます。このメソッドは、特に大きなテーブルの場合、時間がかかり、コストがかかる場合があることに注意してください。

または、スケジュールされたクエリを使用して、目的のパーティショニングスキーマを持つ新しいテーブルにデータを移行することもできます。このメソッドでは、既存のテーブルから読み取り、新しいテーブルに書き込むスケジュールされたクエリを作成します。スケジュールされたクエリは、すべてのデータが移行されるまで定期的に実行されるように設定できます。移行プロセス中にデータの読み取りと書き込みを行うと、料金が発生することに注意してください。

カスタム複合パーティションキーを使用した LiveAnalytics スキーマ検証のタイムストリーム

の Timestream でのスキーマ検証 LiveAnalytics は、データベースに取り込まれたデータが指定されたスキーマに準拠し、取り込みエラーを最小限に抑え、データ品質を向上させるのに役立ちます。特に、スキーマの検証は、クエリパフォーマンスを最適化する目的でユーザー定義のパーティションキーを採用する場合に特に役立ちます。

ユーザー定義のパーティションキーを使用した LiveAnalytics スキーマ検証の Timestream とは？

LiveAnalytics スキーマ検証の Timestream は、事前定義されたスキーマに基づいてテーブルの LiveAnalytics Timestream に取り込まれるデータを検証する機能です。このスキーマは、パーティションキー、データ型、挿入されるレコードの制約など、データモデルを定義します。

ユーザー定義パーティションキーを使用する場合、スキーマの検証がさらに重要になります。パーティションキーを使用すると、パーティションキーを指定して、 の Timestream にデータを保存する方法を決定できます LiveAnalytics。カスタムパーティションキーを使用してスキーマに対して受信データを検証することで、データの整合性を強化し、エラーを早期に検出し、 の Timestream に保存されているデータの全体的な品質を向上させることができます LiveAnalytics。

カスタム複合パーティションキーで LiveAnalytics スキーマ検証に Timestream を使用する方法

カスタム複合パーティションキーで LiveAnalytics スキーマ検証に Timestream を使用するには、次の手順に従います。

クエリパターンがどのようになるかを考えてみましょう。 LiveAnalytics テーブルの Timestream のスキーマを適切に選択して定義するには、クエリ要件から始める必要があります。

カスタム複合パーティションキーを指定する: テーブルを作成するときに、カスタムパーティションキーを指定します。このキーは、テーブルデータのパーティション化に使用される属性を決定します。パーティショニングのディメンションキーとメジャーキーを選択できます。ディメンションキーはディメンション名に基づいてデータをパーティション化し、メジャーキーはメジャー名に基づいてデータをパーティション化します。

適用レベルの設定: 適切なデータパーティショニングとそれに伴う利点を確保するために、 の Amazon Timestream LiveAnalytics では、スキーマ内のパーティションキーごとに適用レベルを設定

できます。適用レベルは、レコードを取り込むときにパーティションキーディメンションが必要かオプションかを決定します。2 つのオプションから選択できます。つまりREQUIRED、パーティションキーが取り込まれたレコードに存在する必要があることを意味します。また、 OPTIONALはパーティションキーが存在する必要がないことを意味します。カスタマー定義パーティションを使用する場合は、REQUIRED適用レベルを使用して、データが適切にパーティション化され、この機能の利点を最大限に活用できるようにすることをお勧めします。さらに、スキーマの作成後はいつでも適用レベルの設定を変更して、データの取り込み要件に合わせることができます。

データの取り込み: テーブルの LiveAnalytics Timestream にデータを取り込む場合、スキーマ検証プロセスは、カスタム複合パーティションキーを使用して、定義されたスキーマとレコードをチェックします。レコードがスキーマに準拠していない場合、 の Timestream LiveAnalytics は検証エラーを返します。

検証エラーの処理： 検証エラーの場合、 の Timestream LiveAnalytics はエラーのタイプに応じてRejectedRecordsExceptionValidationExceptionまたは を返します。これらの例外はアプリケーション内で処理し、誤ったレコードの修正や取り込みの再試行など、適切なアクションを実行してください。

適用レベルの更新: 必要に応じて、 UpdateTableアクションを使用して、テーブルの作成後にパーティションキーの適用レベルを更新できます。ただし、名前やタイプなど、パーティションキー設定の一部の側面は、テーブルの作成後に変更できないことに注意してください。強制レベルをREQUIREDから に変更するとOPTIONAL、カスタマー定義パーティションキーとして選択された属性の有無に関係なく、すべてのレコードが受け入れられます。逆に、適用レベルを OPTIONALから に変更するとREQUIRED、この条件を満たさないレコードに対して 4xx 書き込みエラーが表示されることがあります。したがって、データのパーティショニング要件に基づいてテーブルを作成するときは、ユースケースに適した適用レベルを選択することが重要です。

カスタム複合パーティションキーで LiveAnalytics スキーマ検証に Timestream を使用するタイミング

データの整合性、品質、最適化されたパーティショニングが重要なシナリオでは、カスタム複合パーティションキーを使用した LiveAnalytics スキーマ検証のタイムストリームを使用する必要があります。データ取り込み中にスキーマを適用することで、誤った分析や貴重なインサイトの喪失につながる可能性のあるエラーや不整合を防ぐことができます。

バッチロードジョブとのやり取り

ユーザー定義のパーティションキーを使用してテーブルにデータをインポートするようにバッチロードジョブを設定する場合、プロセスに影響を与える可能性のあるシナリオがいくつかあります。

1. 強制レベルが に設定されている場合OPTIONAL、ジョブ設定中にパーティションキーがマッピングされていない場合、作成フロー中にコンソールにアラートが表示されます。このアラートは、APIまたは を使用する場合には表示されませんCLI。
2. 強制レベルが に設定されている場合REQUIRED、パーティションキーがソースデータ列にマッピングされない限り、ジョブの作成は拒否されます。
3. ジョブの作成REQUIRED後に強制レベルが に変更された場合、ジョブは引き続き実行されますが、パーティションキーに適切なマッピングがないレコードは 4xx エラーで拒否されます。

スケジュールされたクエリとのやり取り

アグリゲート、ロールアップ、その他の形式の前処理されたデータを、ユーザー定義のパーティションキーを持つテーブルに計算して保存するためのスケジュールされたクエリジョブを設定する場合、プロセスに影響を与える可能性のあるシナリオがいくつかあります。

1. 強制レベルが に設定されている場合OPTIONAL、ジョブ設定中にパーティションキーがマッピングされていない場合にアラートが表示されます。このアラートは、 APIまたは を使用する場合には表示されませんCLI。
2. 強制レベルが に設定されている場合REQUIRED、パーティションキーがソースデータ列にマッピングされない限り、ジョブの作成は拒否されます。
3. ジョブの作成REQUIRED後に強制レベルが に変更され、スケジュールされたクエリ結果にパーティションキーディメンションが含まれていない場合、ジョブの次の反復はすべて失敗します。

リソースへのタグとラベルの追加

タグ を使用して LiveAnalytics 、リソースの Amazon Timestream にラベルを付けることができます。タグを使用すると、目的別、所有者別、環境別など、さまざまな方法でリソースを分類できます。タグは、以下のことに役立ちます。

- リソースに割り当てたタグに基づいてリソースをすばやく特定します。
- タグ別に分類された AWS 請求書を参照してください。

タグ付けは、Amazon Elastic Compute Cloud (Amazon EC2) 、Amazon Simple Storage Service (Amazon S3) LiveAnalytics、Timestream for などの AWS サービスでサポートされています。効率的なタグ付けを行うと、特定のタグを持つサービス間でレポートを作成でき、コストインサイトを得ることができます。

タグ付けの使用を開始するには、次のことを行います。

1. タグ付けの制限を理解します。
2. タグ付けオペレーション を使用してタグを作成します。

最後に、最適のタグ付け手法に従うことをお勧めします。詳細については、「AWS タグ付け戦略」を参照してください。

タグ指定の制限

タグはそれぞれ、1 つのキーと 1 つの値で構成されており、どちらもお客様側が定義します。以下の制限が適用されます。

- LiveAnalytics テーブルの各 Timestream には、同じキーを持つタグを 1 つだけ持つことができます。既存のタグを追加しようとすると、既存のタグ値が新しい値に更新されます。
- 値は、タグカテゴリ内の記述子として機能します。値の Timestream LiveAnalytics では、空または null にすることはできません。
- タグのキーと値では、大文字と小文字が区別されます。
- キーの最大長は Unicode 文字で 128 文字です。
- 値の最大長は Unicode 文字で 256 文字です。
- 使用できる文字は、文字、ホワイトスペース、数字、および特殊文字 (+ - = . _ : /) です。
- リソースあたりのタグの最大数は 50 です。
- AWS- 割り当てられたタグ名と値には、割り当てることができない`aws:`プレフィックスが自動的に割り当てられます。AWS- 割り当てられたタグ名は、タグ制限の 50 にはカウントされません。ユーザー側で割り当てたタグ名は、`user:` というプレフィックスを付けてコスト配分レポートに表示されます。
- 過去にさかのぼってタグを適用することはできません。

タグ付けオペレーション

コンソール、クエリ言語、または AWS Command Line Interface () の Amazon Timestream を使用して、データベースとテーブルの LiveAnalyticsタグを追加、一覧表示、編集、または削除できます AWS CLI。

トピック

- コンソールを使用して新規または既存のデータベースやテーブルにタグを追加する

コンソールを使用して新規または既存のデータベースやテーブルにタグを追加する

Timestream for LiveAnalytics コンソールを使用して、新しいデータベース、テーブル、スケジュールされたクエリの作成時にタグを追加できます。既存のテーブルのタグの追加、一覧表示、編集、削除も実行できます。

データベースの作成時にデータベースにタグを付けるには (コンソール)

1. https://console.aws.amazon.com/timestream で Timestream コンソールを開きます。
2. ナビゲーションペインで [Databases] (データベース) を選択してから、[Create database] (データベースを作成) を選択します。
3. データベースの作成ページで、データベースの名前を指定します。タグのキーと値を入力して、[Add new tag (新しいタグの追加)]を選択します。
4. [データベースの作成] を選択します。

テーブルの作成時にテーブルにタグを付けるには (コンソール)

1. https://console.aws.amazon.com/timestream で Timestream コンソールを開きます。
2. ナビゲーションペインで [Tables] (テーブル) を選択して、[Create table (テーブルの作成)] を選択します。
3. LiveAnalytics テーブルの Timestream を作成するページで、テーブルの名前を指定します。タグのキーと値を入力し、新しいタグの追加 を選択します。
4. [Create table (テーブルの作成)] を選択します。

スケジュールされたクエリの作成時にタグ付けするには (コンソール)

1. https://console.aws.amazon.com/timestream で Timestream コンソールを開きます。
2. ナビゲーションペインで、スケジュールされたクエリ を選択し、スケジュールされたクエリ の作成 を選択します。
3. ステップ 3。クエリ設定ページを設定し、新しいタグの追加 を選択します。タグのキーと値を入力します。タグを追加するには、[Add new tag] (新しいタグを追加) を選択します。
4. [Next (次へ)] を選択します。

既存のリソースにタグを付けるには (コンソール)

1. https://console.aws.amazon.com/timestream で Timestream コンソールを開きます。

2. ナビゲーションペインで、データベース、テーブル、またはスケジュールされたクエリ を選択します。
3. リストでデータベースまたはテーブルを選択します。次に [Manage tags (タグの管理)] を選択して、タグの追加、編集または削除を行います。

タグ構造の詳細については、「タグ指定の制限」を参照してください。

の Timestream のセキュリティ LiveAnalytics

のクラウドセキュリティが最優先事項 AWS です。お客様は AWS 、最もセキュリティに敏感な組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャの恩恵を受けることができます。

セキュリティは、 AWS とユーザーの間で責任を共有します。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティとして説明しています。

- クラウドのセキュリティ – AWS クラウドで AWS サービスを実行するインフラストラクチャを保護する AWS 責任があります。 は、安全に使用できるサービス AWS も提供します。セキュリティの有効性は、AWS コンプライアンスプログラムの一環として、サードパーティーの審査機関によって定期的にテストおよび検証されています。Timestream for に適用されるコンプライアンスプログラムについては LiveAnalytics、AWS 「コンプライアンスプログラムによる対象範囲内のサービス」を参照してください。
- クラウドのセキュリティ – お客様の責任は、使用する AWS サービスによって決まります。また、お客様は、お客様のデータの機密性、組織の要件、および適用可能な法律および規制などの他の要因についても責任を担います。

このドキュメントは、 に Timestream を使用する場合に責任共有モデルを適用する方法を理解するのに役立ちます LiveAnalytics。次のトピックでは、セキュリティとコンプライアンスの目標を達成 LiveAnalytics するために の Timestream を設定する方法を示します。また、Timestream の LiveAnalytics リソースのモニタリングと保護に役立つ他の AWS サービスの使用方法についても説明します。

トピック

- の Timestream でのデータ保護 LiveAnalytics
- の Amazon Timestream のアイデンティティとアクセスの管理 LiveAnalytics
- の Timestream での口グ記録とモニタリング LiveAnalytics

- Amazon Timestream Live Analytics の耐障害性
- Amazon Timestream Live Analytics のインフラストラクチャセキュリティ
- Timestream の設定と脆弱性の分析
- の Timestream でのインシデント対応 LiveAnalytics
- VPC エンドポイント (AWS PrivateLink)
- の Amazon Timestream のセキュリティのベストプラクティス LiveAnalytics

の Timestream でのデータ保護 LiveAnalytics

責任 AWS 共有モデル 、Amazon Timestream Live Analytics のデータ保護に適用されます。このモデルで説明されているように、 AWS はすべての を実行するグローバルインフラストラクチャを保護する責任があります AWS クラウド。お客様は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。また、使用する AWS のサービス のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、「データプライバシーFAQ」を参照してください。欧州でのデータ保護の詳細については、AWS 「 責任共有モデル」とGDPRAWS 「セキュリティブログ」のブログ記事を参照してください。

データ保護の目的で、認証情報を保護し AWS アカウント 、 AWS IAM Identity Center または AWS Identity and Access Management () を使用して個々のユーザーを設定することをお勧めしますIAM。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。1.2 が必要でTLS、1.3 TLS をお勧めします。
- を使用して APIとユーザーアクティビティのログ記録を設定します AWS CloudTrail。証 CloudTrail 跡を使用して AWS アクティビティをキャプチャする方法については、AWS CloudTrail 「 ユーザーガイド」の CloudTrail 「証跡の操作」を参照してください。
- AWS 暗号化ソリューションと、 内のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。
- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは AWS を介して にアクセスするときに FIPS 140-3 検証済みの暗号化モジュールが必要な場合はAPI、FIPSエンドポイントを使用します。利用可能なFIPS エンドポイントの詳細については、「連邦情報処理標準 (FIPS) 140-3」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報は、タグ、または名前フィールドなどの自由形式のテキストフィールドに配置しないことを強くお勧めします。これには、コンソール、、または を使用して Timestream Live Analytics または他の AWS のサービス API AWS CLIを操作する場合が含まれます AWS SDKs。名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。URL を外部サーバーに提供する場合は、そのサーバーへのリクエストを検証URLするために認証情報を に含めないことを強くお勧めします。

保管時の暗号化やキー管理などの LiveAnalytics データ保護トピックの Timestream の詳細については、以下の利用可能なトピックのいずれかを選択してください。

トピック

- 保管中の暗号化
- 送信中の暗号化
- キーの管理

保管中の暗号化

保管時の LiveAnalytics 暗号化のタイムストリームは、 AWS Key Management Service （ AWS KMS ） に保存されている暗号化キーを使用して保管中のすべてのデータを暗号化することで、セキュリティを強化します。この機能は、機密データの保護における負担と複雑な作業を減らすのに役立ちます。保管時に暗号化することで、セキュリティを重視したアプリケーションを構築して、暗号化のコンプライアンスと規制の厳格な要件を満たすことができます。

- 暗号化は LiveAnalytics データベースの Timestream でデフォルトで有効になっており、オフにすることはできません。業界標準の AES-256 暗号化アルゴリズムは、使用されるデフォルトの暗号化アルゴリズムです。
- AWS KMS は、 の Timestream で保管中の暗号化に必要です LiveAnalytics。
- テーブル内の項目のサブセットのみを暗号化することはできません。
- 暗号化を使用するために、データベースのクライアントアプリケーションを変更する必要はありません。

キーを指定しない場合、 の Timestream は アカウント`alias/aws/timestream`で という名前の AWS KMS キー LiveAnalytics を作成して使用します。

で独自のカスタマーマネージドキーを使用してKMS、Timestream LiveAnalytics をデータの暗号化できます。の Timestream のキーの詳細については LiveAnalytics、「」を参照してくださいキーの管理。

の Timestream は、メモリストアとマグネティックストアの 2 つのストレージ階層にデータ LiveAnalytics を保存します。メモリストアデータは、 LiveAnalytics サービスキーの Timestream を使用して暗号化されます。マグネティックストアデータは、 キーを使用して暗号化されます AWS KMS。

Timestream クエリサービスには、データにアクセスするための認証情報が必要です。これらの認証情報は、 KMSキーを使用して暗号化されます。

> **Note**
>
> のタイムストリーム LiveAnalytics は、すべての復号化オペレーション AWS KMS に対してを呼び出すわけではありません。代わりに、アクティブなトラフィックで 5 分間キーのローカルキャッシュを維持します。アクセス許可の変更は、最大 5 分以内に最終的な整合性を持つ LiveAnalytics システムの Timestream を通じて伝達されます。

送信中の暗号化

Timestream Live Analytics のすべてのデータは、転送中に暗号化されます。デフォルトでは、 の Timestream との間で送受信されるすべての通信 LiveAnalytics は、Transport Layer Security (TLS) 暗号化を使用して保護されます。

キーの管理

Key AWS Management Service (AWS KMS) を使用して、Amazon Timestream Live Analytics のキーを管理できます。Timestream Live Analytics KMS では、 を使用してデータを暗号化する必要があります。キー管理には、キーに必要なコントロールの量に応じて、次のオプションがあります。

データベースとテーブルのリソース

- Timestream Live Analytics が管理するキー: キーを指定しない場合、Timestream Live Analytics はを使用して`alias/aws/timestream`キーを作成しますKMS。
- カスタマーマネージドキー: KMSカスタマーマネージドキーがサポートされています。キーのアクセス許可とライフサイクルをより詳細に制御する必要がある場合は、このオプションを選択します。これには、キーを毎年自動的にローテーションさせる機能が含まれます。

スケジュールされたクエリリソース

- Timestream Live Analytics が所有するキー: キーを指定しない場合、Timestream Live Analytics は独自のKMSキーを使用してクエリリソースを暗号化します。このキーはタイムストリームアカウントにあります。詳細については、 デKMSベロッパーガイドのAWS 「所有キー」を参照してください。
- カスタマーマネージドキー: KMSカスタマーマネージドキーがサポートされています。キーのアクセス許可とライフサイクルをより詳細に制御する必要がある場合は、このオプションを選択します。これには、キーを毎年自動的にローテーションさせる機能が含まれます。

KMS 外部キーストア (XKS) のキーはサポートされていません。

の Amazon Timestream のアイデンティティとアクセスの管理 LiveAnalytics

AWS Identity and Access Management （ IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を認証 (サインイン) し、誰に Timestream を LiveAnalytics リソースに使用する権限を付与 (アクセス許可を付与) できるかを制御します。IAM は追加料金なしで AWS のサービス 使用できる です。

トピック

- 対象者
- アイデンティティを使用した認証
- ポリシーを使用したアクセスの管理
- での Amazon Timestream の LiveAnalytics 仕組み IAM
- AWS Amazon Timestream Live Analytics の マネージドポリシー
- LiveAnalytics ID ベースのポリシーの Amazon Timestream の例
- LiveAnalytics ID とアクセスに関する Amazon Timestream のトラブルシューティング

対象者

AWS Identity and Access Management （ IAM) の使用方法は、 の Timestream で行う作業によって異なります LiveAnalytics。

サービスユーザー - LiveAnalytics サービスに Timestream を使用してジョブを実行する場合、管理者は必要な認証情報とアクセス許可を提供します。 LiveAnalytics 機能により多くの Timestream を使

用すると、追加のアクセス許可が必要になる場合があります。アクセスの管理方法を理解しておくと、管理者に適切な許可をリクエストするうえで役立ちます。の Timestream の機能にアクセスできない場合は LiveAnalytics、「」を参照してください LiveAnalytics ID とアクセスに関する Amazon Timestream のトラブルシューティング。

サービス管理者 – 社内の LiveAnalytics リソースの Timestream を担当している場合は、 の Timestream にフルアクセスできる可能性があります LiveAnalytics。サービスユーザーがアクセスする必要がある LiveAnalytics 機能やリソースの Timestream を決定するのはお客様の仕事です。その後、IAM管理者にリクエストを送信して、サービスユーザーのアクセス許可を変更する必要があります。このページの情報を確認して、 の基本概念を理解しますIAM。の Timestream IAMで を使用する方法の詳細については LiveAnalytics、「」を参照してくださいでの Amazon Timestream の LiveAnalytics 仕組み IAM。

IAM 管理者 - IAM管理者の場合は、 の Timestream へのアクセスを管理するためのポリシーの作成方法の詳細を知りたい場合があります LiveAnalytics。で使用できる LiveAnalytics アイデンティティベースのポリシーの Timestream の例を表示するにはIAM、「」を参照してください LiveAnalytics ID ベースのポリシーの Amazon Timestream の例。

アイデンティティを使用した認証

認証は、アイデンティティ認証情報 AWS を使用して にサインインする方法です。として、IAMユーザーとして AWS アカウントのルートユーザー、またはIAMロールを引き受けて認証 (にサインイン AWS) する必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。 AWS IAM Identity Center （ IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook 認証情報は、フェデレーティッド ID の例です。フェデレーティッド ID としてサインインすると、管理者は以前にIAMロールを使用して ID フェデレーションをセットアップしていました。フェデレーション AWS を使用して にアクセスすると、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、 AWS サインイン ユーザーガイドの「 へのサインイン方法 AWS アカウント」を参照してください。

AWS プログラムで にアクセスする場合、 はソフトウェア開発キット (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、認証情報を使用してリクエストに暗号化して署名します。 AWS ツールを使用しない場合は、自分でリクエストに署名する必要があります。推奨される方法を使用してリクエストに署名する方法の詳細については、 IAM ユーザーガイドのAWS API「リクエストの署名バージョン 4」を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。例えば、 では、アカウントのセキュリティを高めるために多要素認証 (MFA) を使用する AWS ことをお勧めします。詳細については、AWS IAM Identity Center 「 ユーザーガイド」の「多要素認証」とAWS 「 ユーザーガイド」の「多要素認証IAM」を参照してください。 IAM

IAM ユーザーとグループ

IAM ユーザーとは、1 人のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内の ID です。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を持つIAMユーザーを作成する代わりに、一時的な認証情報に依存することをお勧めします。ただし、IAMユーザーとの長期的な認証情報を必要とする特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、IAM「 ユーザーガイド」の「長期的な認証情報を必要とするユースケースのアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAMユーザーのコレクションを指定する ID です。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、という名前のグループがありIAMAdmins、そのグループにIAMリソースを管理するアクセス許可を付与できます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時認証情報が提供されます。詳細については、「 ユーザーガイド」のIAM「ユーザーのユースケース」を参照してください。 IAM

IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内の ID です。ユーザーと似ていますがIAM、特定の人物には関連付けられていません。でIAMロールを一時的に引き受けるには AWS Management Console、ユーザーからIAMロール (コンソール) に切り替えることができます。または AWS API オペレーションを AWS CLI 呼び出すか、カスタム を使用してロールを引き受けることができますURL。ロールを使用する方法の詳細については、 IAM ユーザーガイドの「ロールを引き受ける方法」を参照してください。

IAM 一時的な認証情報を持つ ロールは、以下の状況で役立ちます。

- フェデレーションユーザーアクセス – フェデレーティッド ID に許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーティッド ID が認証されると、その ID は

ロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションのロールの詳細については、 IAM ユーザーガイドの「サードパーティー ID プロバイダーのロールの作成」を参照してください。IAM Identity Center を使用する場合は、アクセス許可セットを設定します。ID が認証された後にアクセスできるものを制御するために、IAMIdentity Center はアクセス許可セットを のロールに関連付けますIAM。アクセス許可セットの詳細については、「AWS IAM Identity Center ユーザーガイド」の「アクセス許可セット」を参照してください。

- 一時的なIAMユーザーアクセス許可 – IAM ユーザーまたはロールは、特定のタスクに対して異なるアクセス許可を一時的に引き受けるIAMロールを引き受けることができます。
- クロスアカウントアクセス — IAMロールを使用して、別のアカウントの誰か (信頼できるプリンシパル) が自分のアカウントのリソースにアクセスすることを許可できます。クロスアカウントアクセスを許可する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、 (プロキシとしてロールを使用する代わりに) リソースに直接ポリシーをアタッチできます。クロスアカウントアクセスのロールとリソースベースのポリシーの違いについては、IAM「 ユーザーガイド」の「 のクロスアカウントリソースアクセスIAM」を参照してください。
- クロスサービスアクセス — 他の の機能 AWS のサービス を使用するものもあります AWS のサービス。例えば、 サービスで呼び出しを行う場合、そのサービスが Amazon でアプリケーションを実行EC2したりAmazon S3にオブジェクトを保存したりするのが一般的です。サービスでは、呼び出し元プリンシパルの許可、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用して でアクションを実行すると AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、を呼び出すプリンシパルのアクセス許可と AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストのリクエストを使用します。FAS リクエストは、サービスが他の AWS のサービス または リソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するための権限が必要です。FAS リクエストを行う際のポリシーの詳細については、「アクセスセッションの転送」を参照してください。
  - サービスロール – サービスロールは、ユーザーに代わってアクションを実行するためにサービスが引き受けるIAMロールです。IAM 管理者は、 内からサービスロールを作成、変更、削除できますIAM。詳細については、IAM「 ユーザーガイド」の「 にアクセス許可を委任するロールの作成 AWS のサービス」を参照してください。
  - サービスにリンクされたロール – サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカ

ウント 、 サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールのアクセス許可を表示することはできますが、編集することはできません。

- Amazon で実行されているアプリケーション EC2 – IAMロールを使用して、EC2インスタンスで実行され、 AWS CLI または AWS API リクエストを行うアプリケーションの一時的な認証情報を管理できます。これは、EC2インスタンス内にアクセスキーを保存するよりも望ましいです。EC2インスタンスに AWS ロールを割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロファイルには ロールが含まれており、EC2インスタンスで実行されているプログラムが一時的な認証情報を取得できるようにします。詳細については、IAM「 ユーザーガイド」のIAM「ロールを使用して Amazon EC2インスタンスで実行されているアプリケーションにアクセス許可を付与する」を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御するには、ポリシー AWS を作成し、 AWS ID またはリソースにアタッチします。ポリシーは AWS 、アイデンティティまたはリソースに関連付けられているときにアクセス許可を定義するオブジェクトです。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うときに、これらのポリシー AWS を評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。ほとんどのポリシーはJSONドキュメント AWS として に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、「 ユーザーガイド」のJSON「ポリシーの概要」を参照してください。 IAM

管理者はポリシーを使用して AWS JSON、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。必要なリソースに対してアクションを実行するアクセス許可をユーザーに付与するには、IAM管理者はIAMポリシーを作成できます。その後、管理者はIAMポリシーをロールに追加し、ユーザーはロールを引き受けることができます。

IAM ポリシーは、オペレーションの実行に使用する方法に関係なく、アクションのアクセス許可を定義します。例えば、`iam:GetRole`アクションを許可するポリシーがあるとします。そのポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS からロール情報を取得できますAPI。

アイデンティティベースのポリシー

ID ベースのポリシーは、IAMユーザー、ユーザーのグループ、ロールなどの ID にアタッチできるJSONアクセス許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行でき

るアクション、リソース、および条件をコントロールします。ID ベースのポリシーを作成する方法については、IAM「 ユーザーガイド」の「カスタマーマネージドポリシーによるカスタムIAMアクセス許可の定義」を参照してください。

アイデンティティベースのポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。マネージドポリシーは、 内の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには AWS 、管理ポリシーとカスタマー管理ポリシーが含まれます。マネージドポリシーまたはインラインポリシーを選択する方法については、IAM ユーザーガイドの「マネージドポリシーとインラインポリシーの選択」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースにアタッチするJSONポリシードキュメントです。リソースベースのポリシーの例としては、IAMロール信頼ポリシー と Amazon S3 バケットポリシー があります。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスをコントロールできます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーIAMでは、 から AWS 管理ポリシーを使用することはできません。

アクセスコントロールリスト (ACLs)

アクセスコントロールリスト (ACLs) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするためのアクセス許可を持っているかを制御します。ACLs はリソースベースのポリシーに似ていますが、JSONポリシードキュメント形式は使用していません。

Amazon S3、および Amazon VPCは AWS WAF、 をサポートするサービスの例ですACLs。の詳細についてはACLs、「Amazon Simple Storage Service デベロッパーガイド」の「アクセスコントロールリスト (ACL) 概要」を参照してください。

その他のポリシータイプ

AWS は、追加であまり一般的ではないポリシータイプをサポートします。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 – アクセス許可の境界は、アイデンティティベースのポリシーがIAMエンティティ (IAMユーザーまたはロール) に付与できる最大アクセス許可を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。Principal フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界の詳細については、IAM「 ユーザーガイド」のIAM「エンティティのアクセス許可の境界」を参照してください。
- サービスコントロールポリシー (SCPs) – SCPs は、 の組織または組織単位 (OU) の最大アクセス許可を指定するJSONポリシーです AWS Organizations。 AWS Organizations は、ビジネスが所有する複数の をグループ化して一元管理するためのサービス AWS アカウント です。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCPs) をアカウントの一部またはすべてに適用できます。は、各 を含むメンバーアカウントのエンティティのアクセス許可SCPを制限します AWS アカウントのルートユーザー。Organizations と の詳細についてはSCPs、AWS Organizations 「 ユーザーガイド」の「サービスコントロールポリシー」を参照してください。
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、「 ユーザーガイド」の「セッションポリシー」を参照してください。 IAM

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。が複数のポリシータイプが関与する場合にリクエストを許可するかどうか AWS を決定する方法については、「 ユーザーガイド」の「ポリシー評価ロジック」を参照してください。 IAM

での Amazon Timestream の LiveAnalytics 仕組み IAM

IAM を使用して の Timestream へのアクセスを管理する前に LiveAnalytics、 の Timestream で使用できるIAM機能を理解しておく必要があります LiveAnalytics。Timestream for LiveAnalytics およびその他の AWS のサービスが とどのように連携するかの概要についてはIAM、 IAM ユーザーガイドのAWS 「 と連携する のサービスIAM」を参照してください。

トピック

- ID ベースのポリシーの LiveAnalyticsタイムストリーム
- リソースベースのポリシーの LiveAnalyticsタイムストリーム
- LiveAnalytics タグの Timestream に基づく認可
- ロールの LiveAnalytics IAMタイムストリーム

ID ベースのポリシーの LiveAnalyticsタイムストリーム

IAM ID ベースのポリシーでは、許可または拒否されたアクションとリソース、およびアクションが許可または拒否される条件を指定できます。の Timestream は、特定のアクションとリソース、および条件キー LiveAnalytics をサポートします。JSON ポリシーで使用するすべての要素については、IAM ユーザーガイドのIAMJSON「ポリシー要素リファレンス」を参照してください。

アクション

管理者はポリシーを使用して AWS JSON、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

JSON ポリシーの `Action`要素は、ポリシー内のアクセスを許可または拒否するために使用できるアクションを記述します。ポリシーアクションは通常、関連付けられた AWS APIオペレーションと同じ名前になります。一致するAPIオペレーションがないアクセス許可のみのアクションなど、いくつかの例外があります。また、ポリシーに複数のアクションが必要なオペレーションもあります。これらの追加アクションは、依存アクションと呼ばれます。

このアクションは、関連付けられたオペレーションを実行するための権限を付与するポリシーで使用されます。

IAM ポリシーステートメントの Action 要素で次のアクションを指定できます。ポリシーを使用して、 でオペレーションを実行するアクセス許可を付与しますAWS。ポリシーでアクションを使用する場合、通常、同じ名前のAPIオペレーション、CLIコマンド、SQLまたはコマンドへのアクセスを許可または拒否します。

場合によっては、1 つのアクションで APIオペレーションと SQL コマンドへのアクセスを制御します。あるいは、いくつかのオペレーションはいくつかの異なるアクションを必要とします。

でサポートされている Timestream LiveAnalytics `Action`のリストについては、次の表を参照してください。

> **Note**
>
> すべてのデータベース固有の ではActions、特定のデータベースにアクションARNを制限するデータベースを指定できます。

アクション	説明	アクセスレベル	リソースタイプ (* 必須)
DescribeEndpoints	後続のリクエストを行う必要がある Timestream エンドポイントを返します。	すべて	*
選択	1 つ以上のテーブルからデータを選択する Timestream でクエリを実行します。詳細については、このメモを参照してください。	読み取り	テーブル*
CancelQuery	クエリをキャンセルします。	読み取り	*
ListTables	テーブルのリストを取得します。	リスト	データベース*
ListDatabases	データベースのリストを取得します。	リスト	*
ListMeasures	メジャーのリストを取得します。	読み取り	テーブル*
DescribeTable	テーブルの説明を取得します。	読み取り	テーブル*

アクション	説明	アクセスレベル	リソースタイプ (* 必須)
DescribeDatabase	データベースの説明を取得します。	読み取り	データベース*
SelectValues	特定のリソースを指定する必要がないクエリを実行します。詳細な説明については、このメモを参照してください。	読み取り	*
WriteRecords	Timestream にデータを挿入します。	書き込み	テーブル*
CreateTable	テーブルを作成します。	書き込み	データベース*
CreateDatabase	データベースを作成します。	書き込み	*
DeleteDatabase	データベースを削除します。	書き込み	*
UpdateDatabase	データベースを更新します。	書き込み	*
DeleteTable	テーブルを削除します。	書き込み	データベース*
UpdateTable	テーブルを更新します。	書き込み	データベース*

SelectValues と選択：

`SelectValues` は`Action`、リソースを必要としないクエリに使用される です。リソースを必要としないクエリの例を次に示します。

```
SELECT 1
```

このクエリは、 LiveAnalytics リソースの特定の Timestream を参照しないことに注意してください。別の例を考えてみましょう。

```
SELECT now()
```

このクエリは、 now()関数を使用して現在のタイムスタンプを返しますが、リソースを指定する必要はありません。 SelectValuesはテストによく使用されるため、 の Timestream LiveAnalytics はリソースなしでクエリを実行できます。次に、Selectクエリについて考えてみましょう。

```
SELECT * FROM database.table
```

このタイプのクエリでは、指定されたデータをテーブルから取得できるように、リソース、特に の LiveAnalytics table Timestream が必要です。

リソース

管理者はポリシーを使用して AWS JSON、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Resource JSON ポリシー要素は、アクションが適用されるオブジェクトを指定します。ステートメントには、Resource または NotResource 要素を含める必要があります。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの許可と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

Timestream では、 LiveAnalytics データベースとテーブルを IAM アクセス許可の Resource要素で使用できます。

LiveAnalytics データベースリソースの Timestream には、次の がありますARN。

```
arn:${Partition}:timestream:${Region}:${Account}:database/${DatabaseName}
```

LiveAnalytics テーブルリソースの Timestream には、次の がありますARN。

```
arn:${Partition}:timestream:${Region}:${Account}:database/${DatabaseName}/table/
${TableName}
```

の形式の詳細についてはARNs、「Amazon リソースネーム (ARNs) 」および AWS 「サービス名前空間」を参照してください。

例えば、 ステートメントでdatabaseキースペースを指定するには、次の を使用しますARN。

```
"Resource": "arn:aws:timestream:us-east-1:123456789012:database/mydatabase"
```

特定のアカウントに属するすべてのデータベースを指定するには、ワイルドカード (*) を使用します。

```
"Resource": "arn:aws:timestream:us-east-1:123456789012:database/*"
```

リソースを作成するための LiveAnalytics アクションなど、一部のアクションの Timestream は、特定のリソースで実行できません。このような場合は、ワイルドカード *を使用する必要があります。

```
"Resource": "*"
```

条件キー

の Timestream LiveAnalytics は、サービス固有の条件キーを提供しませんが、一部のグローバル条件キーの使用をサポートしています。すべての AWS グローバル条件キーを確認するには、 IAM ユーザーガイドのAWS 「グローバル条件コンテキストキー」を参照してください。

例

LiveAnalytics ID ベースのポリシーの Timestream の例を表示するには、「」を参照してください LiveAnalytics ID ベースのポリシーの Amazon Timestream の例。

リソースベースのポリシーの LiveAnalyticsタイムストリーム

の Timestream LiveAnalytics はリソースベースのポリシーをサポートしていません。詳細なリソースベースのポリシーページの例を表示するには、https://docs.aws.amazon.com/lambda/latest/dg/access-control-resource-based.html を参照してください。

LiveAnalytics タグの Timestream に基づく認可

タグを使用して LiveAnalytics 、リソースの Timestream へのアクセスを管理できます。タグに基づいてリソースアクセスを管理するには、`timestream:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。 LiveAnalytics リソースの Timestream のタグ付けの詳細については、「」を参照してくださいthe section called “ リソースのタグ付け”。

リソースのタグに基づいてリソースへのアクセスを制限するためのアイデンティティベースのポリシーの例を表示するには、「 タグに基づく LiveAnalytics リソースアクセスのタイムストリーム」を参照してください。

ロールの LiveAnalytics IAMタイムストリーム

IAM ロールは、特定のアクセス許可を持つ AWS アカウント内のエンティティです。

の Timestream での一時的な認証情報の使用 LiveAnalytics

一時的な認証情報を使用して、フェデレーションでサインインしたり、IAMロールを引き受けたり、クロスアカウントロールを引き受けたりすることができます。AssumeRole や などのオペレーションを呼び出す AWS STS APIことで、一時的なセキュリティ認証情報を取得しますGetFederationToken。

サービスリンクロール

の Timestream LiveAnalytics は、サービスにリンクされたロールをサポートしていません。

サービスロール

の Timestream LiveAnalytics はサービスロールをサポートしていません。

AWS Amazon Timestream Live Analytics の マネージドポリシー

AWS マネージドポリシーは、 によって作成および管理されるスタンドアロンポリシーです AWS。AWS マネージドポリシーは、多くの一般的なユースケースに対するアクセス許可を提供するように設計されているため、ユーザー、グループ、ロールへのアクセス許可の割り当てを開始できます。

AWS マネージドポリシーは、すべての AWS 顧客が使用できるため、特定のユースケースに対して最小権限のアクセス許可を付与しない場合があることに注意してください。ユースケース別にカスタマーマネージドポリシーを定義して、マネージドポリシーを絞り込むことをお勧めします。

AWS マネージドポリシーで定義されているアクセス許可は変更できません。が マネージドポリシーで AWS 定義されたアクセス許可 AWS を更新すると、ポリシーがアタッチされているすべてのプリンシパル ID (ユーザー、グループ、ロール) に影響します。 AWS は、新しい が起動されるか、新しいAPIオペレーション AWS のサービス が既存のサービスで使用できるようになったときに、 AWS マネージドポリシーを更新する可能性が最も高いです。

詳細については、「 ユーザーガイド」のAWS 「 マネージドポリシー」を参照してください。 IAM

トピック

- AWS マネージドポリシー： AmazonTimestreamReadOnlyAccess
- AWS マネージドポリシー： AmazonTimestreamConsoleFullAccess
- AWS マネージドポリシー： AmazonTimestreamFullAccess
- AWS マネージドポリシーへの Timestream Live Analytics の更新

AWS マネージドポリシー： AmazonTimestreamReadOnlyAccess

ユーザー、グループおよびロールに `AmazonTimestreamReadOnlyAccess` をアタッチできます。このポリシーは、Amazon Timestream への読み取り専用アクセスを提供します。

アクセス許可の詳細

このポリシーには、以下の許可が含まれています。

- `Amazon Timestream` – Amazon Timestream への読み取り専用アクセスを提供します。このポリシーは、実行中のクエリをキャンセルするアクセス許可も付与します。

このポリシーを JSON 形式で確認するには、「」を参照してくださいAmazonTimestreamReadOnlyAccess。

AWS マネージドポリシー： AmazonTimestreamConsoleFullAccess

ユーザー、グループおよびロールに `AmazonTimestreamConsoleFullAccess` をアタッチできます。

このポリシーは、 を使用して Amazon Timestream を管理するためのフルアクセスを提供します AWS Management Console。また、このポリシーは、保存されたクエリを管理するための特定の AWS KMS オペレーションおよびオペレーションに対するアクセス許可も付与します。

アクセス許可の詳細

このポリシーには、以下の権限が含まれています。

- `Amazon Timestream` – Amazon Timestream へのフルアクセスをプリンシパルに付与します。
- `AWS KMS` – プリンシパルがエイリアスを一覧表示し、キーを記述できるようにします。
- `Amazon S3` – プリンシパルがすべての Amazon S3 バケットを一覧表示できるようにします。
- `Amazon SNS` – プリンシパルに Amazon SNSトピックを一覧表示することを許可します。
- `IAM` – プリンシパルがIAMロールを一覧表示できるようにします。
- `DBQMS` — プリンシパルにクエリへのアクセス、作成、削除、記述、および更新を許可します。Database Query Metadata Service (dbqms) は内部専用サービスです。これは、Amazon Timestream を含む複数の の のクエリエディタ AWS Management Console に対して AWS のサービス、最近のクエリと保存されたクエリを提供します。

このポリシーを JSON 形式で確認するには、「」を参照してくださいAmazonTimestreamConsoleFullAccess。

AWS マネージドポリシー： AmazonTimestreamFullAccess

ユーザー、グループおよびロールに `AmazonTimestreamFullAccess` をアタッチできます。

このポリシーは、Amazon Timestream へのフルアクセスを提供します。このポリシーは、特定の AWS KMS オペレーションに対するアクセス許可も付与します。

アクセス許可の詳細

このポリシーには、以下の権限が含まれています。

- `Amazon Timestream` – Amazon Timestream へのフルアクセスをプリンシパルに付与します。
- `AWS KMS` – プリンシパルがエイリアスを一覧表示し、キーを記述できるようにします。
- `Amazon S3` – プリンシパルがすべての Amazon S3 バケットを一覧表示できるようにします。

このポリシーを JSON 形式で確認するには、「」を参照してくださ
いAmazonTimestreamFullAccess。

AWS マネージドポリシーへの Timestream Live Analytics の更新

このサービスがこれらの変更の追跡を開始してからの Timestream Live Analytics の AWS マネージドポリシーの更新に関する詳細を表示します。このページの変更に関する自動アラートについては、Timestream Live Analytics Document 履歴ページのRSSフィードにサブスクライブします。

変更	説明	日付
AmazonTimestreamReadOnlyAccess - 既存ポリシーへの更新	`timestream:DescribeAccountSettings` アクションを既存の `AmazonTimestreamReadOnlyAccess` マネージドポリシーに追加しました。このアクションは、 AWS アカウント 設定の説明に使用されます。 Timestream Live Analytics は、 `Sid`フィールドを追加することで、このマネージドポリシーも更新しました。 ポリシーの更新は、`AmazonTimestreamReadOnlyAccess` マネージドポリシーの使用には影響しません。	2024 年 6 月 3 日
AmazonTimestreamReadOnlyAccess – 既存ポリシーへの更新	既存の`AmazonTimestreamReadOnlyAccess` 管理ポリシーに `timestream:DescribeBatchLoadTask` および `timestream:ListBat`	2023 年 2 月 24 日

変更	説明	日付
	chLoadTasks アクションを追加しました。これらのアクションは、バッチロードタスクを一覧表示および記述するときに使用されます。 ポリシーの更新は、AmazonTimestreamReadOnlyAccess マネージドポリシーの使用には影響しません。	
AmazonTimestreamReadOnlyAccess – 既存ポリシーへの更新	既存のAmazonTimestreamReadOnlyAccess 管理ポリシーに timestream:DescribeScheduledQuery および timestream:ListScheduledQueries アクションを追加しました。これらのアクションは、既存のスケジュールされたクエリを一覧表示および記述するときに使用されます。 ポリシーの更新は、AmazonTimestreamReadOnlyAccess マネージドポリシーの使用には影響しません。	2021 年 11 月 29 日

変更	説明	日付
AmazonTimestreamConsoleFullAccess – 既存ポリシーへの更新	`s3:ListAllMyBuckets` アクションを既存の `AmazonTimestreamConsoleFullAccess` マネージドポリシーに追加しました。このアクションは、Timestream に Amazon S3 バケットを指定してマグネティックストアの書き込みエラーを記録する場合に使用されます。 ポリシーの更新は、`AmazonTimestreamConsoleFullAccess` マネージドポリシーの使用には影響しません。	2021 年 11 月 29 日
AmazonTimestreamFullAccess – 既存ポリシーへの更新	既存の `AmazonTimestreamFullAccess` マネージドポリシーに `s3:ListAllMyBuckets` アクションを追加しました。このアクションは、Timestream に Amazon S3 バケットを指定してマグネティックストアの書き込みエラーを記録する場合に使用されます。 ポリシーの更新は、`AmazonTimestreamFullAccess` マネージドポリシーの使用には影響しません。	2021 年 11 月 29 日

変更	説明	日付
AmazonTimestreamConsoleFullAccess – 既存ポリシーへの更新	既存のAmazonTimestreamConsoleFullAccess 管理ポリシーから冗長アクションを削除しました。以前は、このポリシーには冗長アクション が含まれていましたdbqms:DescribeQueryHistory 。更新されたポリシーは、冗長アクションを削除します。 ポリシーの更新は、AmazonTimestreamConsoleFullAccess マネージドポリシーの使用には影響しません。	2021 年 4 月 23 日
Timestream Live Analytics が変更の追跡を開始	Timestream Live Analytics は AWS 、管理ポリシーの変更の追跡を開始しました。	2021 年 4 月 21 日

LiveAnalytics ID ベースのポリシーの Amazon Timestream の例

デフォルトでは、IAMユーザーとロールには、 LiveAnalytics リソースの Timestream を作成または変更するアクセス許可がありません。また、 AWS Management Console、、CQLSH、 AWS CLIまたは を使用してタスクを実行することはできません AWS API。IAM 管理者は、ユーザーとロールに必要な特定のリソースに対して特定のAPIオペレーションを実行するアクセス許可を付与するIAMポリシーを作成する必要があります。その後、管理者はこれらのアクセス許可を必要とするIAMユーザーまたはグループにこれらのポリシーをアタッチする必要があります。

これらのポリシードキュメント例を使用して IAM ID ベースのJSONポリシーを作成する方法については、IAM「 ユーザーガイド」のJSON「 タブでのポリシーの作成」を参照してください。

トピック

- ポリシーのベストプラクティス

- LiveAnalytics コンソールでの Timestream の使用
- 自分の権限の表示をユーザーに許可する
- の Timestream での一般的なオペレーション LiveAnalytics
- タグに基づく LiveAnalytics リソースアクセスのタイムストリーム
- スケジュールされたクエリ

ポリシーのベストプラクティス

ID ベースのポリシーは、アカウント内の LiveAnalytics リソースの Timestream を作成、アクセス、または削除できるかどうかを決定します。これらのアクションを実行すると、 AWS アカウントに料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS マネージドポリシーを開始し、最小権限のアクセス許可に移行 – ユーザーとワークロードへのアクセス許可の付与を開始するには、多くの一般的なユースケースのアクセス許可を付与するAWS マネージドポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義することで、アクセス許可をさらに減らすことをお勧めします。詳細については、IAM「 ユーザーガイド」の「 管理AWS ポリシー」またはジョブ機能の 管理ポリシーを参照してください。 AWS
- 最小権限のアクセス許可を適用する - IAMポリシーでアクセス許可を設定する場合、タスクの実行に必要なアクセス許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用してアクセス許可を適用する方法の詳細については、IAM「 ユーザーガイド」の「 のポリシーとアクセス許可IAM」を参照してください。
- IAM ポリシーの条件を使用してアクセスをさらに制限する – ポリシーに条件を追加して、アクションとリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを を使用して送信する必要があることを指定できますSSL。また、 などの特定の を通じてサービスアクションが使用されている場合 AWS のサービス、 条件を使用してサービスアクションへのアクセスを許可することもできます AWS CloudFormation。詳細については、IAM「 ユーザーガイド」のIAMJSON「ポリシー要素: 条件」を参照してください。
- IAM Access Analyzer を使用してIAMポリシーを検証し、安全で機能的なアクセス許可を確保する – IAM Access Analyzer は、ポリシーがポリシー言語 (JSON) とIAMベストプラクティスに準拠するように、新規および既存のIAMポリシーを検証します。IAM Access Analyzer には、安全で機能的なポリシーの作成に役立つ 100 を超えるポリシーチェックと実用的なレコメンデーションが用意されています。詳細については、IAM「 ユーザーガイド」のIAM「Access Analyzer を使用したポリシーの検証」を参照してください。

- 多要素認証が必要 (MFA) – でIAMユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、 をオンにMFAしてセキュリティを強化します。API オペレーションが呼び出されるMFAタイミングを要求するには、ポリシーにMFA条件を追加します。詳細については、「 ユーザーガイド」の「 によるセキュアAPIアクセスMFA」を参照してください。 IAM

のベストプラクティスの詳細についてはIAM、「 ユーザーガイド」の「 のセキュリティのベストプラクティスIAM」を参照してください。 IAM

LiveAnalytics コンソールでの Timestream の使用

の Timestream LiveAnalytics では、 LiveAnalytics コンソールの Amazon Timestream にアクセスするための特定のアクセス許可は必要ありません。 AWS アカウントの LiveAnalytics リソースの Timestream の詳細を一覧表示および表示するには、少なくとも読み取り専用のアクセス許可が必要です。最低限必要なアクセス許可よりも制限の厳しいアイデンティティベースのポリシーを作成すると、コンソールは、そのポリシーを持つエンティティ (IAMユーザーまたはロール) に対して意図したとおりに機能しません。

自分の権限の表示をユーザーに許可する

この例では、IAMユーザーがユーザー ID にアタッチされているインラインポリシーとマネージドポリシーを表示できるようにするポリシーを作成する方法を示します。このポリシーには、コンソールで、または AWS CLI または を使用してプログラムでこのアクションを実行するアクセス許可が含まれています AWS API。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
```

```
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

の Timestream での一般的なオペレーション LiveAnalytics

以下は、Timestream for LiveAnalytics Service で一般的なオペレーションを許可するIAMポリシーの例です。

トピック

- すべてのオペレーションを許可する
- SELECT オペレーションの許可
- 複数のリソースに対するSELECTオペレーションの許可
- メタデータオペレーションの許可
- INSERT オペレーションの許可
- CRUD オペレーションの許可
- リソースを指定せずにクエリをキャンセルし、データを選択する
- データベースの作成、記述、削除、記述
- リストされたデータベースをタグで制限する{"Owner": "${username}"}
- データベース内のすべてのテーブルを一覧表示する
- テーブルでの作成、説明、削除、更新、および選択
- テーブルによるクエリの制限

すべてのオペレーションを許可する

以下は、 の Timestream のすべてのオペレーションを許可するサンプルポリシーです LiveAnalytics。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:*"
            ],
            "Resource": "*"
        }
    ]
}
```

SELECT オペレーションの許可

次のサンプルポリシーでは、特定のリソースに対して SELECTスタイルのクエリを許可します。

> **Note**
>
> を Amazon アカウント ID <account_ID>に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:Select",
                "timestream:DescribeTable",
                "timestream:ListMeasures"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/
table/DevOps"
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "timestream:DescribeEndpoints",
                "timestream:SelectValues",
                "timestream:CancelQuery"
            ],
            "Resource": "*"
        }
    ]
}
```

複数のリソースに対するSELECTオペレーションの許可

次のサンプルポリシーでは、複数のリソースに対して SELECTスタイルのクエリを許可します。

> **Note**
>
> を Amazon アカウント ID <account_ID>に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:Select",
                "timestream:DescribeTable",
                "timestream:ListMeasures"
            ],
            "Resource": [
                "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/table/
DevOps",
                "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/table/
DevOps1",
                "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/table/
DevOps2"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "timestream:DescribeEndpoints",
                "timestream:SelectValues",
                "timestream:CancelQuery"
            ],
            "Resource": "*"
        }
    ]
}
```

メタデータオペレーションの許可

次のサンプルポリシーでは、ユーザーはメタデータクエリを実行できますが、 の Timestream で実際のデータを読み取りまたは書き込みするオペレーションは実行できません LiveAnalytics。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:DescribeEndpoints",
                "timestream:DescribeTable",
                "timestream:ListMeasures",
                "timestream:SelectValues",
                "timestream:ListTables",
                "timestream:ListDatabases",
                "timestream:CancelQuery"
            ],
            "Resource": "*"
        }
    ]
}
```

INSERT オペレーションの許可

次のサンプルポリシーでは、ユーザーがアカウント database/sampleDB/table/DevOpsで で INSERTオペレーションを実行することを許可しています<account_id>。

> **Note**
>
> を Amazon アカウント ID <account_ID>に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "timestream:WriteRecords"
            ],
            "Resource": [
                "arn:aws:timestream:us-east-1:<account_id>:database/sampleDB/table/
DevOps"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "timestream:DescribeEndpoints"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

CRUD オペレーションの許可

次のサンプルポリシーでは、ユーザーが の Timestream でCRUDオペレーションを実行できます LiveAnalytics。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:DescribeEndpoints",
                "timestream:CreateTable",
                "timestream:DescribeTable",
                "timestream:CreateDatabase",
                "timestream:DescribeDatabase",
                "timestream:ListTables",
                "timestream:ListDatabases",
                "timestream:DeleteTable",
                "timestream:DeleteDatabase",
```

```
                "timestream:UpdateTable",
                "timestream:UpdateDatabase"
            ],
            "Resource": "*"
        }
    ]
}
```

リソースを指定せずにクエリをキャンセルし、データを選択する

次のサンプルポリシーでは、ユーザーはクエリをキャンセルし、リソース仕様を必要としないデータに対してSelectクエリを実行できます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:SelectValues",
                "timestream:CancelQuery"
            ],
            "Resource": "*"
        }
    ]
}
```

データベースの作成、記述、削除、記述

次のサンプルポリシーでは、ユーザーがデータベース を作成、説明、削除、説明できますsampleDB。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:CreateDatabase",
                "timestream:DescribeDatabase",
                "timestream:DeleteDatabase",
                "timestream:UpdateDatabase"
            ],
```

```
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB"
        }
    ]
}
```

リストされたデータベースをタグで制限する**{"Owner": "${username}"}**

次のサンプルポリシーでは、キー値ペア でタグ付けされたすべてのデータベースを一覧表示できます{"Owner": "${username}"}。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:ListDatabases"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "${aws:username}"
                }
            }
        }
    ]
}
```

データベース内のすべてのテーブルを一覧表示する

データベース内のすべてのテーブルを一覧表示する次のサンプルポリシーsampleDB：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:ListTables"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/"
        }
    ]
```

```
}
```

テーブルでの作成、説明、削除、更新、および選択

次のサンプルポリシーでは、ユーザーがデータベース DevOps のテーブルの作成、テーブルの説明、テーブルの削除、テーブルの更新、テーブルに対するSelectクエリの実行を行うことができますsampleDB。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:CreateTable",
                "timestream:DescribeTable",
                "timestream:DeleteTable",
                "timestream:UpdateTable",
                "timestream:Select"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/
table/DevOps"
        }
    ]
}
```

テーブルによるクエリの制限

次のサンプルポリシーでは、ユーザーはデータベース DevOpsを除くすべてのテーブルをクエリできますsampleDB。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:Select"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/
table/*"
        },
        {
```

```
            "Effect": "Deny",
            "Action": [
                "timestream:Select"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/
table/DevOps"
        }
    ]
}
```

タグに基づく LiveAnalytics リソースアクセスのタイムストリーム

ID ベースのポリシーの条件を使用して、タグに基づいて LiveAnalytics リソースの Timestream へのアクセスを制御できます。このセクションでは、いくつかの例を紹介します。

次の例で、テーブルの Owner にユーザーのユーザー名の値が含まれている場合に、そのテーブルを表示するためのアクセス許可をユーザーに付与するポリシーを作成する方法について説明します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadOnlyAccessTaggedTables",
            "Effect": "Allow",
            "Action": "timestream:Select",
            "Resource": "arn:aws:timestream:us-east-2:111122223333:database/mydatabase/
table/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "${aws:username}"
                }
            }
        }
    ]
}
```

このポリシーは、アカウントのIAMユーザーにアタッチできます。という名前のユーザーが LiveAnalytics テーブルの Timestream を表示richard-roeしようとする場合、テーブルには Owner=richard-roeまたは のタグを付ける必要がありますowner=richard-roe。それ以外の場合、アクセスは拒否されます。条件キー名では大文字と小文字は区別されないため、条件タグキー Owner は Owner と owner に一致します。詳細については、IAM「 ユーザーガイド」のIAMJSON「ポリシー要素: 条件」を参照してください。

次のポリシーは、リクエストに渡されたタグにキーOwnerと値 がある場合に、タグを持つテーブルを作成するアクセス許可をユーザーに付与しますusername。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateTagTableUser",
            "Effect": "Allow",
            "Action": [
                "timestream:Create",
                "timestream:TagResource"
            ],
            "Resource": "arn:aws:timestream:us-east-2:111122223333:database/mydatabase/
table/*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:RequestTag/Owner": "${aws:username}"
                }
            }
        }
    ]
}
```

以下のポリシーでは、 env タグが devまたは に設定されている任意のデータベースDescribeDatabaseAPIで の使用を許可しますtest。

```
{ "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDescribeEndpoints",
      "Effect": "Allow",
      "Action": [
        "timestream:DescribeEndpoints"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowDevTestAccess",
      "Effect": "Allow",
      "Action": [
        "timestream:DescribeDatabase"
      ],
```

```
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "timestream:tag/env": [
              "dev",
              "test"
          ]
        }
      }
    }
  ]
}
{ "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowTagAccessForDevResources",
      "Effect": "Allow",
      "Action": [
        "timestream:TagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/env": [
            "test",
            "dev"
          ]
        }
      }
    }
  ]
}
```

このポリシーは、 Condition キーを使用して、 キーenvと の値を持つタグをリソースtestqadevに追加できるようにします。

スケジュールされたクエリ

一覧表示、削除、更新、実行 ScheduledQuery

次のサンプルポリシーでは、ユーザーがスケジュールされたクエリを一覧表示、削除、更新、実行できます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:DeleteScheduledQuery",
                "timestream:ExecuteScheduledQuery",
                "timestream:UpdateScheduledQuery",
                "timestream:ListScheduledQueries",
                "timestream:DescribeEndpoints"
            ],
            "Resource": "*"
        }
    ]
}
```

CreateScheduledQuery カスタマーマネージドKMSキーの使用

次のサンプルポリシーでは、ユーザーがカスタマーマネージドKMSキーを使用して暗号化されたスケジュールされたクエリを作成することを許可します。*<keyid for ScheduledQuery>*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:role/ScheduledQueryExecutionRole"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "timestream:CreateScheduledQuery",
                "timestream:DescribeEndpoints"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
```

```
            "Action": [
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:us-west-2:123456789012:key/<keyid for
 ScheduledQuery>",
            "Effect": "Allow"
        }
    ]
}
```

DescribeScheduledQuery カスタマーマネージドKMSキーの使用

次のサンプルポリシーでは、ユーザーがカスタマーマネージドKMSキーを使用して作成されたスケジュールされたクエリを記述できます。*<keyid for ScheduledQuery>*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "timestream:DescribeScheduledQuery",
                "timestream:DescribeEndpoints"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:us-west-2:123456789012:key/<keyid for
 ScheduledQuery>",
            "Effect": "Allow"
        }
    ]
}
```

実行ロールのアクセス許可 (スケジュールされたクエリにはカスタマーマネージドKMSキーを使用し、エラーレポートには SSE-KMS を使用します)

次のサンプルポリシーを、スケジュールされたクエリ暗号化とエラーレポートのSSE-KMS暗号化にカスタマーマネージドKMSキーCreateScheduledQueryAPIを使用する のScheduledQueryExecutionRoleArnパラメータで指定されたIAMロールにアタッチします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "kms:GenerateDataKey",
            ],
            "Resource": "arn:aws:kms:us-west-2:123456789012:key/<keyid for
 ScheduledQuery>",
            "Effect": "Allow"
        },
        {
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-west-2:123456789012:key/<keyid for database-1>",
                "arn:aws:kms:us-west-2:123456789012:key/<keyid for database-n>",
                "arn:aws:kms:us-west-2:123456789012:key/<keyid for ScheduledQuery>"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:us-west-2:123456789012:scheduled-query-notification-topic-
*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "timestream:Select",
                "timestream:SelectValues",
```

```
                "timestream:WriteRecords"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:PutObject",
                "s3:GetBucketAcl"
            ],
            "Resource": [
                "arn:aws:s3:::scheduled-query-error-bucket",
                "arn:aws:s3:::scheduled-query-error-bucket/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

実行ロールの信頼関係

以下は、 の CreateScheduledQuery ScheduledQueryExecutionRoleArnパラメータで指定されたIAMロールの信頼関係ですAPI。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "timestream.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

アカウント内で作成されたすべてのスケジュールされたクエリへのアクセスを許可する

次のサンプルポリシーを の ScheduledQueryExecutionRoleArnパラメータで指定されたIAMロールにアタッチCreateScheduledQueryAPIして、アカウント内で作成されたすべてのスケジュールされたクエリへのアクセスを許可します。*Account_ID*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "timestream.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "Account_ID"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:timestream:us-
west-2:Account_ID:scheduled-query/*"
                }
            }
        }
    ]
}
```

特定の名前でスケジュールされたすべてのクエリへのアクセスを許可する

以下のサンプルポリシーを の ScheduledQueryExecutionRoleArnパラメータで指定されたIAMロールにアタッチCreateScheduledQueryAPIして、 で始まる名前でスケジュールされたすべてのクエリへのアクセスを許可します。*Scheduled_Query_Name*アカウント内 *Account_ID*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "timestream.amazonaws.com"
            },
```

```
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "Account_ID"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:timestream:us-
west-2:Account_ID:scheduled-query/Scheduled_Query_Name*"
                }
            }
        }
    ]
}
```

LiveAnalytics ID とアクセスに関する Amazon Timestream のトラブルシューティング

および の Timestream を使用する際に発生する可能性のある一般的な問題を診断 LiveAnalytics して修正するには、以下の情報を使用しますIAM。

トピック

- の Timestream でアクションを実行する権限がありません LiveAnalytics
- iam を実行する権限がありません。PassRole
- 自分の AWS アカウント外のユーザーに、 LiveAnalytics リソースの Timestream へのアクセスを許可したい

の Timestream でアクションを実行する権限がありません LiveAnalytics

がアクションを実行する権限がないと AWS Management Console 通知した場合は、管理者に連絡してサポートを依頼する必要があります。管理者とは、サインイン認証情報を提供した担当者です。

次のエラー例は、mateojacksonIAMユーザーがコンソールを使用して の詳細を表示しようとすると発生します。*table* ただし、 にはテーブルに対するtimestream:*Select*アクセス許可がありません。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 timestream:Select on resource: mytable
```

この場合、Mateo は、timestream:*Select* アクションを使用して *mytable* リソースへのアクセスが許可されるように、管理者にポリシーの更新を依頼します。

iam を実行する権限がありません。PassRole

`iam:PassRole` アクションを実行する権限がないというエラーが表示された場合は、 の Timestream にロールを渡すことができるようにポリシーを更新する必要があります LiveAnalytics。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、そのサービスに既存のロールを渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

次のエラー例は、 という名前のIAMユーザーがコンソールを使用して の Timestream `marymajor` でアクションを実行しようとすると発生します LiveAnalytics。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。メアリーには、ロールをサービスに渡す許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに `iam:PassRole` アクションの実行を許可する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン認証情報を提供した担当者が管理者です。

自分の AWS アカウント外のユーザーに、 LiveAnalytics リソースの Timestream へのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACLs) をサポートするサービスでは、これらのポリシーを使用して、リソースへのアクセスをユーザーに許可できます。

詳細については、以下を参照してください。

- Timestream for がこれらの機能 LiveAnalytics をサポートしているかどうかについては、「」を参照してください[での Amazon Timestream の LiveAnalytics 仕組み IAM]。
- 所有 AWS アカウント している のリソースへのアクセスを提供する方法については、 IAM ユーザーガイドの[「所有 AWS アカウント している別の のIAMユーザーへのアクセスを提供する]」を参照してください。

- サードパーティー にリソースへのアクセスを提供する方法については AWS アカウント、 IAM ユーザーガイドの「サードパーティー AWS アカウント が所有する へのアクセスを提供する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、 IAM ユーザーガイドの「外部認証されたユーザーへのアクセスを提供する (ID フェデレーション)」を参照してください。
- クロスアカウントアクセスにロールとリソースベースのポリシーを使用する違いについては、 IAM ユーザーガイドの「 のクロスアカウントリソースアクセスIAM」を参照してください。

の Timestream でのログ記録とモニタリング LiveAnalytics

モニタリングは、 LiveAnalytics および AWS ソリューションの Timestream の信頼性、可用性、パフォーマンスを維持する上で重要です。マルチポイント障害が発生した場合に簡単にデバッグできるように、 AWS ソリューションのすべての部分からモニタリングデータを収集する必要があります。ただし、 の Timestream のモニタリングを開始する前に LiveAnalytics、以下の質問に対する回答を含むモニタリング計画を作成する必要があります。

- モニタリングの目的は何ですか?
- どのリソースをモニタリングしますか?
- どのくらいの頻度でこれらのリソースをモニタリングしますか?
- どのモニタリングツールを利用しますか?
- 誰がモニタリングタスクを実行しますか?
- 問題が発生したときに誰が通知を受け取りますか?

次のステップでは、さまざまな時間およびさまざまな負荷条件で LiveAnalytics パフォーマンスを測定することで、環境のパフォーマンスの通常の Timestream のベースラインを確立します。の Timestream をモニタリングするときに LiveAnalytics、履歴モニタリングデータを保存して、現在のパフォーマンスデータと比較し、通常のパフォーマンスパターンとパフォーマンス異常を特定し、問題に対処する方法を考案できるようにします。

ベースラインを確立するには、少なくとも次の項目をモニタリングする必要があります。

- システムエラー。リクエストでエラーが発生したかどうかを判断できます。

トピック

- モニタリングツール

- を使用した通話の LiveAnalytics API Timestream のログ記録 AWS CloudTrail

モニタリングツール

AWS には、 の Timestream をモニタリングするために使用できるさまざまなツールが用意されています LiveAnalytics。これらのツールの一部はモニタリングを行うように設定できますが、一部のツールは手動による介入が必要です。モニタリングタスクをできるだけ自動化することをお勧めします。

トピック

- 自動モニタリングツール
- 手動モニタリングツール

自動モニタリングツール

次の自動モニタリングツールを使用して Timestream を監視 LiveAnalytics して、問題が発生したときに報告できます。

- Amazon CloudWatch アラーム – 指定した期間に 1 つのメトリクスを監視し、複数の期間にわたって特定のしきい値に対するメトリクスの値に基づいて 1 つ以上のアクションを実行します。アクションは、Amazon Simple Notification Service (Amazon SNS) トピックまたは Amazon EC2 Auto Scaling ポリシーに送信される通知です。 CloudWatch アラームは、単に特定の状態にあるという理由だけでアクションを呼び出すわけではありません。状態は変更され、指定された期間維持されている必要があります。詳細については、「Amazon によるモニタリング CloudWatch」を参照してください。

手動モニタリングツール

の Timestream をモニタリングするもう 1 つの重要な部分 LiveAnalytics には、 CloudWatch アラームでカバーされない項目を手動でモニタリングすることが含まれます。 LiveAnalytics、 CloudWatch Trusted Advisor、およびその他の AWS Management Console ダッシュボードの Timestream は、at-a-glance AWS 環境の状態を表示します。

- CloudWatch ホームページには以下が表示されます。
  - 現在のアラームとステータス
  - アラームとリソースのグラフ
  - サービスのヘルスステータス

さらに、 CloudWatch を使用して以下を実行できます。

- 重視するサービスをモニタリングするためのカスタマイズしたダッシュボードを作成します
- メトリクスデータをグラフ化して、問題のトラブルシューティングを行い、傾向を確認する
- すべての AWS リソースメトリクスを検索して参照する
- 問題があることを通知するアラームを作成/編集する

を使用した通話の LiveAnalytics API Timestream のログ記録 AWS CloudTrail

の Timestream LiveAnalytics は AWS CloudTrail、ユーザー、ロール、または Timestream for の AWS サービスによって実行されたアクションの記録を提供するサービス と統合されています LiveAnalytics。 CloudTrail は Timestream for をイベント LiveAnalytics としてAPI呼び出します (DDL) 。キャプチャされる呼び出しには、 LiveAnalytics コンソールの Timestream からの呼び出しと、オペレーションの LiveAnalytics API Timestream へのコード呼び出しが含まれます。証跡を作成する場合は、 の CloudTrail Timestream のイベントなど、Amazon Simple Storage Service (Amazon S3) バケットへのイベントの継続的な配信を有効にできます LiveAnalytics。証跡を設定しない場合でも、 CloudTrail コンソールでイベント履歴 の最新のイベントを表示できます。によって収集された情報を使用して CloudTrail、 の Timestream に対して行われたリクエスト LiveAnalytics、リクエスト元の IP アドレス、リクエスト者、リクエスト日時、その他の詳細を確認できます。

の詳細については CloudTrail、AWS CloudTrail 「 ユーザーガイド」を参照してください。

での LiveAnalytics 情報のタイムストリーム CloudTrail

CloudTrail AWS アカウントを作成すると、 はアカウントで有効になります。の Timestream でアクティビティが発生すると LiveAnalytics、そのアクティビティは CloudTrail イベント履歴 の他の AWS サービスイベントとともにイベントに記録されます。 AWS アカウントで最近のイベントを表示、検索、ダウンロードできます。詳細については、「イベント履歴 での CloudTrail イベントの表示」を参照してください。

> **Warning**
>
> 現在、 の Timestream はすべての管理とQueryAPIオペレーションの CloudTrail イベント LiveAnalytics を生成しますが、 `WriteRecords` と `DescribeEndpoints` のイベントは生成しませんAPIs。

の Timestream のイベントなど、 AWS アカウント内のイベントを継続的に記録するには LiveAnalytics、証跡を作成します。証跡により CloudTrail 、 はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成すると、証跡はすべてのAWSリージョンに適用されます。証跡は、 AWS パーティション内のすべてのリージョンからのイベントを口グに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、 CloudTrail ログで収集されたイベントデータをさらに分析して対処するように、他の AWS サービスを設定できます。

詳細については、『AWS CloudTrail ユーザーガイド:』の以下のトピックを参照してください。

- 追跡作成の概要
- CloudTrail サポートされているサービスと統合
- の Amazon SNS Notifications の設定 CloudTrail
- 複数のリージョンからの CloudTrail ログファイルの受信
- 複数のアカウントからの CloudTrail ログファイルの受信
- データイベントのログ記録

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。アイデンティティ情報は、以下を判別するのに役立ちます:

- リクエストがルートまたは AWS Identity and Access Management （ IAM) ユーザー認証情報を使用して行われたかどうか
- リクエストが、ロールとフェデレーティッドユーザーのどちらの一時的なセキュリティ認証情報を使用して送信されたか
- リクエストが別の AWS サービスによって行われたかどうか

詳細については、CloudTrail userIdentity「 要素」を参照してください。

Query API イベントの場合 :

- すべてのイベントを受信する証跡を作成するか、 LiveAnalytics リソースタイプ `AWS::Timestream::Database` または の Timestream でイベントを選択します`AWS::Timestream::Table`。
- Query API データベースやテーブルにアクセスしないリクエスト、または不正な形式のクエリ文字列が原因で検証例外が発生するリクエストは、リソースタイプ`AWS::Timestream::Database`と ARN の値 CloudTrail で に記録されます。

```
arn:aws:timestream:(region):(accountId):database/NO_RESOURCE_ACCESSED
```

これらのイベントは、リソースタイプ のイベントを受信する証跡にのみ配信されますAWS::Timestream::Database。

Amazon Timestream Live Analytics の耐障害性

AWS グローバルインフラストラクチャは、 AWS リージョンとアベイラビリティーゾーンを中心に構築されています。 AWS リージョンは、低レイテンシー、高スループット、および冗長性の高いネットワークに接続された、物理的に分離され、分離された複数のアベイラビリティーゾーンを提供します。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性が高く、フォールトトレラントで、スケーラブルです。

AWS リージョンとアベイラビリティーゾーンの詳細については、AWS 「グローバルインフラストラクチャ」を参照してください。

で利用可能な Timestream のデータ保護機能の詳細については AWS Backup、「」を参照してくださいの使用 AWS Backup。

Amazon Timestream Live Analytics のインフラストラクチャセキュリティ

マネージドサービスである Amazon Timestream Live Analytics は、Amazon Web Services: セキュリティプロセスの概要ホワイトペーパーで説明されている AWS グローバルネットワークセキュリティ手順によって保護されています。

ネットワーク経由で Timestream Live Analytics にアクセスするには、 AWS 公開されたAPI呼び出しを使用します。クライアントは Transport Layer Security (TLS) 1.0 以降をサポートする必要があります。1.2 TLS 以降をお勧めします。また、クライアントは、エフェメラルディフィヘルマン (PFS) や楕円曲線エフェメラルディフィヘルマン () など、完全なフォワードシークレット (DHE) を持つ暗号スイートもサポートする必要がありますECDHE。これらのモードは、Java 7 以降など、ほとんどの最新システムでサポートされています。

さらに、リクエストは、アクセスキー ID とプリンシIAMパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時セキュリティ認証情報を生成し、リクエストに署名することもできます。

Timestream Live Analytics は、Timestream Live Analytics インスタンスが存在する特定の AWS リージョンにトラフィックが分離されるように設計されています。

Timestream の設定と脆弱性の分析

設定と IT コントロールは、 AWS お客様とお客様の責任を共有します。詳細については、 AWS 「 責任共有モデル」を参照してください。責任共有モデルに加えて、 LiveAnalytics ユーザーの Timestream は以下に注意する必要があります。

- 関連するクライアント側の依存関係を使用して、クライアントアプリケーションにパッチを適用するのはお客様の責任です。
- お客様は、必要に応じてペネトレーションテストを検討する必要があります (https://aws.amazon.com/security/「ペネトレーションテスト/」を参照) 。

の Timestream でのインシデント対応 LiveAnalytics

LiveAnalytics サービスインシデントの Amazon Timestream は、Personal Health Dashboard で報告されます。ダッシュボードの詳細については、 AWS Health こちら を参照してください。

の Timestream は、 を使用したレポート LiveAnalytics をサポートします AWS CloudTrail。詳細については、「を使用した通話の LiveAnalytics API Timestream のログ記録 AWS CloudTrail」を参照してください。

VPC エンドポイント (AWS PrivateLink)

インターフェイスVPCエンドポイント を作成 LiveAnalytics することで、 VPCと の Amazon Timestream 間にプライベート接続を確立できます。インターフェイスエンドポイントは、インターネットゲートウェイAWS PrivateLink、NATデバイス、VPN接続、または AWS Direct Connect 接続なしで の LiveAnalytics APIs Timestream にプライベートにアクセスできるテクノロジーであるを搭載しています。のインスタンスは、 の Timestream と通信するためにパブリック IP アドレスを必要としVPCません LiveAnalytics APIs。VPC と の LiveAnalytics Timestream 間のトラフィックは、Amazon ネットワークを離れません。

各インターフェースエンドポイントは、サブネット内の 1 つ以上の Elastic Network Interface によって表されます。インターフェイスVPCエンドポイントの詳細については、「Amazon ユーザーガイド」の「インターフェイスVPCエンドポイント (AWS PrivateLink) 」を参照してください。 VPC

Timestream for LiveAnalytics および VPCエンドポイントの使用を開始するには、VPCエンドポイント LiveAnalytics を使用した の Timestream に関する特定の考慮事項、 の Timestream のインター

フェイスVPCエンドポイントの作成 LiveAnalytics、 の Timestream のVPCエンドポイントポリシーの作成 LiveAnalytics、エンドポイントを使用した Timestream クライアント (書き込みまたはクエリSDKのいずれか) の使用について説明しますVPC。

トピック

- Timestream でのVPCエンドポイントの仕組み
- の Timestream のインターフェイスVPCエンドポイントの作成 LiveAnalytics
- の Timestream のVPCエンドポイントポリシーの作成 LiveAnalytics

Timestream でのVPCエンドポイントの仕組み

Timestream Write または Timestream Query のいずれかにアクセスするためのVPCエンドポイントを作成するとSDK、すべてのリクエストは Amazon ネットワーク内のエンドポイントにルーティングされ、パブリックインターネットにはアクセスしません。具体的には、リクエストは、特定のリージョンでアカウントがマッピングされたセルの書き込みエンドポイントとクエリエンドポイントにルーティングされます。Timestream のセルラーアーキテクチャとセル固有のエンドポイントの詳細については、「」を参照してくださいセルラーアーキテクチャ。例えば、アカウントが `cell1`で にマッピングされ`us-west-2`、書き込み (`ingest-cell1.timestream.us-west-2.amazonaws.com`) とクエリ () のVPCインターフェイスエンドポイントをセットアップしたとします`query-cell1.timestream.us-west-2.amazonaws.com`。この場合、これらのエンドポイントを使用して送信された書き込みリクエストはすべて Amazon ネットワーク内に完全に保持され、パブリックインターネットにはアクセスされません。

Timestream VPCエンドポイントに関する考慮事項

Timestream のVPCエンドポイントを作成するときは、次の点を考慮してください。

- Timestream for のインターフェイスVPCエンドポイントを設定する前に LiveAnalytics、Amazon VPC ユーザーガイドの「インターフェイスエンドポイントのプロパティと制限」を確認してください。
- の Timestream LiveAnalytics は、 からのすべてのAPIアクションへの呼び出しをサポートしますVPC。
- VPC エンドポイントポリシーは、 の Timestream でサポートされています LiveAnalytics。デフォルトでは、 の Timestream へのフルアクセス LiveAnalytics はエンドポイントを通じて許可されます。詳細については、「Amazon VPCユーザーガイド」のVPC「エンドポイントを使用したサービスへのアクセスの制御」を参照してください。

- Timestream のアーキテクチャにより、書き込みアクションとクエリアクションの両方にアクセスするには、 ごとに 1 つずつ、2 つのVPCインターフェイスエンドポイントを作成する必要がありますSDK。さらに、セルエンドポイントを指定する必要があります (マッピングされている Timestream セルのエンドポイントのみを作成できます) 。詳細については、このガイドの「Timestream のインターフェイスVPCエンドポイントの作成 LiveAnalytics」セクションを参照してください。

これで、 の Timestream がVPCエンドポイントとどのように LiveAnalytics 連携するかを理解できたので、 の Timestream のインターフェイスVPCエンドポイントを作成します LiveAnalytics。

の Timestream のインターフェイスVPCエンドポイントの作成 LiveAnalytics

Amazon VPCコンソールまたは AWS Command Line Interface () を使用して LiveAnalytics 、Timestream for Service のインターフェイスVPCエンドポイントを作成できますAWS CLI。Timestream のVPCエンドポイントを作成するには、以下で説明する Timestream 固有のステップを実行します。

> **Note**
>
> 以下のステップを完了する前に、Timestream VPCエンドポイントに関する具体的な考慮事項を理解してください。

Timestream セルを使用してVPCエンドポイントサービス名を作成する

Timestream の一意のアーキテクチャのため、インターフェイスVPCエンドポイントは SDK (書き込みとクエリ) ごとに個別に作成する必要があります。さらに、Timestream セルエンドポイントを指定する必要があります (マップされている Timestream セルのエンドポイントのみを作成できます) 。Interface VPC Endpoints を使用して 内から Timestream に直接接続するにはVPC、以下の手順を実行します。

1. まず、利用可能な Timestream セルエンドポイントを見つけます。使用可能なセルエンドポイントを検索するには、 `DescribeEndpoints`アクション (書き込みとクエリの両方から使用可能 APIs) を使用して、Timestream アカウントで使用可能なセルエンドポイントを一覧表示します。詳細については、例を参照してください。
2. 使用するセルエンドポイントを選択したら、Timestream Write または Query のインターフェイスVPCエンドポイント文字列を作成しますAPI。
   - 書き込みの場合API :

```
com.amazonaws.<region>.timestream.ingest-<cell>
```

- クエリの場合API：

```
com.amazonaws.<region>.timestream.query-<cell>
```

この場合、次のようになります。*<region>* は有効な AWS リージョンコードであり、*<cell>* は、アクション によって Endpoints オブジェクトで返されるセルエンドポイントアドレス (cell1や などcell2) の 1 つです。 DescribeEndpoints 詳細については、例を参照してください。

3. VPC エンドポイントサービス名を作成したら、インターフェイスエンドポイント を作成します。VPC エンドポイントサービス名の入力を求められたら、ステップ 2 で作成したVPCエンドポイントサービス名を使用します。

例: VPCエンドポイントサービス名の作成

次の例では、 us-west-2 リージョンの書き込みを使用して AWS CLIAPIで DescribeEndpointsアクションを実行します。

```
aws timestream-write describe-endpoints --region us-west-2
```

このコマンドは、次の出力を返します。

```
{
    "Endpoints": [
        {
            "Address": "ingest-cell1.timestream.us-west-2.amazonaws.com",
            "CachePeriodInMinutes": 1440
        }
    ]
}
```

この場合、*cell1* は *<cell>* および *us-west-2* は *<region>*。 したがって、結果のVPCエンドポイントサービス名は次のようになります。

```
com.amazonaws.us-west-2.timestream.ingest-cell1
```

の Timestream 用のインターフェイスVPCエンドポイントを作成したら LiveAnalytics、 の Timestream 用のVPCエンドポイントポリシーを作成します LiveAnalytics。

の Timestream のVPCエンドポイントポリシーの作成 LiveAnalytics

の Timestream へのアクセスを制御するエンドポイントポリシーをVPCエンドポイントにアタッチできます LiveAnalytics。このポリシーでは、以下の情報を指定します。

- アクションを実行できるプリンシパル。
- 実行可能なアクション。
- このアクションを実行できるリソース。

詳細については、「Amazon VPCユーザーガイド」のVPC「エンドポイントを使用したサービスへのアクセスの制御」を参照してください。

例: LiveAnalytics アクションの Timestream のVPCエンドポイントポリシー

の Timestream のエンドポイントポリシーの例を次に示します LiveAnalytics。エンドポイントにアタッチされると、このポリシーは、すべてのリソースのすべてのプリンシパルの LiveAnalytics アクション (この場合は `ListDatabases`) に対して、リストされている Timestream へのアクセスを許可します。

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "timestream:ListDatabases"
         ],
         "Resource":"*"
      }
   ]
}
```

の Amazon Timestream のセキュリティのベストプラクティス LiveAnalytics

Amazon Timestream for LiveAnalytics には、独自のセキュリティポリシーを開発および実装する際に考慮すべきセキュリティ機能が多数用意されています。以下のベストプラクティスは一般的なガ

イドラインであり、完全なセキュリティソリューションを提供するものではありません。これらのベストプラクティスはお客様の環境に必ずしも適切または十分でない可能性があるため、処方箋ではなく、あくまで有用な検討事項とお考えください。

トピック

- LiveAnalytics 予防セキュリティのベストプラクティスのタイムストリーム

LiveAnalytics 予防セキュリティのベストプラクティスのタイムストリーム

以下のベストプラクティスは、 の Timestream でセキュリティインシデントを予測および防止するのに役立ちます LiveAnalytics。

保管中の暗号化

のタイムストリームは、 AWS Key Management Service （ AWS KMS ） に保存されている LiveAnalytics 暗号化キーを使用して、テーブルに保存されているすべてのユーザーデータを保管時に暗号化します。この機能は、基になるストレージへの不正アクセスからデータを保護することによって、データ保護の追加レイヤーを提供します。

の Timestream LiveAnalytics は、すべてのテーブルを暗号化するために 1 つのサービスのデフォルトキー (AWS 所有CMK) を使用します。このキーが存在しない場合は、自動的に作成されます。サービスデフォルトキーは無効にできません。詳細については、「Timestream for LiveAnalytics Encryption at Rest」を参照してください。

IAM ロールを使用して の Timestream へのアクセスを認証する LiveAnalytics

ユーザー、アプリケーション、およびその他の AWS サービスが の Timestream にアクセスするには LiveAnalytics、リクエスト AWS APIに有効な AWS 認証情報を含める必要があります。AWS 認証情報をアプリケーションまたはEC2インスタンスに直接保存しないでください。自動更新されない長期認証情報条件のため、漏洩すると業務に深刻な悪影響が及ぶ可能性があります。IAM ロールを使用すると、 サービスとリソースへのアクセス AWS に使用できる一時的なアクセスキーを取得できます。

詳細については、「IAM ロール」を参照してください。

LiveAnalytics ベース認証IAMに Timestream ポリシーを使用する

アクセス許可を付与するときは、アクセス許可を取得するユーザー、アクセス許可APIsを取得する Timestream LiveAnalytics、およびそれらのリソースで許可する特定のアクションを決定します。最小限の特権の実装は、セキュリティリスクはもちろん、エラーや悪意ある行動によってもたらされる可能性のある影響を減らす上での鍵となります。

IAM アクセス許可ポリシーを ID (ユーザー、グループ、ロール) にアタッチし、リソースの Timestream でオペレーションを実行するアクセス許可を付与します LiveAnalytics。

これを行うには、次を使用します。

- AWS マネージド (事前定義) ポリシー
- カスタマー管理ポリシー
- タグベースの認証

クライアント側の暗号化を考慮する

の機密データまたは機密データを Timestream に保存する場合は LiveAnalytics、そのデータをオリジンにできるだけ近い場所に暗号化して、ライフサイクルを通じてデータを保護することをお勧めします。転送中および保管中の機密データを暗号化すると、プレーンテキストデータがサードパーティーに公開されないようになります。

他の サービスでの使用

Amazon Timestream for は、さまざまな AWS サービスや一般的なサードパーティーツールと LiveAnalytics 統合されています。現在、 の Timestream は、以下との統合 LiveAnalytics をサポートしています。

トピック

- Amazon DynamoDB
- AWS Lambda
- AWS IoT Core
- Amazon Managed Service for Apache Flink
- Amazon Kinesis
- Amazon MQ
- Amazon MSK
- Amazon QuickSight
- Amazon SageMaker
- Amazon SQS
- DBeaver を使用して Amazon Timestream を操作する
- Grafana
- SquaredUp を使用して Amazon Timestream を操作する

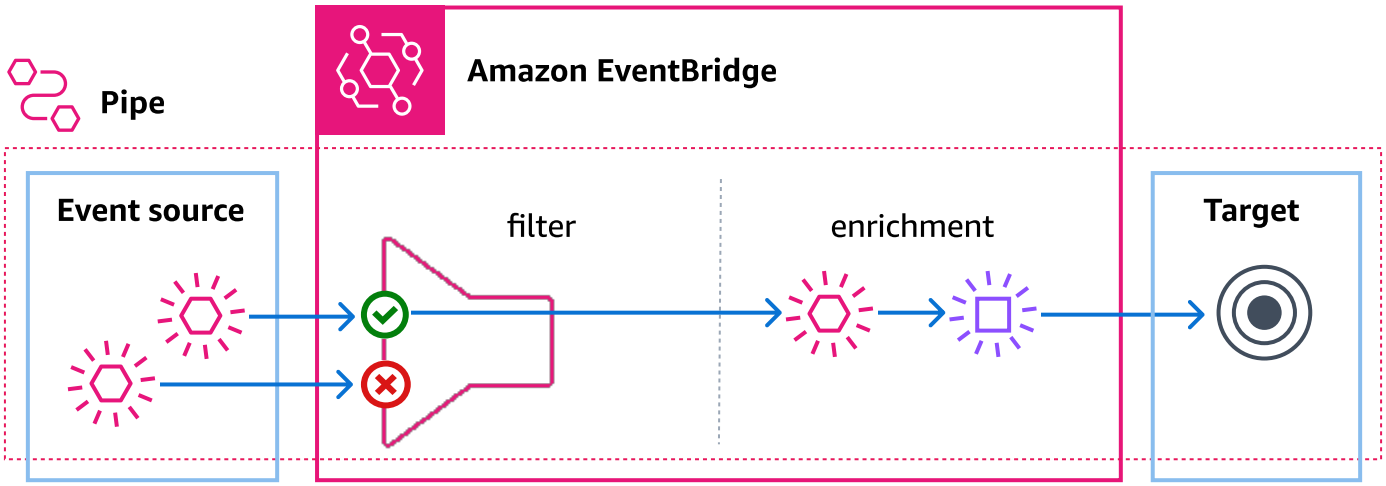
- オープンソース Telegraf
- JDBC
- ODBC
- VPC エンドポイント (AWS PrivateLink)

Amazon DynamoDB

EventBridge Pipes を使用して DynamoDB データを に送信する Timestream

EventBridge Pipes を使用して、DynamoDB ストリームから LiveAnalytics テーブルの Amazon Timestream にデータを送信できます。

パイプは、サポートされているソースとターゲットの統合を目的として point-to-pointおり、高度な変換とエンリッチメントをサポートしています。Pipes は、イベント駆動型アーキテクチャを開発する際の専門知識と統合コードの必要性を減らします。パイプをセットアップするには、ソースを選択し、オプションのフィルタリングを追加し、オプションのエンリッチメントを定義し、イベントデータのターゲットを選択します。



EventBridge パイプの詳細については、EventBridge 「 ユーザーガイド」のEventBridge 「パイプ」を参照してください。 LiveAnalytics テーブルの Amazon Timestream にイベントを配信するようにパイプを設定する方法については、EventBridge 「Pipes ターゲットの詳細」を参照してください。

AWS Lambda

の Timestream を操作する Lambda 関数を作成できます LiveAnalytics。例えば、定期的に実行される Lambda 関数を作成して Timestream でクエリを実行し、1 つ以上の基準を満たすクエリ結果に基

づいてSNS通知を送信できます。Lambda の詳細については、AWS 「Lambda ドキュメント」を参照してください。

トピック

- Amazon Timestream for LiveAnalytics with Python を使用して AWS Lambda 関数を構築する
- LiveAnalytics で の Amazon Timestream を使用して AWS Lambda 関数を構築する JavaScript
- Amazon Timestream for LiveAnalytics with Go を使用して AWS Lambda 関数を構築する
- C# LiveAnalytics で の Amazon Timestream を使用して AWS Lambda 関数を構築する

Amazon Timestream for LiveAnalytics with Python を使用して AWS Lambda 関数を構築する

Amazon Timestream for LiveAnalytics with Python を使用して AWS Lambda 関数を構築するには、以下の手順に従います。

1. で説明されているように、 が Timestream Service にアクセスするために必要なアクセス許可を付与することを Lambda が引き受けるIAMロールを作成します LiveAnalytics アクセスに Timestream を提供する。
2. Lambda サービスを追加するには、IAMロールの信頼関係を編集します。以下のコマンドを使用して既存のロールを更新し、 AWS Lambda が引き受けられるようにできます。

    a. 信頼ポリシードキュメントを作成します。

```
cat > Lambda-Role-Trust-Policy.json << EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

```
EOF
```

b. 信頼ドキュメントを使用して前のステップからロールを更新する

```
aws iam update-assume-role-policy --role-name <name_of_the_role_from_step_1> --
policy-document file://Lambda-Role-Trust-Policy.json
```

関連するリファレンスは TimestreamWriteおよび にありますTimestreamQuery。

LiveAnalytics で の Amazon Timestream を使用して AWS Lambda 関数を構築する JavaScript

LiveAnalytics で の Amazon Timestream を使用して AWS Lambda 関数を構築するには JavaScript、ここで説明されている手順に従ってください。

関連するリファレンスは、v3 の場合は Timestream Write Client - AWS SDK、 JavaScript v JavaScript 3 の場合は Timestream Query Client - AWS SDK にあります。

Amazon Timestream for LiveAnalytics with Go を使用して AWS Lambda 関数を構築する

Amazon Timestream for LiveAnalytics with Go を使用して AWS Lambda 関数を構築するには、https://docs.aws.amazon.com/lambda/latest/dg/golang-package.html「」で説明されている手順に従ってください。

関連する参照は、timestreamwrite および timestreamquery です。

C# LiveAnalytics で の Amazon Timestream を使用して AWS Lambda 関数を構築する

C# LiveAnalytics で の Amazon Timestream を使用して AWS Lambda 関数を構築するには、ここで説明されている手順に従ってください。

関連するリファレンスは、Amazon TimestreamWriteおよび Amazon TimestreamQueryにあります。

AWS IoT Core

AWS IoT Core を使用して IoT デバイスからデータを収集し、IoT Core ルールアクションを通じて Amazon Timestream IoT にデータをルーティングできます。 AWS IoT ルールアクションは、ルールがトリガーされたときに何をすべきかを指定します。Amazon Timestream テーブル、Amazon

DynamoDB データベースにデータを送信し、 AWS Lambda 関数を呼び出すアクションを定義できます。

IoT ルールの Timestream アクションは、受信メッセージから Timestream に直接データを挿入するために使用されます。このアクションは、IoT Core SQL ステートメントの結果を解析し、データを Timestream に保存します。返されたSQL結果セットのフィールドの名前は measure::name として使用され、フィールドの値は measure::value です。

例えば、 SQLステートメントとサンプルメッセージペイロードを考えてみましょう。

```
SELECT temperature, humidity from 'iot/topic'
```

```
{
  "dataFormat": 5,
  "rssi": -88,
  "temperature": 24.04,
  "humidity": 43.605,
  "pressure": 101082,
  "accelerationX": 40,
  "accelerationY": -20,
  "accelerationZ": 1016,
  "battery": 3007,
  "txPower": 4,
  "movementCounter": 219,
  "device_id": 46216,
  "device_firmware_sku": 46216
}
```

上記のSQLステートメントを使用して Timestream の IoT Core ルールアクションが作成された場合、2 つのレコードが Timestream に追加され、測定名の温度と湿度、および測定値はそれぞれ 24.04 と 43.605 になります。

SELECT ステートメントの AS 演算子を使用して、Timestream に追加するレコードのメジャー名を変更できます。以下のSQLステートメントは、温度ではなくメッセージ名の温度でレコードを作成します。

メジャーのデータ型は、メッセージペイロードの値のデータ型から推測されます。JSON 整数、二重、ブール、文字列などのデータ型は、VARCHARそれぞれ BIGINT、、DOUBLEBOOLEAN、および の Timestream データ型にマッピングされます。cast() 関数を使用して、特定のデータ型にデータを強制することもできます。メジャーのタイムスタンプを指定できます。タイムスタンプを空白のままにすると、エントリが処理された時刻が使用されます。

詳細については、Timestream ルールアクションドキュメントを参照してください。

Timestream にデータを保存する IoT Core ルールアクションを作成するには、以下の手順に従います。

トピック

前提条件

1. で説明されている手順に従って、Amazon Timestream にデータベースを作成します データベースを作成する。
2. で説明されている手順に従って、Amazon Timestream でテーブルを作成しますテーブルを作成する。

コンソールを使用する場合

1. AWS Management Console for AWS IoT Core を使用して、Manage > Message routing > Rules をクリックしてルールを作成し、その後にルールを作成します。
2. ルール名を任意の名前に設定し、 SQLを以下に示すテキストに設定します。

```
SELECT temperature as temp, humidity from 'iot/topic'
```

3. アクションリストから Timestream を選択する
4. Timestream データベース、テーブル、ディメンション名と、Timestream にデータを書き込むロールを指定します。ロールが存在しない場合は、ロールの作成をクリックして作成できます。
5. ルールをテストするには、ここに記載されている手順に従ってください。

CLI の使用

AWS コマンドラインインターフェイス (AWS CLI) をインストールしていない場合は、ここからインストールします。

1. 次のルールペイロードを timestream_rule.json というJSONファイルに保存します。置換 *arn:aws:iam::123456789012:role/TimestreamRole* Amazon Timestream にデータを保存するための AWS IoT アクセスを許可するロール arn を使用する

```
{
    "actions": [
            {
                "timestream": {
                    "roleArn": "arn:aws:iam::123456789012:role/TimestreamRole",
                    "tableName": "devices_metrics",
                    "dimensions": [
                        {
                            "name": "device_id",
                            "value": "${clientId()}"
                        },
                        {
                            "name": "device_firmware_sku",
                            "value": "My Static Metadata"
                        }
                    ],
                    "databaseName": "record_devices"
                }
            }
    ],
    "sql": "select * from 'iot/topic'",
    "awsIotSqlVersion": "2016-03-23",
    "ruleDisabled": false
}
```

2. 次のコマンドを使用してトピックルールを作成する

```
aws iot create-topic-rule --rule-name timestream_test --topic-rule-payload file://
<path/to/timestream_rule.json> --region us-east-1
```

3. 次のコマンドを使用してトピックルールの詳細を取得する

```
aws iot get-topic-rule --rule-name timestream_test
```

4. 次のメッセージペイロードを timestream_msg.json というファイルに保存します。

```
{
  "dataFormat": 5,
```

```
  "rssi": -88,
  "temperature": 24.04,
  "humidity": 43.605,
  "pressure": 101082,
  "accelerationX": 40,
  "accelerationY": -20,
  "accelerationZ": 1016,
  "battery": 3007,
  "txPower": 4,
  "movementCounter": 219,
  "device_id": 46216,
  "device_firmware_sku": 46216
}
```

5. 次のコマンドを使用してルールをテストする

```
aws iot-data publish --topic 'iot/topic' --payload file://<path/to/
timestream_msg.json>
```

サンプルアプリケーション

AWS IoT Core で Timestream の使用を開始するのに役立つように、トピックルールを作成するために必要なアーティファクトを AWS IoT Core と Timestream に作成するフル機能のサンプルアプリケーションと、そのトピックにデータを公開するためのサンプルアプリケーションを作成しました。

1. IoT AWS IoT Core 統合のサンプルアプリケーションの GitHub リポジトリを、 の指示に従ってクローンします。 GitHub
2. の手順に従って、 AWS CloudFormation テンプレートREADMEを使用して Amazon Timestream と AWS IoT Core に必要なアーティファクトを作成し、サンプルメッセージをトピックに発行します。

チュートリアル動画

このビデオでは、IoT Core が Timestream と連携する方法について説明します。

Amazon Managed Service for Apache Flink

Apache Flink を使用して、時系列データを Amazon Managed Service for Apache Flink、Amazon MSK、Apache Kafka、およびその他のストリーミングテクノロジーから Amazon Timestream for に

直接転送できます LiveAnalytics。Timestream 用の Apache Flink サンプルデータコネクタを作成しました。また、データを Amazon Kinesis に送信するためのサンプルアプリケーションも作成しました。これにより、データは Kinesis から Managed Service for Apache Flink に流れ、最後に Amazon Timestream に流れます。これらのアーティファクトはすべて、 で入手できます GitHub。このビデオチュートリアルでは、セットアップについて説明します。

> **Note**
>
> Java 11 は、 Managed Service for Apache Flink Application を使用するための推奨バージョンです。複数の Java バージョンがある場合は、Java 11 を JAVA_HOME 環境変数にエクスポートしてください。

トピック

- サンプルアプリケーション
- チュートリアル動画

サンプルアプリケーション

開始するには、以下の手順に従います。

1. 「」で説明されているkdaflink手順に従って、名前で Timestream にデータベースを作成します。 データベースを作成する
2. Timestream で、「」で説明kinesisdata1されている手順に従って、 という名前のテーブルを作成します。 テーブルを作成する
3. 「ストリームの作成」で説明されているTimestreamTestStream手順に従って、名前で Amazon Kinesis Data Stream を作成する
4. の指示に従って、Timestream の Apache Flink データコネクタの GitHub リポジトリをクローンします。 GitHub
5. サンプルアプリケーションをコンパイル、実行、使用するには、Apache Flink サンプルデータコネクタの手順に従います。 README
6. Managed Service for Apache Flink アプリケーションをアプリケーションコードのコンパイル手順に従ってコンパイルする
7. Apache Flink ストリーミングコードをアップロードする手順に従って、Managed Service for Apache Flink アプリケーションバイナリをアップロードする

a. アプリケーションの作成をクリックし、アプリケーションのIAMロールのリンクをクリックします。

b. AmazonKinesisReadOnlyAccess および のIAMポリシーをアタッチしますAmazonTimestreamFullAccess。

> **Note**
>
> 上記のIAMポリシーは特定のリソースに限定されず、本番稼働での使用には適していません。本番稼働システムでは、特定のリソースへのアクセスを制限するポリシーの使用を検討してください。

8. の指示に従って、Kinesis にデータを書き込むサンプルアプリケーションの GitHub リポジトリをクローンします。 GitHub

9. の手順に従ってREADME、Kinesis にデータを書き込むためのサンプルアプリケーションを実行します。

10. Timestream で 1 つ以上のクエリを実行して、指示に従ってデータが Kinesis から Managed Service for Apache Flink から Timestream に送信されていることを確認します。 テーブルを作成する

チュートリアル動画

このビデオでは、Managed Service for Apache Flink で Timestream を使用する方法について説明します。

Amazon Kinesis

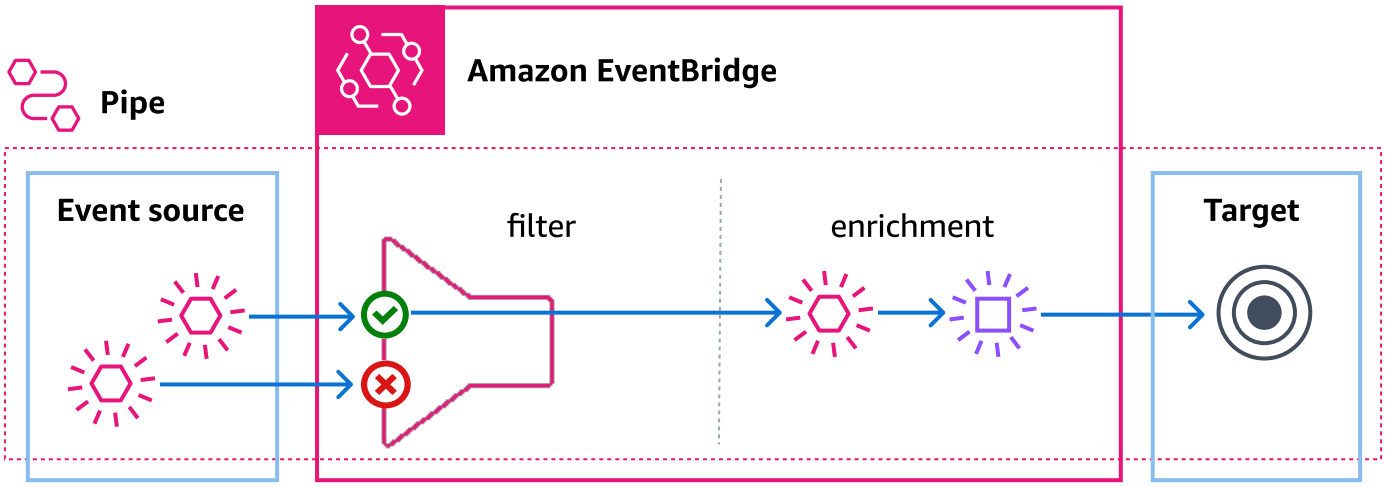
の使用 Amazon Managed Service for Apache Flink

Managed Service for Apache Flink のサンプル Timestream データコネクタ LiveAnalytics を使用して、Kinesis Data Streams から Timestream にデータを送信できます。詳細については、Amazon Managed Service for Apache Flink「Apache Flink」を参照してください。

EventBridge Pipes を使用して Kinesis データを送信する Timestream

EventBridge Pipes を使用して、Kinesis ストリームから LiveAnalytics テーブルの Amazon Timestream にデータを送信できます。

パイプは、サポートされているソースとターゲットの統合を目的として point-to-pointおり、高度な変換とエンリッチメントをサポートしています。Pipes は、イベント駆動型アーキテクチャを開発する際の専門知識と統合コードの必要性を減らします。パイプをセットアップするには、ソースを選択し、オプションのフィルタリングを追加し、オプションのエンリッチメントを定義し、イベントデータのターゲットを選択します。



この統合により、データ取り込みパイプラインを簡素化しながら、 Timestreamの時系列データ分析機能を活用できます。

EventBridge Pipes を で使用する Timestream と、次の利点があります。

- リアルタイムのデータ取り込み: Kinesis から Timestream にデータを直接ストリーミングし LiveAnalytics、リアルタイムの分析とモニタリングを可能にします。
- シームレスな統合: EventBridge Pipes を使用して、複雑なカスタム統合を必要とせずにデータフローを管理します。
- 拡張フィルタリングと変換: Kinesis レコードを に保存する前にフィルタリングまたは変換 Timestream して、特定のデータ処理要件を満たします。
- スケーラビリティ: 高スループットのデータストリームを処理し、組み込みの並列処理とバッチ処理機能を使用して効率的なデータ処理を実現します。

構成

Kinesis から にデータをストリーミングする EventBridge Pipe を設定するには Timestream、次の手順に従います。

1. Kinesis Stream を作成する

データを取り込むアクティブな Kinesis データストリームがあることを確認します。

2. Timestream データベースとテーブルを作成する

   データを保存する Timestream データベースとテーブルを設定します。

3. EventBridge パイプの設定：

   - ソース: Kinesis ストリームをソースとして選択します。
   - ターゲット: ターゲット Timestream として を選択します。
   - バッチ設定: バッチウィンドウとバッチサイズを定義してデータ処理を最適化し、レイテンシーを短縮します。

> **Important**
>
> パイプを設定するときは、いくつかのレコードを取り込んで、すべての設定の正確性をテストすることをお勧めします。パイプが正常に作成されても、パイプラインが正しいことは保証されず、データはエラーなしで流れることに注意してください。マッピングを適用した後、誤ったテーブル、誤った動的パスパラメータ、無効な Timestream レコードなどのランタイムエラーが発生する場合があり、実際のデータがパイプを流れるときに検出されます。

次の設定は、データを取り込む速度を決定します。

- BatchSize: の Timestream に送信するバッチの最大サイズ LiveAnalytics。範囲: 0～100。最大スループットを得るために、この値を 100 にしておくことをお勧めします。
- MaximumBatchingWindowInSeconds: バッチが LiveAnalytics ターゲットの Timestream に送信 batchSize されるまでに がいっぱいになるまでの最大待機時間。受信イベントのレートに応じて、この設定によって取り込みの遅延が決まります。この値は < 10 秒のままにして、データを Timestream ほぼリアルタイムで に送信し続けることをお勧めします。
- ParallelizationFactor: 各シャードから同時に処理するバッチの数。最大値 10 を使用して、最大スループットとほぼリアルタイムの取り込みを取得することをお勧めします。

  ストリームが複数のターゲットによって読み取られる場合は、拡張ファンアウトを使用してパイプに専用コンシューマーを提供し、高いスループットを実現します。詳細については、Kinesis Data Streams 「 ユーザーガイド」の「 を使用した拡張ファンアウトコンシューマーの開発 Kinesis Data Streams API」を参照してください。

> **Note**
>
> 達成できる最大スループットは、アカウントあたりの同時パイプ実行によって制限されます。

次の設定により、データ損失を防止できます。

- DeadLetterConfig: ユーザーエラー LiveAnalytics が原因でイベント DeadLetterConfig を Timestream に取り込むことができなかった場合のデータ損失を避けるために、常に を設定することをお勧めします。

次の設定でパイプのパフォーマンスを最適化します。これにより、レコードの速度低下やブロックを防ぐことができます。

- MaximumRecordAgeInSeconds: これより古いレコードは処理されず、 に直接移動されますDLQ。この値は、ターゲット Timestream テーブルの設定済みメモリストアの保持期間を超えないように設定することをお勧めします。
- MaximumRetryAttempts: レコードが に送信されるまでのレコードの再試行回数 DeadLetterQueue。これを 10 に設定することをお勧めします。これにより、一時的な問題に対処でき、永続的な問題の場合、レコードは に移動 DeadLetterQueue され、残りのストリームのブロックが解除されます。
- OnPartialBatchItemFailure: 部分バッチ処理をサポートするソースの場合、これを有効にし、 にドロップ/送信する前に、失敗したレコードをさらに再試行するために AUTOMATIC_BISECT として設定することをお勧めしますDLQ。

設定例

以下は、Kinesis ストリームから Timestream テーブルにデータをストリーミングするように EventBridge Pipe を設定する方法の例です。

Example IAM のポリシー更新 Timestream

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "timestream:WriteRecords"
            ],
            "Resource": [
                "arn:aws:timestream:us-east-1:123456789012:database/my-database/table/
my-table"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "timestream:DescribeEndpoints"
            ],
            "Resource": "*"
        }
    ]
}
```

Example Kinesis ストリーム設定

```
{
  "Source": "arn:aws:kinesis:us-east-1:123456789012:stream/my-kinesis-stream",
  "SourceParameters": {
    "KinesisStreamParameters": {
        "BatchSize": 100,
        "DeadLetterConfig": {
            "Arn": "arn:aws:sqs:us-east-1:123456789012:my-sqs-queue"
        },
       "MaximumBatchingWindowInSeconds": 5,
        "MaximumRecordAgeInSeconds": 1800,
        "MaximumRetryAttempts": 10,
        "StartingPosition": "LATEST",
       "OnPartialBatchItemFailure": "AUTOMATIC_BISECT"
    }
  }
}
```

Example Timestream ターゲット設定

```
{
    "Target": "arn:aws:timestream:us-east-1:123456789012:database/my-database/table/my-
table",
    "TargetParameters": {
```

```
        "TimestreamParameters": {
            "DimensionMappings": [
                {
                    "DimensionName": "sensor_id",
                    "DimensionValue": "$.data.device_id",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "DimensionName": "sensor_type",
                    "DimensionValue": "$.data.sensor_type",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "DimensionName": "sensor_location",
                    "DimensionValue": "$.data.sensor_loc",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": [
                {
                    "MultiMeasureName": "readings",
                    "MultiMeasureAttributeMappings": [
                        {
                            "MultiMeasureAttributeName": "temperature",
                            "MeasureValue": "$.data.temperature",
                            "MeasureValueType": "DOUBLE"
                        },
                        {
                            "MultiMeasureAttributeName": "humidity",
                            "MeasureValue": "$.data.humidity",
                            "MeasureValueType": "DOUBLE"
                        },
                        {
                            "MultiMeasureAttributeName": "pressure",
                            "MeasureValue": "$.data.pressure",
                            "MeasureValueType": "DOUBLE"
                        }
                    ]
                }
            ],
            "SingleMeasureMappings": [],
            "TimeFieldType": "TIMESTAMP_FORMAT",
            "TimestampFormat": "yyyy-MM-dd HH:mm:ss.SSS",
            "TimeValue": "$.data.time",
```

```
            "VersionValue": "$.approximateArrivalTimestamp"
        }
    }
}
```

イベント変換

EventBridge パイプを使用すると、 に達する前にデータを変換できます Timestream。変換ルールを定義して、フィールド名の変更など、受信 Kinesis レコードを変更できます。

Kinesis ストリームに温度と湿度のデータが含まれているとします。 EventBridge 変換を使用して、これらのフィールドを に挿入する前に名前を変更できます Timestream。

ベストプラクティス

バッチ処理とバッファリング

- 書き込みレイテンシーと処理効率のバランスを取るように、バッチ処理ウィンドウとサイズを設定します。
- バッチ処理ウィンドウを使用して処理前に十分なデータを蓄積し、頻繁な小さなバッチのオーバーヘッドを減らします。

並列処理

特に高スループットストリームでは、 ParallelizationFactor設定を使用して同時実行数を増やします。これにより、各シャードの複数のバッチを同時に処理できます。

データ変換

EventBridge Pipes の変換機能を活用して、 に保存する前にレコードをフィルタリングして強化します Timestream。これは、データを分析要件に合わせるのに役立ちます。

セキュリティ

- EventBridge Pipes に使用されるIAMロールに、 との間で読み Kinesis 書きするために必要なアクセス許可があることを確認します Timestream。
- 暗号化とアクセスコントロールの手段を使用して、転送中および保管中のデータを保護します。

デバッグ失敗

- パイプの自動無効化

  ターゲットが存在しない場合、またはアクセス許可に問題がある場合、パイプは約 2 時間後に自動的に無効になります。
- Throttles

  パイプには、スロットルが減少するまで自動的にバックオフして再試行する機能があります。
- ログの有効化

  ERROR 失敗したログを有効にし、実行データを含めることで、失敗したログに関するより多くのインサイトを取得することをお勧めします。障害が発生すると、これらのログには request/response sent/receivedからの が含まれます Timestream。これにより、関連するエラーを理解し、必要に応じて修正後にレコードを再処理できます。

モニタリング

データフローの問題を検出するために、次のアラームを設定することをお勧めします。

- ソース内のレコードの最大経過時間
  - `GetRecords.IteratorAgeMilliseconds`
- Pipes の障害メトリクス
  - `ExecutionFailed`
  - `TargetStageFailed`
- Timestream 書き込みAPIエラー
  - `UserErrors`

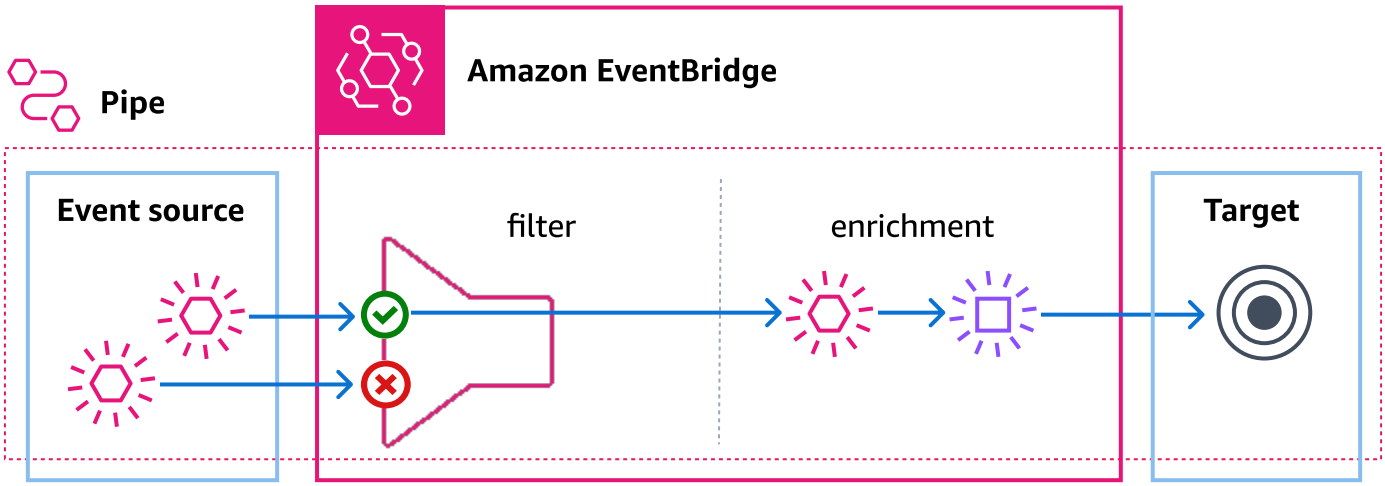
その他のモニタリングメトリクスについては、「 ユーザーガイド」の「モニタリング EventBridge」を参照してください。 EventBridge

Amazon MQ

EventBridge Pipes を使用して Amazon MQ データを に送信する Timestream

EventBridge Pipes を使用して、Amazon MQ ブローカーから LiveAnalytics テーブルの Amazon Timestream にデータを送信できます。

パイプは、サポートされているソースとターゲットの統合を目的として point-to-pointおり、高度な変換とエンリッチメントをサポートしています。Pipes は、イベント駆動型アーキテクチャを開発する際の専門知識と統合コードの必要性を減らします。パイプをセットアップするには、ソースを選択し、オプションのフィルタリングを追加し、オプションのエンリッチメントを定義し、イベントデータのターゲットを選択します。



EventBridge パイプの詳細については、EventBridge 「 ユーザーガイド」のEventBridge 「パイプ」を参照してください。 LiveAnalytics テーブルの Amazon Timestream にイベントを配信するようにパイプを設定する方法については、EventBridge 「Pipes ターゲットの詳細」を参照してください。

Amazon MSK

Managed Service for Apache Flink を使用して の Timestream に Amazon MSK データを送信する LiveAnalytics

Managed Service for Apache Flink のサンプルデータコネクタのような Timestream データコネクタを構築 Timestream することで、 から Amazon MSK にデータを送信できます。詳細については、「Amazon Managed Service for Apache Flink」を参照してください。

Kafka Connect を使用して の Timestream に Amazon MSK データを送信する LiveAnalytics

Kafka Connect を使用して、時系列データを の Timestream Amazon MSK に直接取り込むことができます LiveAnalytics。

用の Kafka シンクコネクタのサンプルを作成しました Timestream。また、データを Kafka トピックに公開するためのサンプル Apache jMeter テストプランも作成しました。これにより、データは

Timestream Kafka シンクコネクタを介してトピックからテーブルの LiveAnalytics Timestream に流れることができます。これらのアーティファクトはすべて、 で入手できます GitHub。

> **Note**
>
> Java 11 は、Timestream Kafka シンクコネクタを使用するための推奨バージョンです。複数の Java バージョンがある場合は、Java 11 を JAVA_HOME 環境変数にエクスポートしてください。

サンプルアプリケーションの作成

開始するには、以下の手順に従ってください。

1. の Timestream で LiveAnalytics、 という名前のデータベースを作成しますkafkastream。

   詳細な手順???については、「」の手順を参照してください。
2. の Timestream で LiveAnalytics、 という名前のテーブルを作成しますpurchase_history。

   詳細な手順???については、「」の手順を参照してください。
3. で共有されている手順に従って、、、および を作成します。

   - Amazon MSK クラスター
   - Kafka プロデューサークライアントマシンとして設定されている Amazon EC2 インスタンス
   - Kafka トピック

   詳細な手順については、kafka_ingestor プロジェクトの前提条件を参照してください。
4. Timestream Kafka Sink Connector リポジトリをクローンします。

   詳細な手順 GitHub については、「 のリポジトリのクローン作成」を参照してください。
5. プラグインコードをコンパイルします。

   詳細な手順 GitHub については、「 コネクタ - ソースから構築する」を参照してください。
6. S3 バケットに次のファイルをアップロードします。「」の手順に従います。

   - `/target` ディレクトリからの jar ファイル (kafka-connector-timestream->VERSION<-jar-with-dependencies.jar)
   - サンプル json スキーマファイル `purchase_history.json`。

詳細な手順については、Amazon S3 「 ユーザーガイド」の「オブジェクトのアップロード」を参照してください。

7. 2 つのVPCエンドポイントを作成します。これらのエンドポイントは、 を使用してリソースにアクセスするために MSK Connector によって使用されます AWS PrivateLink。

   - Amazon S3 バケットにアクセスする 1 つ
   - LiveAnalytics テーブルの Timestream にアクセスする 1 つ。

   詳細な手順については、VPC「エンドポイント」を参照してください。

8. アップロードされた jar ファイルを使用してカスタムプラグインを作成します。

   詳細な手順については、Amazon MSK 「 デベロッパーガイド」の「プラグイン」を参照してください。

9. 「ワーカー設定パラメータ」で説明されているJSON内容を使用して、カスタムワーカー設定を作成します。「」で説明されている手順に従います。 https://github.com/awslabs/amazon-timestream-tools/tree/mainline/integrations/kafka_connector#worker-configuration-parameters

   詳細については、「 Amazon MSK デベロッパーガイド」の「カスタムワーカー設定の作成」を参照してください。

10. サービス実行 IAM ロールを作成します。

    詳細な手順については、IAM 「サービスロール」を参照してください。

11. 前のステップで作成したカスタムプラグイン、カスタムワーカー設定、サービス実行 IAM ロール、およびサンプル Amazon MSK コネクタ設定 を使用してコネクタを作成します。 https://github.com/awslabs/amazon-timestream-tools/tree/mainline/integrations/kafka_connector#sample-connector-configuration

    詳細については、「 Amazon MSK デベロッパーガイド」の「コネクタの作成」を参照してください。

    以下の設定パラメータの値は、必ずそれぞれの値で更新してください。詳細については、「コネクタ設定パラメータ」を参照してください。

    - `aws.region`
    - `timestream.schema.s3.bucket.name`
    - `timestream.ingestion.endpoint`

コネクタの作成が完了するまでに 5～10 分かかります。パイプラインのステータスが に変わると、パイプラインの準備が整いますRunning。

12. 作成された Kafka トピックにデータを書き込むためのメッセージの継続的なストリームを発行します。

    詳細については、「使用方法」を参照してください。

13. 1 つ以上のクエリを実行して、データが から LiveAnalytics テーブルの Timestream MSK Amazon MSK に接続に送信されることを確認します。

    詳細な手順???については、「」の手順を参照してください。

追加リソース

このブログでは、Kafka Connect LiveAnalytics を使用して Kafka クラスターから Timestream へのリアルタイムサーバーレスデータ取り込みについて説明します。これは、Apache jMeter テストプランを使用して Kafka トピックに数千のサンプルメッセージをパブリッシュし、 LiveAnalytics テーブルの Timestream で取り込まれたレコードを検証する Kafka プロデューサークライアントマシンから始まる、 LiveAnalytics Kafka Sink Connector の Timestream を使用してパイプラインをセットアップend-to-endする方法について説明します。

Amazon QuickSight

Amazon を使用して QuickSight 、Amazon Timestream データを含むデータダッシュボードを分析および公開できます。このセクションでは、新しい QuickSight データソース接続の作成、アクセス許可の変更、新しいデータセットの作成、分析の実行方法について説明します。このビデオチュートリアルでは、Timestream と Amazon の操作方法について説明します QuickSight。

> **Note**
>
> Amazon のすべてのデータセット QuickSight は読み取り専用です。Amazon を使用してデータソース、データセット、またはフィールド QuickSight を削除することで、Timestream の実際のデータを変更することはできません。

トピック

- から Amazon Timestream にアクセスする QuickSight

- Timestream の新しい QuickSight データソース接続を作成する
- Timestream の QuickSight データソース接続のアクセス許可を編集する
- Timestream の新しい QuickSight データセットを作成する
- Timestream の新しい分析を作成する
- チュートリアル動画

から Amazon Timestream にアクセスする QuickSight

続行する前に、Amazon Timestream への接続を Amazon に許可 QuickSight する必要があります。接続が有効になっていない場合、接続しようとするとエラーが表示されます。 QuickSight 管理者は、 AWS リソースへの接続を承認できます。から QuickSight Timestream への接続を承認するには、「その他の AWS サービスの使用: アクセスの停止」の手順に従い、ステップ 5 で Amazon Timestream を選択します。

Timestream の新しい QuickSight データソース接続を作成する

> **Note**
>
> Amazon QuickSight と Amazon Timestream 間の接続はSSL、 (TLS 1.2) を使用して転送中に暗号化されます。暗号化されていない接続を作成することはできません。

1. の説明に従って、Amazon QuickSight が Amazon Timestream にアクセスするための適切なアクセス許可が設定されていることを確認します から Amazon Timestream にアクセスする QuickSight。
2. まず、新しいデータセットを作成します。ナビゲーションペインからデータセットを選択し、新しいデータセット を選択します。
3. Timestream データソースカードを選択します。
4. データソース名 には、Timestream データソース接続の名前を入力します。例えば、 ですUS `Timestream Data`。

   > **Note**
   >
   > Timestream への接続から多数のデータセットを作成できるようにするため、名前はシンプルにしておくことをお勧めします。

5. [Validate connection] (接続を検証) を選択し、Timestream に正常に接続できていることを確認します。

> **Note**
>
> 接続の検証は、接続できるという検証のみを行います。ただし、特定のテーブルやクエリは検証されません。

6. [Create data source] (データソースを作成) を選択し、先へ進みます。
7. データベース では、選択... を選択して、使用可能なオプションのリストを表示します。使用するものを選択します。
8. 選択を選択して続行します。
9. 以下のうちのひとつを選択します。
   - のイン QuickSightメモリエンジン (と呼ばれるSPICE) にデータをインポートするには、 にインポートを選択して分析をSPICE迅速化します。
   - QuickSight がデータセットを更新するか、分析またはダッシュボードを使用するたびにデータに対してクエリを実行するようにするには、データ を直接クエリします。
10. [Edit/Preview] (編集/プレビュー) を選択し、[Save] (保存) を選択してデータセットを保存し、閉じます。

Timestream の QuickSight データソース接続のアクセス許可を編集する

次の手順では、他のユーザーが同じ Timestream データソースにアクセスできるように、 QuickSight 他のユーザーのアクセス許可を表示、追加、および取り消す方法について説明します。ユーザーは、追加 QuickSight する前に でアクティブユーザーである必要があります。

> **Note**
>
> では QuickSight、データソースにはユーザーと所有者の 2 つのアクセス許可レベルがあります。
>
> - 読み取りアクセスを許可するときは、[user] (ユーザー) を選択します。
> - 所有者を選択すると、そのユーザーがこの QuickSight データソースを編集、共有、または削除できるようになります。

1. の説明に従って、Amazon QuickSight が Amazon Timestream にアクセスするための適切なアクセス許可が設定されていることを確認します から Amazon Timestream にアクセスする QuickSight。
2. 左側で [Datasets] (データセット) を選択し、下にスクロールして Timestream 接続のデータソースカードを見つけます。例えば、 US Timestream Dataです。
3. Timestream データソースカードを選択します。
4. [Share data source] を選択します。現在のアクセス許可のリストが表示されます。
5. (オプション) アクセス許可を編集するには、 userまたは を選択しますowner。
6. (オプション) アクセス許可を取り消すには、 を選択しますRevoke access。取り消したユーザーは、このデータソースから新しいデータセットを作成することはできません。ただし、その人の既存のデータセットからは、引き続きこのデータソースにアクセスすることができます。
7. アクセス許可を追加するには、 を選択しInvite users、次のステップに従ってユーザーを追加します。

   a. 同じデータソースの使用を許可するユーザーを追加します。

   b. それぞれに適用する を選択しますPermission。
8. 完了したら、[Close] を選択します。

Timestream の新しい QuickSight データセットを作成する

1. の説明に従って、Amazon QuickSight が Amazon Timestream にアクセスするための適切なアクセス許可が設定されていることを確認します から Amazon Timestream にアクセスする QuickSight。
2. 左側で [Datasets] (データセット) を選択し、下にスクロールして Timestream 接続のデータソースカードを見つけます。データソースが多数ある場合は、ページ上部の検索バーを使用して、名前に部分一致で検索できます。
3. [Timestream] のデータソースカードを選択します。次に、データセットの作成 を選択します。
4. [Database] (データベース) で、[Select] (選択) を選択し、使用可能なオプションの一覧を表示します。使用するデータベースを選択します。
5. [Tables] (テーブル) で、使用するテーブルを選択します。
6. [Edit/Preview] (編集/プレビュー) を選択します。
7. (オプション) データを追加するには、右上にデータを追加を選択します。

   a. Switch data source を選択し、別のデータソースを選択します。

b. UI のプロンプトに従ってデータの追加を完成させます。

c. 新しいデータを同じデータセットに追加したら、[Configure this join] (この結合を設定します) (2 つの赤いドット) を選択します。追加した各テーブルで結合をセットアップします。

d. 計算フィールドを追加するときは、[Add calculated field] (計算フィールドを追加) を選択します。

e. Sagemaker を使用するには、 で Augment SageMakerを選択します。このオプションは QuickSight Enterprise Edition でのみ使用できます。

f. 省略するフィールドのチェックを外します。

g. 変更するデータ型を更新します。

8. 完了したら、[Save] (保存) を選択し、データセットを保存して閉じます。

Timestream の新しい分析を作成する

1. の説明に従って、Amazon QuickSight が Amazon Timestream にアクセスするための適切なアクセス許可が設定されていることを確認します から Amazon Timestream にアクセスする QuickSight。
2. 左側で [Analyses] (分析) を選択します。
3. 以下のうちのひとつを選択します。
   - 新しい分析を作成するときは、右側で [New analysis] (新しい分析) を選択します。
   - Timestream データセットを既存の分析に追加するには、編集する分析を開きます。左上に近い鉛筆アイコンを選択し、データセット を追加します。
4. 左側のフィールドを選択して、最初のデータ視覚化を開始します。
5. 詳細については、「分析の使用 - Amazon QuickSight 」を参照してください。

チュートリアル動画

このビデオでは、Amazon が Timestream と QuickSight 連携する方法について説明します。

Amazon SageMaker

Amazon SageMaker Notebooks を使用して、機械学習モデルを Amazon Timestream と統合できます。開始しやすくするために、Timestream のデータを処理するサンプル SageMaker ノートブックを作成しました。データは、マルチスレッドの Python アプリケーションから Timestream に挿入

され、継続的にデータを送信します。サンプル SageMaker ノートブックとサンプル Python アプリケーションのソースコードは、 で入手できます GitHub。

1. および で説明されている手順に従って、データベース データベースを作成するとテーブルを作成します。 テーブルを作成する
2. からの手順に従って、マルチスレッド Python サンプルアプリケーションの GitHub リポジトリをクローンします。 GitHub
3. の指示に従って、サンプル Timestream SageMaker Notebook の GitHub リポジトリをクローンします GitHub。
4. 「」の指示に従って、Timestream にデータを継続的に取り込むアプリケーションを実行します。 README
5. 「」の説明 SageMaker に従って、Amazon 用の Amazon S3 バケットを作成します。 https://docs.aws.amazon.com/sagemaker/latest/dg/gs-config-permissions.html
6. 最新の boto3 がインストールされた Amazon SageMaker インスタンスを作成します。ここで説明する手順に加えて、以下の手順に従います。

   a. ノートブックインスタンスの作成ページで、追加設定をクリックします。
   b. ライフサイクル設定 - オプションをクリックし、新しいライフサイクル設定を作成するを選択します。
   c. ライフサイクル設定の作成ウィザードボックスで、次の操作を行います。

      i. 設定に必要な名前を入力します。例: `on-start`
      ii. Start Notebook スクリプトで、Github からスクリプトコンテンツをコピーペーストします。
      iii. 貼り付けたスクリプト`PACKAGE=boto3`の を `PACKAGE=scipy`に置き換えます。
7. 設定の作成をクリックします。
8. AWS マネジメントコンソールの IAMサービスに移動し、ノートブックインスタンス用に新しく作成された SageMaker実行ロールを見つけます。
9. のIAMポリシーを実行ロール`AmazonTimestreamFullAccess`にアタッチします。

> **Note**
>
> この`AmazonTimestreamFullAccess`IAMポリシーは特定のリソースに限定されておらず、本番稼働での使用には適していません。本番稼働システムでは、特定のリソースへのアクセスを制限するポリシーの使用を検討してください。

10. ノートブックインスタンスのステータスが の場合InService、Open Jupyter を選択してインスタンスの SageMaker ノートブックを起動します。
11. アップロードボタンを選択して、ノートブックTimestream_SageMaker_Demo.ipynbに ファイルtimestreamquery.pyと をアップロードします。
12. 選択 Timestream_SageMaker_Demo.ipynb

> **Note**
>
> カーネルが見つからないポップアップが表示された場合は、conda_python3 を選択し、カーネルの設定 をクリックします。

13. DB_NAME、TABLE_NAME、bucket、および を変更ENDPOINTして、トレーニングモデルのデータベース名、テーブル名、S3 バケット名、リージョンと一致させます。
14. 個々のセルを実行するには、再生アイコンを選択します。
15. セル に到達するとLeverage Timestream to find hosts with average CPU utilization across the fleet、出力が少なくとも 2 つのホスト名を返すことを確認します。

> **Note**
>
> 出力にホスト名が 2 つ未満の場合は、より多くのスレッドとホストスケールで Timestream にデータを取り込み、サンプル Python アプリケーションを再実行する必要がある場合があります。

16. セル に移動したらTrain a Random Cut Forest (RCF) model using the CPU utilization history、トレーニングジョブのリソース要件train_instance_typeに基づいて を変更します。
17. セル に到達したらDeploy the model for inference、推論ジョブのリソース要件instance_typeに基づいて を変更します。

> **Note**
>
> モデルのトレーニングには数分かかる場合があります。トレーニングが完了すると、セルの出力に完了 - トレーニングジョブが完了したというメッセージが表示されます。

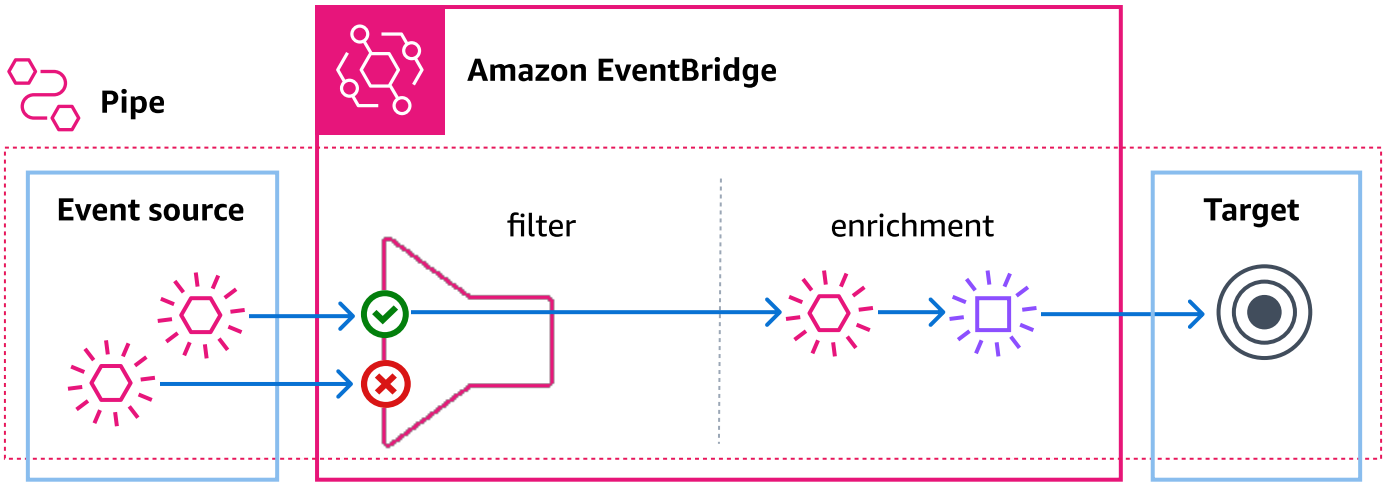
18. セルを実行してリソースStop and delete the endpointをクリーンアップします。SageMaker コンソールからインスタンスを停止または削除することもできます。

Amazon SQS

EventBridge Pipes を使用して Amazon SQS データを送信する Timestream

EventBridge Pipes を使用して、Amazon SQSキューから LiveAnalytics テーブルの Amazon Timestream にデータを送信できます。

パイプは、サポートされているソースとターゲットの統合を目的として point-to-pointおり、高度な変換とエンリッチメントをサポートしています。Pipes は、イベント駆動型アーキテクチャを開発する際の専門知識と統合コードの必要性を減らします。パイプをセットアップするには、ソースを選択し、オプションのフィルタリングを追加し、オプションのエンリッチメントを定義し、イベントデータのターゲットを選択します。



EventBridge パイプの詳細については、EventBridge 「 ユーザーガイド」のEventBridge 「パイプ」を参照してください。 LiveAnalytics テーブルの Amazon Timestream にイベントを配信するようにパイプを設定する方法については、EventBridge 「Pipes ターゲットの詳細」を参照してください。

DBeaver を使用して Amazon Timestream を操作する

DBeaver は、JDBCドライバーを持つデータベースを管理するために使用できる無料のユニバーサルSQLクライアントです。堅牢なデータ表示、編集、管理機能を備えているため、デベロッパーやデータベース管理者の間で広く使用されています。

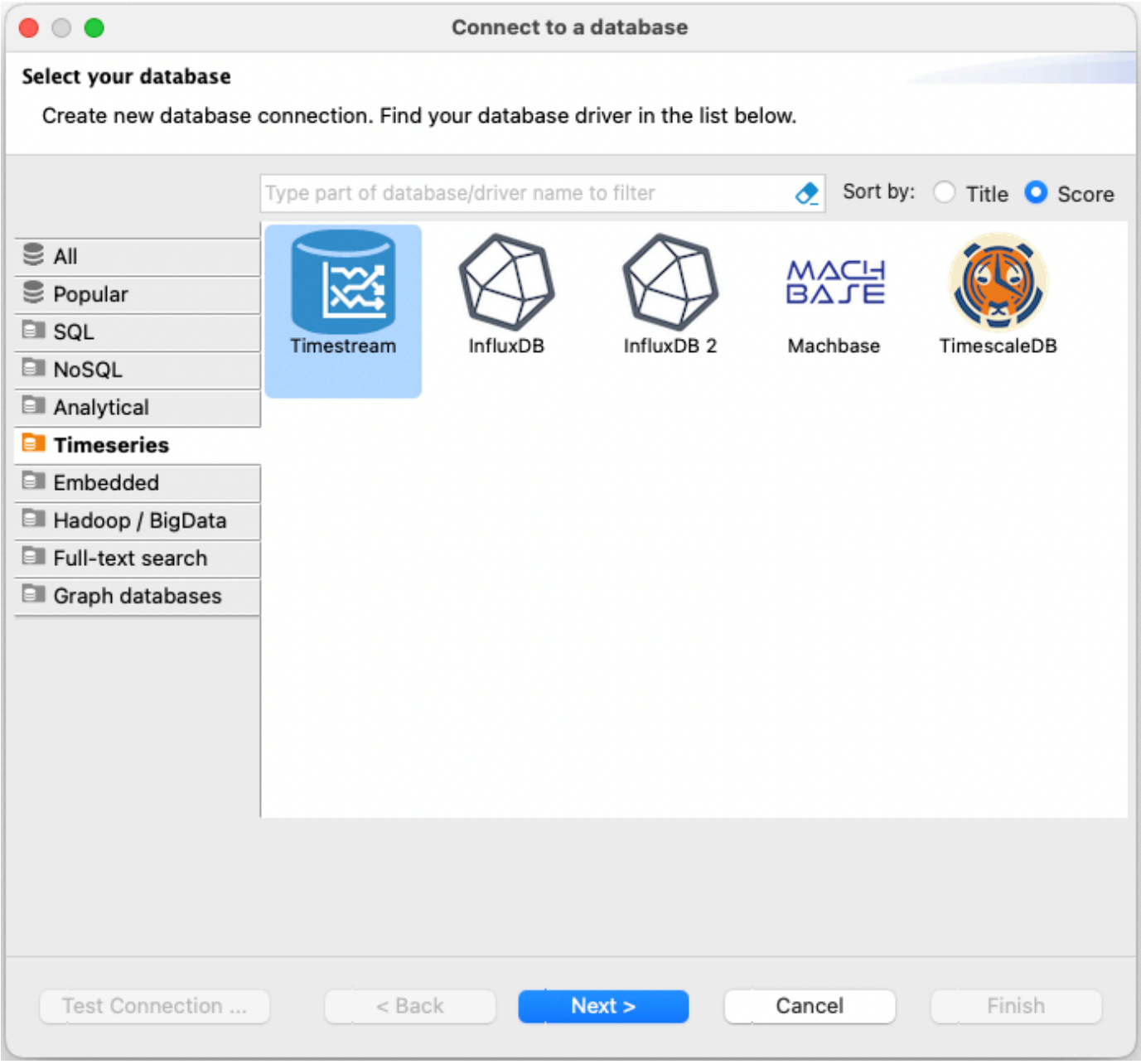
DBeaverのクラウド接続オプションを使用すると、Amazon Timestream DBeaverにネイティブに接続できます。DBeaver は、DBeaverアプリケーション内から直接時系列データを操作するための包括的で直感的なインターフェイスを提供します。認証情報を使用すると、別のクエリインターフェイ

スから実行できるクエリへのフルアクセスも可能になります。さらに、クエリ結果をよりよく理解して視覚化するためのグラフを作成することもできます。

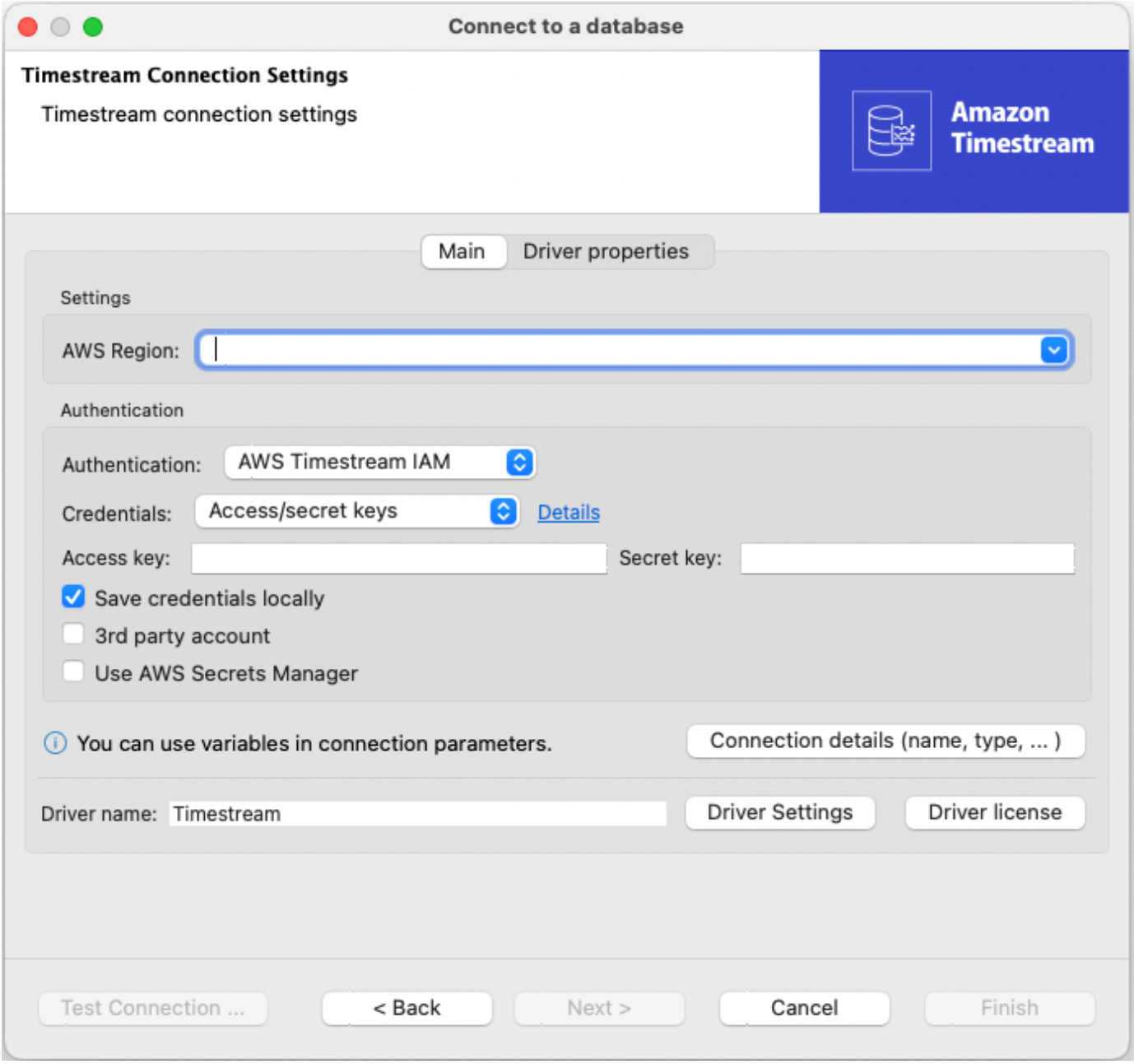
Timestream を操作するDBeaverための の設定

Timestream を使用するDBeaverように を設定するには、次の手順に従います。

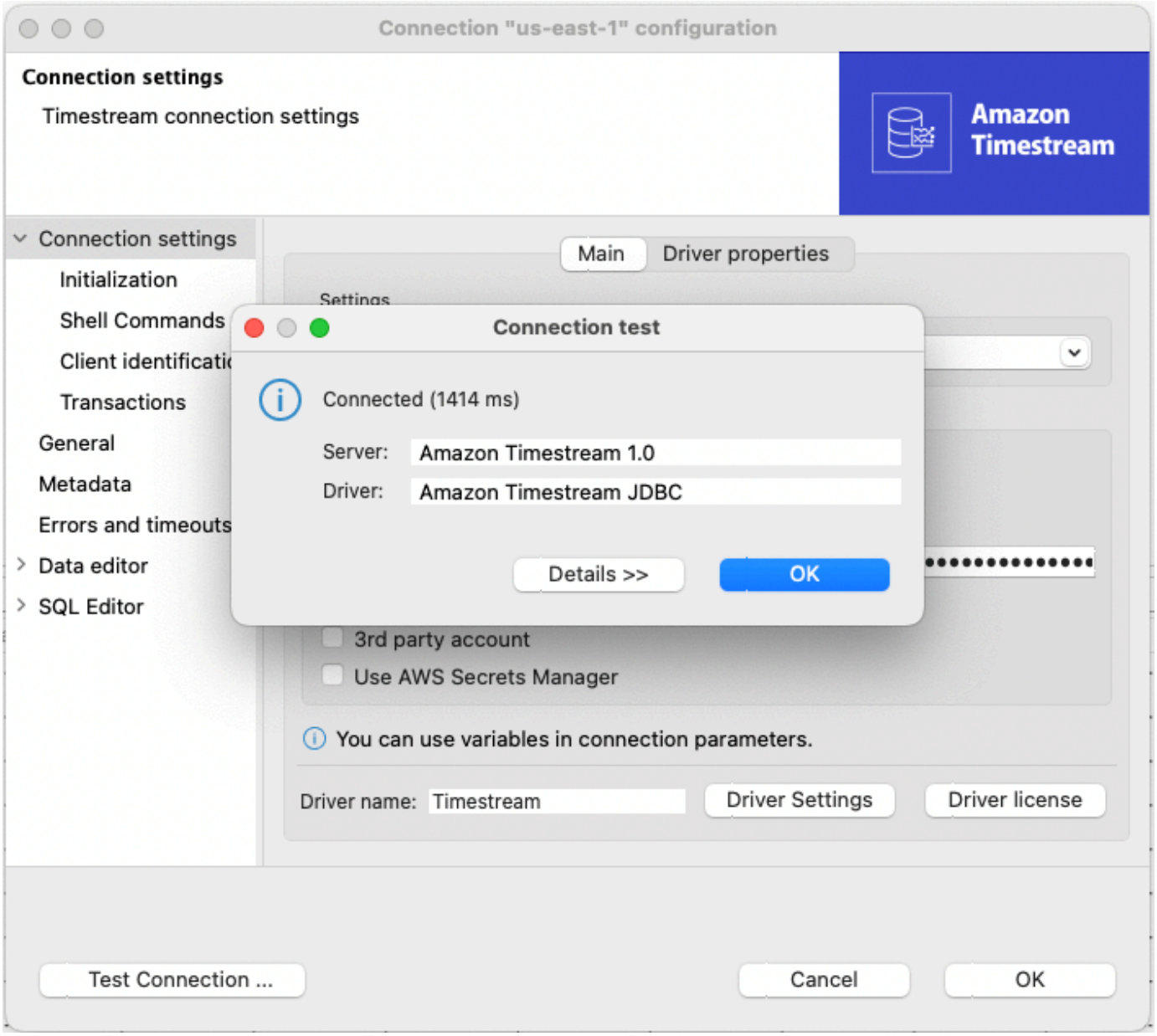
1. ローカルマシンにダウンロードしてインストールDBeaverします。
2. を起動しDBeaver、データベース選択エリアに移動し、左側のペインで時系列を選択し、右側のペインで Timestream アイコンを選択します。



3. Timestream Connection Settings ウィンドウで、Amazon Timestream データベースへの接続に必要なすべての情報を入力します。入力したユーザーキーに Timestream データベースへのアクセスに必要なアクセス許可があることを確認してください。また、入力した情報とキーは、機密情報と同様に、DBeaver安全かつプライベートに保持してください。



4. 接続をテストして、すべてが正しく設定されていることを確認します。



5. 接続テストが成功すると、 の他のデータベースと同様に Amazon Timestream データベースとやり取りできるようになりましたDBeaver。例えば、SQL工ディタまたは ER 図ビューに移動してクエリを実行できます。

6. DBeaver には、強力なデータ視覚化ツールも用意されています。これを使用するには、クエリを実行し、グラフアイコンを選択して結果セットを視覚化します。グラフ作成ツールは、時間の経過に伴うデータの傾向をよりよく理解するのに役立ちます。

Amazon Timestream を とペアリングすると、時系列データを管理するための効果的な環境DBeaverが作成されます。既存のワークフローにシームレスに統合して、生産性と効率を向上させることができます。

Grafana

Grafana を使用して時系列データを視覚化し、アラートを作成できます。データ視覚化の開始に役立つように、Python アプリケーションから Timestream に送信されたデータを視覚化するサンプルダッシュボードを Grafana に作成し、セットアップを説明するビデオチュートリアルを作成しました。

トピック

- サンプルアプリケーション
- チュートリアル動画

サンプルアプリケーション

1. 詳細については、データベースを作成する「」で説明されている手順に従って、Timestream にデータベースとテーブルを作成します。

   > **Note**
   >
   > Grafana ダッシュボードのデフォルトのデータベース名とテーブル名は、 grafanaTable それぞれ grafanaDB と に設定されています。これらの名前を使用して、セットアップを最小限に抑えます。

2. Python 3.7 以降をインストールする
3. Timestream Python のインストールと設定 SDK
4. マルチスレッド Python アプリケーションの GitHub リポジトリをクローンし、 の指示に従って Timestream に継続的にデータを取り込みます。 GitHub
5. 「」の指示に従って、Timestream にデータを継続的に取り込むアプリケーションを実行します。 README
6. Amazon Managed Grafana の開始方法を完了するか、Grafana のインストールを完了します。
7. Amazon Managed Grafana を使用する代わりに Grafana をインストールする場合は、Grafana の Timestream プラグインのインストールを完了します。

8. 任意のブラウザを使用して Grafana ダッシュボードを開きます。Grafana をローカルにインストールしている場合は、Grafana ドキュメントに記載されている手順に従ってログインできます。
9. Grafana を起動したら、データソースに移動し、データソースの追加をクリックし、Timestream を検索して、Timestream データソースを選択します。
10. 認証プロバイダーとリージョンを設定し、保存してテストをクリックします。
11. デフォルトのマクロを設定する
    a. $__database を Timestream データベースの名前に設定する (例: grafanaDB)
    b. Timestream テーブルの名前に $__table を設定する (例: grafanaTable)
    c. $__measure を タブから最もよく使用されるメジャーに設定します。
12. Save and Test をクリックします。
13. Dashboards タブをクリックします。
14. インポートをクリックしてダッシュボードをインポートする
15. サンプルアプリケーションダッシュボードをダブルクリックします。
16. ダッシュボード設定をクリックします。
17. 変数の選択
18. Timestream データベースとテーブルの名前と一致する tableName ように dbName と を変更します。
19. Save をクリックします。
20. ダッシュボードを更新する
21. アラートを作成するには、Grafana ドキュメントに記載されている手順に従って Grafana マネージドアラートルールを作成します。
22. アラートをトラブルシューティングするには、トラブルシューティングの Grafana ドキュメントに記載されている手順に従ってください。
23. 詳細については、Grafana ドキュメントを参照してください。

チュートリアル動画

このビデオでは、Grafana が Timestream と連携する方法について説明します。

SquaredUp を使用して Amazon Timestream を操作する

SquaredUp は、Amazon Timestream と統合するオブザーバビリティプラットフォームです。SquaredUpの直感的なダッシュボードデザイナーを使用して、時系列データを視覚化、分析、モニタリングできます。ダッシュボードはパブリックまたはプライベートに共有でき、モニターのヘルス状態が変わったときに警告する通知チャネルを作成できます。

Amazon Timestream SquaredUp での の使用

1. にサインアップSquaredUpして、無料で開始してください。
2. AWS データソース を追加します。
3. Timestream Query データストリームを使用するダッシュボードタイルを作成します。
4. 必要に応じて、タイルのモニタリングを有効にしたり、通知チャネルを作成したり、ダッシュボードをパブリックまたはプライベートに共有したりできます。
5. 必要に応じて、他のタイルを作成して、他のモニタリングツールやオブザーバビリティツールのデータとともに Timestream データを表示します。

オープンソース Telegraf

Telegraf の LiveAnalytics 出力プラグインに Timestream を使用して、オープンソースの Telegraf から直接 Timestream LiveAnalyticsにメトリクスを書き込むことができます。

このセクションでは、 LiveAnalytics 出力プラグインの Timestream を使用して Telegraf をインストールする方法、 LiveAnalytics 出力プラグインの Timestream を使用して Telegraf を実行する方法、およびオープンソース Telegraf が の Timestream と連携する方法について説明します LiveAnalytics。

トピック

- 出力プラグインの LiveAnalytics Timestream を使用した Telegraf のインストール
- LiveAnalytics 出力プラグインの Timestream で Telegraf を実行する
- Telegraf/InfluxDB メトリクスをモデルの Timestream にマッピングする LiveAnalytics

出力プラグインの LiveAnalytics Timestream を使用した Telegraf のインストール

バージョン 1.16 以降、 LiveAnalytics 出力プラグインの Timestream は公式 Telegraf リリースで利用できます。出力プラグインをほとんどの主要なオペレーティングシステムにインストールするに

は、InfluxData 「Telegraf ドキュメント」で説明されているステップに従います。Amazon Linux 2 OS に をインストールするには、以下の手順に従ってください。

Amazon Linux 2 での LiveAnalytics 出力プラグインの Timestream を使用した Telegraf のインストール

Amazon Linux 2 に Timestream Output Plugin を使用して Telegraf をインストールするには、次の手順を実行します。

1. yum パッケージマネージャーを使用して Telegraf をインストールします。

```
cat <<EOF | sudo tee /etc/yum.repos.d/influxdb.repo
[influxdb]
name = InfluxDB Repository - RHEL \$releasever
baseurl = https://repos.influxdata.com/rhel/\$releasever/\$basearch/stable
enabled = 1
gpgcheck = 1
gpgkey = https://repos.influxdata.com/influxdb.key
EOF
```

2. 以下のコマンドを実行します。

```
sudo sed -i "s/\$releasever/$(rpm -E %{rhel})/g" /etc/yum.repos.d/influxdb.repo
```

3. Telegraf をインストールして起動します。

```
sudo yum install telegraf
sudo service telegraf start
```

LiveAnalytics 出力プラグインの Timestream で Telegraf を実行する

LiveAnalytics プラグインの Timestream を使用して Telegraf を実行するには、以下の手順に従ってください。

1. Telegraf を使用して設定例を生成します。

```
telegraf --section-filter agent:inputs:outputs --input-filter cpu:mem --output-
filter timestream config > example.config
```

2. 管理コンソール 、、または を使用して Timestream にデータベースを作成しますSDKs。 CLI

3. example.config ファイルで、[[outputs.timestream]] セクションで次のキーを編集して、データベース名を追加します。

```
database_name = "yourDatabaseNameHere"
```

4. デフォルトでは、Telegraf はテーブルを作成します。テーブルを手動で作成する場合は、create_table_if_not_exists を に設定falseし、管理コンソール 、、または を使用してテーブルを作成する手順に従ってくださいSDKs。 CLI

5. example.config ファイルで、[[outputs.timestream]] セクションで認証情報を設定します。認証情報では、次の操作を許可する必要があります。

```
timestream:DescribeEndpoints
timestream:WriteRecords
```

> **Note**
>
> create_table_if_not_exists を に設定したままにする場合はtrue、以下を含めます。
>
> ```
> timestream:CreateTable
> ```

> **Note**
>
> describe_database_on_start を に設定する場合はtrue、以下を含めます。
>
> ```
> timestream:DescribeDatabase
> ```

6. 設定に応じて、残りの設定を編集できます。

7. 設定ファイルの編集が完了したら、以下を使用して Telegraf を実行します。

```
./telegraf --config example.config
```

8. メトリクスは、エージェントの設定に応じて数秒以内に表示されます。Timestream コンソールに新しいテーブル cpu と mem も表示されます。

Telegraf/InfluxDB メトリクスをモデルの Timestream にマッピングする LiveAnalytics

の Telegraf から Timestream にデータを書き込む場合 LiveAnalytics、データは次のようにマッピングされます。

- タイムスタンプは時刻フィールドとして記述されます。
- タグはディメンションとして記述されます。
- フィールドはメジャーとして記述されます。
- 測定値は主にテーブル名として記述されます (以下を参照) 。

Telegraf の LiveAnalytics 出力プラグインの Timestream には、 の Timestream にデータを整理して保存するための複数のオプションが用意されています LiveAnalytics。これは、行プロトコル形式のデータで始まる例で説明できます。

```
weather,location=us-midwest,season=summer temperature=82,humidity=71
1465839830100400200 airquality,location=us-west no2=5,pm25=16
1465839830100400200
```

以下に、データについて説明します。

- 測定名は `weather`と ですairquality。
- タグは `location`と ですseason。
- フィールドは、`temperature`、`humidity`、`no2`、および ですpm25。

トピック

- データを複数のテーブルに保存する
- データを 1 つのテーブルに保存する

データを複数のテーブルに保存する

測定ごとに個別のテーブルを作成し、各フィールドをテーブルごとに個別の行に保存することを選択できます。

設定は ですmapping_mode = "multi-table"。

- LiveAnalytics アダプターの Timestream は、 `weather`と の 2 つのテーブルを作成しますairquality。

- 各テーブル行には 1 つのフィールドのみが含まれます。

LiveAnalytics テーブル`weather`と の結果の Timestream `airquality`は、次のように表示されます。

`weather`

時系	ロケーション	season	measure_name	measure_value::bigint
2016-06-13 17:43:50	us-midwest	夏	temperature	82
2016-06-13 17:43:50	us-midwest	夏	湿度	71

`airquality`

時系	ロケーション	measure_name	measure_value::bigint
2016-06-13 17:43:50	us-midwest	no2	5
2016-06-13 17:43:50	us-midwest	pm25	16

データを 1 つのテーブルに保存する

すべての測定値を 1 つのテーブルに保存し、各フィールドを別のテーブル行に保存することを選択できます。

設定は `mapping_mode = "single-table"`。、`single_table_name`、および を使用する場合`single-table`、2 つの追加設定がありますsingle_table_dimension_name_for_telegraf_measurement_name。

- LiveAnalytics 出力プラグインの Timestream は、 という名前のテーブルを 1 つ作成します。*<single_table_name>* これには、*<single_table_dimension_name_for_telegraf_measurement_name>* 列。
- テーブルには、1 つのテーブル行に複数のフィールドを含めることができます。

テーブルの結果の Timestream LiveAnalytics は次のようになります。

weather

時系	ロケーション	season	*<single_table_dimension_name_for_telegraf_measurement_name>*	measure_name	measure_value::bigint
2016-06-13 17:43:50	us-midwest	夏	天気	temperature	82
2016-06-13 17:43:50	us-midwest	夏	天気	湿度	71
2016-06-13 17:43:50	us-midwest	夏	空気品質	no2	5
2016-06-13 17:43:50	us-midwest	夏	天気	pm25	16

JDBC

Java Database Connectivity (JDBC) 接続を使用して、 の Timestream LiveAnalytics をビジネスインテリジェンスツールや SQL Workbench などの他のアプリケーションに接続できます。ドライバーの LiveAnalytics JDBC Timestream は現在、Okta と Microsoft Azure AD SSOで をサポートしています。

トピック

- の Timestream 用JDBCドライバーの設定 LiveAnalytics
- 接続プロパティ
- JDBC URL 例
- Okta によるシングルサインオン認証の LiveAnalytics JDBC Timestream のセットアップ

- Microsoft Azure AD によるシングルサインオン認証の LiveAnalytics JDBC Timestream のセットアップ

の Timestream 用JDBCドライバーの設定 LiveAnalytics

JDBC ドライバーを設定するには、次の手順に従います。

トピック

- ドライバーの LiveAnalytics JDBCタイムストリーム JARs
- ドライバークラスとURL形式のタイムストリーム LiveAnalytics JDBC
- サンプルアプリケーション

ドライバーの LiveAnalytics JDBCタイムストリーム JARs

ドライバーの LiveAnalytics JDBC Timestream は、直接ダウンロードするか、ドライバーを Maven の依存関係として追加することで取得できます。

- 直接ダウンロードとして：JDBC ドライバーの LiveAnalytics Timestream を直接ダウンロードするには、次の手順を実行します。
  1. https://github.com/awslabs/amazon-timestream-driver-jdbc/releases に移動する
  2. ビジネスインテリジェンスツールやアプリケーションと`amazon-timestream-jdbc-1.0.1-shaded.jar`直接使用できます。
  3. 任意のディレクトリ`amazon-timestream-jdbc-1.0.1-javadoc.jar`にダウンロードします。
  4. をダウンロードしたディレクトリで`amazon-timestream-jdbc-1.0.1-javadoc.jar`、次のコマンドを実行して Javadoc HTMLファイルを抽出します。

     ```
     jar -xvf amazon-timestream-jdbc-1.0.1-javadoc.jar
     ```
- Maven 依存関係として： ドライバーの LiveAnalytics JDBC Timestream を Maven 依存関係として追加するには、次の手順を実行します。
  1. 選択した`pom.xml`エディタでアプリケーションのファイルに移動して開きます。
  2. アプリケーションの`pom.xml`ファイルに依存関係としてJDBCドライバーを追加します。

```
<!-- https://mvnrepository.com/artifact/software.amazon.timestream/amazon-
timestream-jdbc -->
<dependency>
    <groupId>software.amazon.timestream</groupId>
    <artifactId>amazon-timestream-jdbc</artifactId>
    <version>1.0.1</version>
</dependency>
```

ドライバークラスとURL形式のタイムストリーム LiveAnalytics JDBC

ドライバーの Timestream の LiveAnalytics JDBCドライバークラスは次のとおりです。

```
software.amazon.timestream.jdbc.TimestreamDriver
```

Timestream JDBCドライバーには、次のJDBCURL形式が必要です。

```
jdbc:timestream:
```

を使用してデータベースプロパティを指定するにはJDBCURL、次のURL形式を使用します。

```
jdbc:timestream://
```

サンプルアプリケーション

LiveAnalytics で Timestream の使用を開始できるようにJDBC、 でフル機能のサンプルアプリケーションを作成しました GitHub。

1. ???「」で説明されている手順に従って、サンプルデータを含むデータベースを作成します。
2. サンプルアプリケーションのJDBC GitHub リポジトリをクローンし、 の指示に従いますGitHub。
3. サンプルアプリケーションの使用を開始するREADMEには、「」の手順に従います。

接続プロパティ

ドライバーの LiveAnalytics JDBC Timestream は、次のオプションをサポートしています。

トピック

- 基本的な認証オプション
- 標準クライアント情報オプション
- ドライバー設定オプション
- SDK オプション
- エンドポイント設定オプション
- 認証情報プロバイダーのオプション
- SAMLOkta の ベースの認証オプション
- SAMLAzure AD の ベースの認証オプション

> **Note**
>
> プロパティが指定されていない場合、ドライバーの LiveAnalytics JDBC Timestream はデフォルトの認証情報チェーンを使用して認証情報をロードします。

> **Note**
>
> すべてのプロパティキーでは、大文字と小文字が区別されます。

基本的な認証オプション

次の表に、使用可能な基本認証オプションを示します。

オプション	説明	デフォルト
AccessKeyId	AWS ユーザーアクセスキー ID。	NONE
SecretAccessKey	AWS ユーザーシークレットアクセスキー。	NONE
SessionToken	多要素認証 (MFA) が有効になっているデータベースにアクセスするために必要な一時的なセッショントークン。	NONE

標準クライアント情報オプション

次の表は、標準クライアント情報オプションについて説明しています。

オプション	説明	デフォルト
ApplicationName	現在接続を利用しているアプリケーションの名前。`ApplicationName` はデバッグ目的で使用され、Timestream にサービスとして LiveAnalytics通信されません。	ドライバーによって検出されたアプリケーション名。

ドライバー設定オプション

次の表は、ドライバー設定オプションについて説明しています。

オプション	説明	デフォルト
EnableMetaDataPreparedStatement	ドライバーが の LiveAnalytics JDBCメタデータを返す Timestream を有効にしますが`PreparedStatements` 、メタデータの取得 LiveAnalytics 時に の Timestream に追加コストが発生します。	FALSE
リージョン	データベースのリージョン。	us-east-1

SDK オプション

次の表は、 SDKオプションについて説明しています。

オプション	説明	デフォルト
RequestTimeout	がタイムアウトする前にクエリリクエストを待機するミリ秒単位の AWS SDK時間。正以外の値は、リクエストタイムアウトを無効にします。	0
SocketTimeout	は、タイムアウトする前に、オープン接続を介してデータが転送されるまで待機するミリ秒単位の AWS SDK時間です。値は負でない必要があります。の値は、ソケットタイムアウト0を無効にします。	50000
MaxRetryCountClient	の 5XX エラーコードで再試行可能なエラーの最大再試行回数SDK。値は負でない必要があります。	NONE
MaxConnections	LiveAnalytics サービスの Timestream HTTPへの同時オープン接続の最大数。値は正である必要があります。	50

エンドポイント設定オプション

次の表は、エンドポイント設定オプションについて説明しています。

オプション	説明	デフォルト
エンドポイント	LiveAnalytics サービスの Timestream のエンドポイント。	NONE

認証情報プロバイダーのオプション

次の表は、使用可能な認証情報プロバイダーのオプションを示しています。

オプション	説明	デフォルト
AwsCredentialsProviderClass	認証InstanceProfileCredentialsProvider に使用する PropertiesFileCredentialsProvider または の 1 つ。	NONE
CustomCredentialsFilePath	AWS セキュリティ認証情報accessKey と を含むプロパティファイルへのパスsecretKey 。これは、AwsCredentialsProviderClass が PropertiesFileCredentialsProvider として指定されている場合にのみ必要です。	NONE

SAMLOkta の ベースの認証オプション

次の表は、Okta で使用可能な SAMLベースの認証オプションを示しています。

オプション	説明	デフォルト
IdpName	SAMLベースの認証に使用する ID プロバイダー (Idp) 名。Okta または のいずれかAzureAD。	NONE
IdpHost	指定された Idp のホスト名。	NONE
IdpUserName	指定された Idp アカウントのユーザー名。	NONE

オプション	説明	デフォルト
IdpPassword	指定された Idp アカウントのパスワード。	NONE
OktaApplicationID	アプリケーションの Timestream LiveAnalytics に関連付けられた一意の Okta 提供 ID。 `AppId`は、アプリケーションメタデータで提供されている `entityID`フィールドにあります。次の例を考えてみましょう。 `entityID = http://www.okta.com//IdpAppID`	NONE
ロールARN	発信者が引き受けるロールの Amazon リソースネーム (ARN) 。	NONE
IDPARN	Idp を記述IAMする のSAMLプロバイダーの Amazon リソースネーム (ARN) 。	NONE

SAMLAzure AD の ベースの認証オプション

次の表は、Azure AD で使用可能な SAMLベースの認証オプションを示しています。

オプション	説明	デフォルト
IdpName	SAMLベースの認証に使用する ID プロバイダー (Idp) 名。`Okta` または `AzureAD` のいずれか。	NONE
IdpHost	指定された Idp のホスト名。	NONE

オプション	説明	デフォルト
IdpUserName	指定された Idp アカウントのユーザー名。	NONE
IdpPassword	指定された Idp アカウントのパスワード。	NONE
AADApplicationID	Azure AD に登録されたアプリケーションの一意の ID。	NONE
AADClientSecret	トークンの取得を承認するために使用される Azure AD の登録済みアプリケーションに関連付けられたクライアントシークレット。	NONE
AADTenant	Azure AD テナント ID。	NONE
IDPARN	Idp を記述IAMする のSAMLプロバイダーの Amazon リソースネーム (ARN) 。	NONE

JDBC URL 例

このセクションではURL、JDBC接続 を作成する方法と例について説明します。オプションの接続プロパティ を指定するには、次のURL形式を使用します。

```
jdbc:timestream://PropertyName1=value1;PropertyName2=value2...
```

> **Note**
>
> すべての接続プロパティはオプションです。すべてのプロパティキーでは、大文字と小文字が区別されます。

以下に、JDBC接続 の例をいくつか示しますURLs。

基本的な認証オプションとリージョンの例：

```
jdbc:timestream://
AccessKeyId=<myAccessKeyId>;SecretAccessKey=<mySecretAccessKey>;SessionToken=<mySessionToken
east-1
```

クライアント情報、リージョン、SDKオプションの例：

```
jdbc:timestream://ApplicationName=MyApp;Region=us-
east-1;MaxRetryCountClient=10;MaxConnections=5000;RequestTimeout=20000
```

デフォルトの認証情報プロバイダーチェーンを使用して接続し、環境変数に AWS 認証情報を設定します。

```
jdbc:timestream
```

デフォルトの認証情報プロバイダーチェーンを使用して接続し、接続 で AWS 認証情報を設定しますURL。

```
jdbc:timestream://
AccessKeyId=<myAccessKeyId>;SecretAccessKey=<mySecretAccessKey>;SessionToken=<mySessionToken
```

認証方法 PropertiesFileCredentialsProvider として を使用して接続します。

```
jdbc:timestream://
AwsCredentialsProviderClass=PropertiesFileCredentialsProvider;CustomCredentialsFilePath=<pat
 to properties file>
```

認証方法 InstanceProfileCredentialsProvider として を使用して接続します。

```
jdbc:timestream://AwsCredentialsProviderClass=InstanceProfileCredentialsProvider
```

Okta 認証情報を認証方法として使用して接続します。

```
jdbc:timestream://
IdpName=Okta;IdpHost=<host>;IdpUserName=<name>;IdpPassword=<password>;OktaApplicationID=<id>
```

認証方法として Azure AD 認証情報を使用して接続します。

```
jdbc:timestream://
IdpName=AzureAD;IdpUserName=<name>;IdpPassword=<password>;AADApplicationID=<id>;AADClientSec
```

特定のエンドポイントに接続します。

```
jdbc:timestream://Endpoint=abc.us-east-1.amazonaws.com;Region=us-east-1
```

Okta によるシングルサインオン認証の LiveAnalytics JDBC Timestream のセットアップ

の Timestream は、Okta によるシングルサインオン認証の LiveAnalytics JDBC Timestream LiveAnalytics をサポートします。Okta でのシングルサインオン認証に LiveAnalytics JDBC Timestream を使用するには、以下の各セクションを完了します。

トピック

- 前提条件
- AWS Okta でのアカウントフェデレーション
- の Okta のセットアップ SAML

前提条件

Okta でのJDBCシングルサインオン認証に LiveAnalytics Timestream を使用する前に、次の前提条件を満たしていることを確認してください。

- で ID プロバイダーとロール を作成 AWS するための管理者アクセス許可。
- Okta アカウント (に移動https://www.okta.com/login/してアカウントを作成します)。
- の Amazon Timestream へのアクセス LiveAnalytics。

前提条件を完了したので、 に進むことができますAWS Okta でのアカウントフェデレーション。

AWS Okta でのアカウントフェデレーション

ドライバーの LiveAnalytics JDBC Timestream は、Okta で AWS アカウントフェデレーションをサポートします。Okta で AWS アカウントフェデレーションを設定するには、次の手順を実行します。

1. 次の を使用して Okta 管理者ダッシュボードにサインインしますURL。

```
https://<company-domain-name>-admin.okta.com/admin/apps/active
```

> **Note**
>
> < company-domain-name > をドメイン名に置き換えます。

2. サインインに成功したら、アプリケーションの追加 を選択し、AWS アカウントフェデレーション を検索します。
3. 追加を選択する
4. ログインを適切な URLに変更しますURL。
5. [Next] (次へ) を選択します。
6. サインオンメソッドとして SAML 2.0 を選択する
7. Identity Provider メタデータを選択して、メタデータXMLファイルを開きます。ファイルをローカルに保存します。
8. 他のすべての設定オプションは空白のままにします。
9. [完了] を選択します。

Okta で AWS アカウントフェデレーションを完了したので、 に進むことができますの Okta のセットアップ SAML。

の Okta のセットアップ SAML

1. [Sign On] (サインオン) タブを選択します。ビュー を選択します。
2. 設定セクションで、セットアップ手順ボタンを選択します。

Okta メタデータドキュメントの検索

1. ドキュメントを検索するには、次に移動します。

```
https://<domain>-admin.okta.com/admin/apps/active
```

> **Note**
>
> <domain> は、Okta アカウントの一意のドメイン名です。

2. AWS Account Federation アプリケーションを選択する
3. サインオンタブを選択する

Microsoft Azure AD によるシングルサインオン認証の LiveAnalytics JDBC Timestream のセットアップ

の Timestream は、Microsoft Azure AD によるシングルサインオン認証の LiveAnalytics JDBC Timestream LiveAnalytics をサポートします。Microsoft Azure AD でのシングルサインオン認証に LiveAnalytics JDBC Timestream を使用するには、以下の各セクションを完了します。

トピック

- 前提条件
- Azure AD のセットアップ
- での IAM ID プロバイダーとロールの設定 AWS

前提条件

Microsoft Azure AD でのJDBCシングルサインオン認証に LiveAnalytics Timestream を使用する前に、次の前提条件を満たしていることを確認してください。

- で ID プロバイダーとロール を作成 AWS するための管理者アクセス許可。
- Azure Active Directory アカウント (https://azure.microsoft.com/en-ca/サービス/active-directory/ に移動してアカウントを作成する)
- の Amazon Timestream へのアクセス LiveAnalytics。

Azure AD のセットアップ

1. Azure Portal にサインインする
2. Azure サービスのリストで Azure Active Directory を選択します。これにより、デフォルトのディレクトリページにリダイレクトされます。
3. サイドバーの管理セクションでエンタープライズアプリケーションを選択する

4. + 新しいアプリケーション を選択します。
5. Amazon Web Services を検索して選択します。
6. サイドバーの管理セクションでシングルサインオンを選択する
7. シングルサインオンメソッドSAMLとして を選択します。
8. 基本SAML設定 セクションで、識別子 と 返信 の両方URLに以下を入力しますURL。

```
https://signin.aws.amazon.com/saml
```

9. [保存] を選択します。
10. SAML 署名証明書セクションXMLのフェデレーションメタデータをダウンロードします。これは、後で IAM Identity Provider を作成するときに使用されます。
11. デフォルトのディレクトリページに戻り、 の管理でアプリ登録を選択します。
12. All Applications セクションから の Timestream LiveAnalytics を選択します。ページがアプリケーションの概要ページにリダイレクトされます

> **Note**
>
> アプリケーション (クライアント) ID とディレクトリ (テナント) ID を書き留めます。これらの値は、接続を作成するときに に必要です。

13. 証明書とシークレットを選択する
14. クライアントシークレット で、+ 新しいクライアントシークレット を使用して新しいクライアントシークレットを作成します。

> **Note**
>
> の Timestream への接続を作成するときに必要になるため、生成されたクライアントシークレットに注意してください LiveAnalytics。

15. の管理のサイドバーで、APIアクセス許可を選択します。
16. 設定済みアクセス許可 で、アクセス許可を追加を使用して、 の Timestream にサインインするアクセス許可を Azure AD に付与します LiveAnalytics。アクセスAPI許可のリクエストページで Microsoft Graph を選択します。
17. 委任されたアクセス許可を選択し、User.Read アクセス許可を選択します。
18. アクセス許可の追加を選択する

19.デフォルトディレクトリの管理者同意を付与を選択します。

での IAM ID プロバイダーとロールの設定 AWS

Microsoft Azure AD によるシングルサインオン認証の IAM LiveAnalytics JDBC Timestream を設定するには、以下の各セクションを完了します。

トピック

- SAML ID プロバイダーを作成する
- IAM ロールを作成する
- IAM ポリシーを作成する
- プロビジョニング

SAML ID プロバイダーを作成する

Microsoft Azure AD によるシングルサインオン認証用の LiveAnalytics JDBC Timestream の SAML Identity Provider を作成するには、次の手順を実行します。

1. AWS マネジメントコンソールにサインインする
2. サービスを選択し、Security、Identity、および Compliance IAM で を選択します。
3. アクセス管理で ID プロバイダーを選択する
4. プロバイダーの作成を選択し、プロバイダータイプSAMLとして を選択します。プロバイダー名を入力します。この例では、A を使用しますzureADProvider。
5. 以前にダウンロードした Federation メタデータXMLファイルをアップロードする
6. Next を選択し、Create を選択します。
7. 完了すると、ページは ID プロバイダーページにリダイレクトされます

IAM ロールを作成する

Microsoft Azure AD によるシングルサインオン認証の LiveAnalytics JDBC Timestream のIAMロールを作成するには、次の手順を実行します。

1. サイドバーで、アクセス管理でロールを選択します。
2. [ロールの作成] を選択します。
3. 信頼されたエンティティとして SAML 2.0 フェデレーションを選択する

4. Azure AD プロバイダーを選択する
5. プログラムによるアクセスと AWS マネジメントコンソールへのアクセスを許可するを選択します。
6. [Next: Permissions] (次へ: アクセス権限) を選択します。
7. アクセス許可ポリシーをアタッチするか、Next:Tags に進みます。
8. オプションのタグを追加するか、Next:Review に進みます。
9. [Role name] (ロール名) に入力します。この例では、A を使用します。zureSAMLRole
10. ロールの説明を入力します。
11. ロールの作成を選択して完了する

IAM ポリシーを作成する

Microsoft Azure AD によるシングルサインオン認証の LiveAnalytics JDBC Timestream のIAMポリシーを作成するには、次の手順を実行します。

1. サイドバーで、アクセス管理のポリシーを選択します。
2. ポリシーの作成を選択し、 JSON タブを選択します。
3. 次のポリシーを追加する

```
{
"Version": "2012-10-17",
"Statement": [
    {
          "Effect": "Allow",
          "Action": [
                "iam:ListRoles",
                "iam:ListAccountAliases"
           ],
           "Resource": "*"
      }
]
}
```

4. [ポリシーの作成] を選択します。
5. ポリシー名を入力します。この例では を使用します TimestreamAccessPolicy。
6. ポリシーの作成を選択する
7. サイドバーで、アクセス管理でロールを選択します。

8. 以前に作成した Azure AD ロールを選択し、アクセス許可でポリシーのアタッチを選択します。
9. 以前に作成したアクセスポリシーを選択します。

プロビジョニング

Microsoft Azure AD によるシングルサインオン認証のために LiveAnalytics JDBC Timestream の ID プロバイダーをプロビジョニングするには、次の手順を実行します。

1. Azure Portal に戻る
2. Azure サービスのリストで Azure Active Directory を選択します。これにより、デフォルトのディレクトリページにリダイレクトされます。
3. サイドバーの管理セクションでエンタープライズアプリケーションを選択する
4. プロビジョニングの選択
5. プロビジョニング方法の自動モードを選択する
6. 管理認証情報で、クライアントシークレットの ID とシークレットトークンSecretAccessKeyの AwsAccessKeyID を入力します。
7. プロビジョニングステータスをオンに設定する
8. 保存 を選択します。これにより、Azure AD は必要なIAMロールをロードできます。
9. 現在のサイクルステータスが完了したら、サイドバーのユーザーとグループを選択します。
10選択 + ユーザーの追加
11.の Timestream へのアクセスを提供する Azure AD ユーザーを選択します。 LiveAnalytics
12.で作成された IAM Azure AD ロールと対応する Azure Identity Provider を選択します。 AWS
13割り当てを選択する

ODBC

の Amazon Timestream 用のオープンソースODBCドライバー LiveAnalytics は、開発者 LiveAnalytics 向けに Timestream への SQLリレーショナルインターフェイスを提供し、Power BI Desktop や Microsoft Excel などのビジネスインテリジェンス (BI) ツールからの接続を可能にします。ドライバーの LiveAnalytics ODBC Timestream は現在、Windows、macOS、Linux で利用可能であり、Okta および Microsoft Azure Active Directory (AD) でもサポートSSOされています。

詳細については、「」のODBC「ドライバードキュメントの LiveAnalytics Amazon Timestream GitHub」を参照してください。

トピック

ドライバーの Timestream LiveAnalytics ODBC のセットアップ

AWS アカウント LiveAnalytics で の Timestream へのアクセスを設定する

の Timestream を使用するように AWS アカウントをまだ設定していない場合は LiveAnalytics、「」のインサクションに従ってくださいの Timestream へのアクセス LiveAnalytics。

システムにODBCドライバーをインストールする

ODBC GitHub リポジトリ からシステムに適した Timestream ODBCドライバーインストーラをダウンロードし、システムに適用されるインストール手順に従ってください。

- Windows インストールガイド
- MacOS インストールガイド
- Linux インストールガイド

ODBC ドライバーのデータソース名 (DSN) を設定する

システムDSNの設定ガイドの手順に従ってください。

- Windows DSN設定
- MacOS DSN設定
- Linux DSN設定

ODBC ドライバーと連携するようにビジネスインテリジェンス (BI) アプリケーションをセットアップする

ODBC ドライバーを操作するためにいくつかの一般的な BI アプリケーションを設定する手順は次のとおりです。

- Microsoft Power BI のセットアップ。

- Microsoft Excel のセットアップ
- Tableau のセットアップ

他のアプリケーションの場合

ODBC ドライバーの接続文字列の構文とオプション

ODBC ドライバーの接続文字列オプションを指定するための構文は次のとおりです。

```
DRIVER={Amazon Timestream ODBC Driver};(option)=(value);
```

使用できるオプションは以下のとおりです。

ドライバー接続オプション

- **Driver**（必須）– で使用されているドライバーODBC。

  デフォルトは Amazon Timestream です。
- **DSN** – 接続の設定に使用するデータソース名 (DSN)。

  デフォルト: NONE。
- **Auth** – 認証モード。これは、次のいずれかである必要があります。
  - AWS_PROFILE – デフォルトの認証情報チェーンを使用します。
  - IAM – AWS IAM認証情報を使用します。
  - AAD – Azure Active Directory (AD) ID プロバイダーを使用します。
  - OKTA – Okta ID プロバイダーを使用します。

  デフォルト: AWS_PROFILE。

エンドポイント設定オプション

- **EndpointOverride** – LiveAnalytics サービスの Timestream のエンドポイントオーバーライド。これは、リージョンを上書きする高度なオプションです。例:

  ```
  query-cell2.timestream.us-east-1.amazonaws.com
  ```
- **Region** – LiveAnalytics サービスエンドポイントの Timestream の署名リージョン。

デフォルト: us-east-1。

認証情報プロバイダーオプション

- **ProfileName** – AWS 設定ファイルのプロファイル名。

  デフォルト: NONE。

AWS IAM 認証オプション

- **UID** または **AccessKeyId** – AWS ユーザーアクセスキー ID。接続文字列に UIDと の両方AccessKeyIdが指定されている場合、UID値は空でない限り使用されます。

  デフォルト: NONE。

- **PWD** または **SecretKey** – AWS ユーザーシークレットアクセスキー。接続文字列に PWDと の両方SecretKeyが指定されている場合、空でない限り、 PWDの値が使用されます。

  デフォルト: NONE。

- **SessionToken** – 多要素認証 (MFA) が有効になっているデータベースにアクセスするために必要な一時的なセッショントークン。 = 入力に末尾を含めないでください。

  デフォルト: NONE。

SAMLOkta の ベースの認証オプション

- **IdPHost** – 指定された IdP のホスト名。

  デフォルト: NONE。

- **UID** または **IdPUserName** – 指定された IdP アカウントのユーザー名。接続文字列に UIDと の両方IdPUserNameが指定されている場合、UID値は空でない限り使用されます。

  デフォルト: NONE。

- **PWD** または **IdPPassword** – 指定された IdP アカウントのパスワード。接続文字列に PWDと の両方IdPPasswordが指定されている場合、PWD値は空でない限り使用されます。

  デフォルト: NONE。

- **OktaApplicationID** – LiveAnalytics アプリケーションの Timestream に関連付けられた Okta が提供する一意の ID。アプリケーション ID (AppId) を検索する場所は、アプリケーションメタデータで指定された entityIDフィールドにあります。例として、次のものがあります。

  ```
  entityID="http://www.okta.com//(IdPAppID)
  ```

  デフォルト: NONE。
- **RoleARN** – 発信者が引き受けるロールの Amazon リソースネーム (ARN) 。

  デフォルト: NONE。
- **IdPARN** – IdP を記述IAMする のSAMLプロバイダーの Amazon リソースネーム (ARN) 。

  デフォルト: NONE。

SAMLAzure Active Directory の ベースの認証オプション

- **UID** または **IdPUserName** – 指定された IdP アカウントのユーザー名。

  デフォルト: NONE。
- **PWD** または **IdPPassword** – 指定された IdP アカウントのパスワード。

  デフォルト: NONE。
- **AADApplicationID** – Azure AD に登録されたアプリケーションの一意の ID。

  デフォルト: NONE。
- **AADClientSecret** – トークンの取得を承認するために使用される Azure AD の登録済みアプリケーションに関連付けられたクライアントシークレット。

  デフォルト: NONE。
- **AADTenant** – Azure AD テナント ID。

  デフォルト: NONE。
- **RoleARN** – 発信者が引き受けるロールの Amazon リソースネーム (ARN) 。

  デフォルト: NONE。
- **IdPARN** – IdP を記述IAMする のSAMLプロバイダーの Amazon リソースネーム (ARN) 。

  デフォルト: NONE。

AWS SDK (アドバンスド) オプション

- **RequestTimeout** – が AWS SDKタイムアウトする前にクエリリクエストを待機するミリ秒単位の時間。正以外の値は、リクエストタイムアウトを無効にします。

  デフォルト: 3000。

- **ConnectionTimeout** – タイムアウトする前に、 が AWS SDKオープン接続を介してデータを転送するのを待機するミリ秒単位の時間。値が 0 の場合、接続タイムアウトは無効になります。この値は負にすることはできません。

  デフォルト: 1000。

- **MaxRetryCountClient** – の 5xx エラーコードで再試行可能なエラーの最大再試行回数SDK。値は負にすることはできません。

  デフォルト: 0。

- **MaxConnections** – Timestream サービスへの同時オープンHTTP接続の最大許容数。値は正である必要があります。

  デフォルト: 25。

ODBC ドライバーのログ記録オプション

- **LogLevel** – ドライバーログ記録のログレベル。次のいずれかにする必要があります。
  - 0 (OFF) 。
  - 1 (ERROR) 。
  - 2 (WARNING) 。
  - 3 (INFO) 。
  - 4 (DEBUG) 。

  デフォルトは 1 () ですERROR。

  警告： ログ記録モードを使用すると、ドライバーが個人情報DEBUGを記録する可能性があります。

- **LogOutput** – ログファイルを保存するフォルダ。

  デフォルトは次のとおりです。

  - Windows: %USERPROFILE%、または利用できない場合は、%HOMEDRIVE%%HOMEPATH%。

- macOS と Linux: $HOME、または使用できない場合は、関数pw_dirの フィールドが値getpwuid(getuid())を返します。

SDK ログ記録オプション

AWS SDK ログレベルは、ドライバーログレベルの Timestream LiveAnalytics ODBCとは別です。一方を設定しても、他方には影響しません。

SDK ログレベルは、環境変数 を使用して設定されますTS_AWS_LOG_LEVEL。有効な値は次のとおりです。

- OFF
- ERROR
- WARN
- INFO
- DEBUG
- TRACE
- FATAL

が設定されTS_AWS_LOG_LEVELていない場合、SDKログレベルはデフォルト に設定されますWARN。

プロキシを介した接続

ODBC ドライバーは、プロキシ LiveAnalytics を介した の Amazon Timestream への接続をサポートしています。この機能を使用するには、プロキシ設定に基づいて次の環境変数を設定します。

- **TS_PROXY_HOST** – プロキシホスト。
- **TS_PROXY_PORT** – プロキシポート番号。
- **TS_PROXY_SCHEME** – プロキシスキーム、 http または https。
- **TS_PROXY_USER** – プロキシ認証のユーザー名。
- **TS_PROXY_PASSWORD** – プロキシ認証のユーザーパスワード。
- **TS_PROXY_SSL_CERT_PATH** – HTTPSプロキシへの接続に使用するSSL証明書ファイル。
- **TS_PROXY_SSL_CERT_TYPE** – プロキシクライアントSSL証明書のタイプ。
- **TS_PROXY_SSL_KEY_PATH** – HTTPSプロキシへの接続に使用するプライベートキーファイル。

- **TS_PROXY_SSL_KEY_TYPE** – HTTPSプロキシへの接続に使用されるプライベートキーファイルのタイプ。
- **TS_PROXY_SSL_KEY_PASSWORD** – HTTPSプロキシへの接続に使用されるプライベートキーファイルへのパスフレーズ。

ドライバーの Timestream LiveAnalytics ODBC の接続文字列の例

IAM 認証情報を使用してODBCドライバーに接続する例

```
Driver={Amazon Timestream ODBC Driver};Auth=IAM;AccessKeyId=(your access key
 ID);secretKey=(your secret key);SessionToken=(your session token);Region=us-east-2;
```

プロファイルを使用してODBCドライバーに接続する例

```
Driver={Amazon Timestream ODBC Driver};ProfileName=(the profile name);region=us-west-2;
```

ドライバーは、 で提供されている認証情報を使用して接続を試みます。または`~/.aws/credentials`、環境変数 でファイルが指定されている場合は`AWS_SHARED_CREDENTIALS_FILE`、そのファイルの認証情報を使用して接続を試みます。

Okta でODBCドライバーに接続する例

```
driver={Amazon Timestream ODBC Driver};auth=okta;region=us-west-2;idPHost=(your host at
 Okta);idPUsername=(your user name);idPPassword=(your password);OktaApplicationID=(your
 Okta AppId);roleARN=(your role ARN);idPARN=(your Idp ARN);
```

Azure Active Directory を使用してODBCドライバーに接続する例 (AAD)

```
driver={Amazon Timestream ODBC Driver};auth=aad;region=us-west-2;idPUsername=(your
 user name);idPPassword=(your password);aadApplicationID=(your AAD
 AppId);aadClientSecret=(your AAD client secret);aadTenant=(your AAD
 tenant);roleARN=(your role ARN);idPARN=(your idP ARN);
```

指定されたエンドポイントとログレベルが 2 のODBCドライバーに接続する例 (WARNING)

```
Driver={Amazon Timestream ODBC Driver};Auth=IAM;AccessKeyId=(your access
 key ID);secretKey=(your secret key);EndpointOverride=ingest.timestream.us-
west-2.amazonaws.com;Region=us-east-2;LogLevel=2;
```

ODBC ドライバーとの接続のトラブルシューティング

> **Note**
>
> ユーザー名とパスワードが で既に指定されている場合DSN、ODBCドライバーマネージャーが要求したときにユーザー名とパスワードを再度指定する必要はありません。

メッセージ01S02を含む のエラーコードは、接続文字列オプションが接続文字列で複数回渡された場合にRe-writing *(connection string option)* (have you specified it several times?発生します。オプションを複数回指定すると、エラーが発生します。DSN と接続文字列との接続を行う場合、 で接続オプションが既に指定されている場合はDSN、接続文字列で再度指定しないでください。

VPC エンドポイント (AWS PrivateLink)

インターフェイスVPCエンドポイント を作成 LiveAnalytics することで、 VPCと の Amazon Timestream 間にプライベート接続を確立できます。詳細については、「VPC エンドポイント (AWS PrivateLink)」を参照してください。

ベストプラクティス

の Amazon Timestream の利点を完全に実現するには LiveAnalytics、以下のベストプラクティスに従ってください。

> **Note**
>
> アプリケーションを実行する proof-of-conceptときは、 の Timestream のパフォーマンスとスケールを評価しながら、アプリケーションが数か月または数年にわたって蓄積するデータの量を考慮してください LiveAnalytics。時間の経過とともにデータが増大するにつれて、 の Timestream のパフォーマンスはほぼ変化 LiveAnalytics しないことに気付くでしょう。サーバーレスアーキテクチャは大量の並列処理を活用してより大きなデータボリュームを処理し、アプリケーションのニーズに合わせて自動的にスケーリングできるためです。

トピック

- データモデリング
- セキュリティ

- の Amazon Timestream の設定 LiveAnalytics
- 書き込み
- クエリ
- スケジュールされたクエリ
- クライアントアプリケーションとサポートされている統合
- 全般

データモデリング

Amazon Timestream for LiveAnalytics は、タイムスタンプ付きの一連のデータを送信するアプリケーションやデバイスから時系列データを収集、保存、分析するように設計されています。最適なパフォーマンスを得るには、 の Timestream に送信するデータに時間的特性 LiveAnalytics があり、時間がデータの本質的なコンポーネントである必要があります。

の Timestream LiveAnalytics は、アプリケーションの要件に合わせてさまざまな方法でデータをモデル化する柔軟性を提供します。このセクションでは、これらのパターンのいくつかについて説明し、コストとパフォーマンスを最適化するためのガイドラインを提供します。ディメンションやメジャーなどの LiveAnalytics 概念について、主要な Timestream に精通してください。このセクションでは、以下について詳しく学習します。

データを保存するために単一のテーブルを作成するか、複数のテーブルを作成するかを決定するときは、以下を考慮してください。

- 同じテーブルに配置するデータと、複数のテーブルとデータベース間でデータを分離する場合。
- LiveAnalytics マルチメジャーレコードと単一メジャーレコードのどちらに Timestream を選択するか、特にアプリケーションが複数の測定値を同時に追跡している場合にマルチメジャーレコードを使用してモデリングする利点。
- ディメンションまたはメジャーとしてモデル化する属性。
- メジャー名属性を効果的に使用してクエリのレイテンシーを最適化する方法。

トピック

- 単一のテーブルと複数のテーブル
- マルチメジャーレコードと単一メジャーレコードの比較
- ディメンションとメジャー
- マルチメジャーレコードでのメジャー名の使用

- マルチメジャーレコードをパーティション分割するための推奨事項

単一のテーブルと複数のテーブル

アプリケーションでデータをモデリングする場合、もう 1 つの重要な側面は、データをテーブルとデータベースにモデル化する方法です。の Timestream のデータベースとテーブル LiveAnalytics は、アクセスコントロール、KMSキーの指定、保持期間などの抽象化です。のデータを自動的にパーティション化する Timestream LiveAnalytics は、アプリケーションの取り込み、ストレージ、クエリの負荷と要件に合わせてリソースをスケーリングするように設計されています。

の Timestream のテーブル LiveAnalytics は、保存されているデータペタバイト、数十ギガバイト/秒のデータ書き込み、およびクエリで 1 時間TBsあたり数百を処理できます。の Timestream のクエリ LiveAnalytics は、複数のテーブルとデータベースにまたがることができ、結合と結合を提供して、複数のテーブルとデータベース間でデータへのシームレスなアクセスを提供します。したがって、の Timestream でデータを整理する方法を決定する場合、データまたはリクエストボリュームのスケールは通常、主な懸念事項ではありません LiveAnalytics。以下は、同じテーブルと異なるテーブル、または異なるデータベースのテーブルに、どのデータを共同配置するかを決定する際の重要な考慮事項です。

- データ保持ポリシー (メモリストア保持、磁気ストア保持など) は、テーブルの粒度でサポートされます。したがって、異なる保持ポリシーを必要とするデータは、異なるテーブルに存在する必要があります。
- AWS KMS データの暗号化に使用されるキーは、データベースレベルで設定されます。したがって、暗号化キーの要件が異なる場合、データは異なるデータベースに存在する必要があることを意味します。
- の Timestream は、テーブルとデータベースの粒度でリソースベースのアクセスコントロール LiveAnalytics をサポートします。同じテーブルに書き込むデータと異なるテーブルに書き込むデータを決定するときは、アクセスコントロールの要件を考慮してください。
- どのデータをどのテーブルに保存するかを決定するときは、ディメンション数、メジャー名、マルチメジャー属性名の制限に注意してください。
- クエリのレイテンシーと書き込みのしやすさはそれに依存するため、データの整理方法を決定する際には、クエリのワークロードとアクセスパターンを考慮してください。
  - 同じテーブルに頻繁にクエリするデータを保存すると、クエリの書き込み方法が一般的に容易になり、結合、結合、または一般的なテーブル式を書き込む必要がなくなります。また、通常、クエリのレイテンシーも低くなります。ディメンションとメジャー名の述語を使用して、クエリに関連するデータをフィルタリングできます。

例えば、6 つの大陸にあるデバイスからのデータを保存する場合を考えてみましょう。クエリが複数の大陸からのデータに頻繁にアクセスしてグローバル集計ビューを取得する場合、これらの大陸からのデータを同じテーブルに保存すると、クエリの書き込みが容易になります。一方、データを異なるテーブルに保存しても、同じクエリでデータを結合することはできますが、テーブル間でデータを結合するにはクエリを記述する必要があります。

- の Timestream LiveAnalytics は、データに対してアダプティブパーティショニングとインデックス作成を使用します。そのため、クエリはクエリに関連するデータに対してのみ課金されます。例えば、6 大陸にまたがる 100 万台のデバイスからのデータを保存するテーブルがある場合、クエリに `WHERE device_id = 'abcdef'`または の述語がある場合`WHERE continent = 'North America'`、クエリはデバイスまたは大陸のデータに対してのみ課金されます。
- 可能な限り、メジャー名を使用して、同時に出力されない、または頻繁にクエリされない同じテーブル内のデータを分離する場合、クエリ`WHERE measure_name = 'cpu'`で などの述語を使用すると、計測の利点が得られるだけでなく、 の Timestream LiveAnalytics は、クエリ述語でメジャー名が使用されていないパーティションを効果的に削除することもできます。これにより、クエリのレイテンシーやコストに影響を与えることなく、同じテーブルに異なるメジャー名を持つ関連データを保存でき、データを複数のテーブルに分散することを回避できます。メジャー名は、基本的にデータをパーティション化し、クエリとは無関係なパーティションをプルーンするために使用されます。

マルチメジャーレコードと単一メジャーレコードの比較

の Timestream LiveAnalytics では、レコードごとに複数のメジャー (マルチメジャー) またはレコードごとに 1 つのメジャー (単一メジャー) を使用してデータを書き込むことができます。

マルチメジャーレコード

多くの場合、追跡しているデバイスまたはアプリケーションは、同じタイムスタンプで複数のメトリクスまたはイベントを出力することがあります。このような場合は、同じタイムスタンプで出力されたすべてのメトリクスを同じマルチメジャーレコードに保存できます。つまり、同じマルチメジャーレコードに保存されているすべてのメジャーは、同じデータの行に異なる列として表示されます。

例えば、アプリケーションが cpu、メモリ、disk_iops などのメトリクスを、同時に測定されるデバイスから出力しているとします。以下は、同時に出力される複数のメトリクスが同じ行に保存されるテーブルの例です。2 つのホストが 1 秒に 1 回メトリクスを送信していることがわかります。

ホスト名	measure_name	時間	cpu	「メモリ」	disk_iops
host-24Gju	メトリクス	2021-12-01 19:00:00	35	54.9	38.2
host-24Gju	メトリクス	2021-12-01 19:00:01	36	58	39
host-28Gju	メトリクス	2021-12-01 19:00:00	15	55	92
host-28Gju	メトリクス	2021-12-01 19:00:01	16	50	40

単一測定レコード

単一メジャーレコードは、デバイスが異なる期間に異なるメトリクスを出力する場合や、metrics/events at different time periods (for instance, when a device's reading/state変更を出力するカスタム処理ロジックを使用している場合に適しています)。すべてのメジャーには一意のタイムスタンプがあるため、 の Timestream の独自のレコードにメジャーを保存できます LiveAnalytics。例えば、土壌の温度と湿度を追跡する IoT センサーを考えてみましょう。このセンサーは、以前に報告されたエントリからの変化を検出した場合にのみレコードを生成します。次の例は、単一メジャーレコードを使用して出力されるこのようなデータの例を示しています。

device_id	measure_name	時間	measure_value::double	measure_value::bigint
sensor-sea478	temperature	2021-12-01 19:22:32	35	NULL
sensor-sea478	temperature	2021-12-01 18:07:51	36	NULL
sensor-sea478	湿度	2021-12-01 19:05:30	NULL	21

device_id	measure_name	時間	measure_value::double	measure_value::bigint
sensor-sea478	湿度	2021-12-01 19:00:01	NULL	23

単一メジャーレコードとマルチメジャーレコードの比較

の Timestream LiveAnalytics は、アプリケーションの要件と特性に応じて、データを単一メジャーまたは複数メジャーのレコードとしてモデル化する柔軟性を提供します。アプリケーション要件に応じて、単一のテーブルに単一のメジャーレコードとマルチメジャーレコードの両方を保存できます。一般的に、アプリケーションが複数のメジャー/イベントを同時に出力する場合は、パフォーマンスの高いデータアクセスとコスト効率の高いデータストレージのために、マルチメジャーレコードとしてデータをモデリングすることが推奨されます。

例えば、数十万のサーバーからのDevOps ユースケース追跡メトリクスとイベントを考慮すると、各サーバーは定期的に 20 個のメトリクスと 5 個のイベントを出力し、イベントとメトリクスは同時に出力されます。このデータは、単一メジャーレコードまたはマルチメジャーレコードを使用してモデル化できます (結果のスキーマについては、オープンソースのデータジェネレーターを参照してください)。このユースケースでは、単一メジャーレコードと比較してマルチメジャーレコードを使用してデータをモデリングすると、次の結果になります。

- 取り込み計測 - マルチメジャーレコードでは、書き込み取り込みバイト数が約 40% 減少します。
- 取り込みバッチ処理 - マルチメジャーレコードは、送信されるデータのバッチが大きくなり、クライアントが取り込みCPUを処理するために必要なスレッドが少なく、少ないことを意味します。
- ストレージの計測 - マルチメジャーレコードでは、ストレージ が約 8X減少するため、メモリとマグネティックストアの両方でストレージが大幅に節約されます。
- クエリレイテンシー - マルチメジャーレコードは、単一メジャーレコードと比較して、ほとんどのクエリタイプでクエリレイテンシーが低くなります。
- クエリ計測バイト数 - 10MB 未満のデータをスキャンするクエリの場合、単一メジャーレコードとマルチメジャーレコードの両方が同等です。単一メジャーにアクセスして > 10MB データをスキャンするクエリの場合、単一メジャーレコードは通常、計測されるバイト数が低くなります。3 つ以上のメジャーを参照するクエリの場合、マルチメジャーレコードは計測されたバイト数を低くします。

- マルチメジャークエリの表現のしやすさ - クエリが複数のメジャーを参照する場合、マルチメジャーレコードでデータをモデリングすると、よりコンパクトなクエリの書き込みが容易になります。

前の要素は、追跡するメトリクスの数、データに含まれるディメンションの数などによって異なります。前の例では 1 つの例について具体的なデータをいくつか示していますが、多くのアプリケーションシナリオとユースケースで、アプリケーションが複数のメジャーを同時に実行した場合、データをマルチメジャーレコードとして保存することがより効果的であることがわかります。さらに、マルチメジャーレコードは、データ型の柔軟性を提供し、他の複数の値をコンテキストとして保存します (リクエストの保存IDs、および後で説明する追加のタイムスタンプなど) 。

マルチメジャーレコードは、前述の例のような単一メジャーレコードのスパースメジャーをモデル化することもできます。measure_name を使用してメジャーの名前を保存し、value_double を使用してDOUBLEメジャーを保存したり、value_bigint を使用してBIGINTメジャーを保存したり、value_timestamp を使用して追加のTIMESTAMP値を保存したりできます。

ディメンションとメジャー

の Timestream のテーブル LiveAnalytics を使用すると、ディメンション (保存しているデバイス/データの属性を特定する) と測定値 (追跡しているメトリクス/値) を保存できます。詳細については、「Timestream の LiveAnalytics 概念」を参照してください。の Timestream でアプリケーションをモデル化する場合 LiveAnalytics、データをディメンションとメジャーにマッピングする方法は、取り込みとクエリのレイテンシーに影響します。以下は、ユースケースに適用できるディメンションと対策としてデータをモデル化する方法に関するガイドラインです。

ディメンションの選択

時系列データを送信しているソースを識別するデータは、時間の経過とともに変化しない属性であるディメンションに自然に適合します。例えば、サーバーがメトリクスを出力している場合、ホスト名、リージョン、ラック、アベイラビリティーゾーンなど、サーバーを識別する属性がディメンションの候補になります。同様に、時系列データをレポートする複数のセンサーを持つ IoT デバイスの場合、デバイス ID、センサー ID などがディメンションの候補になります。

マルチメジャーレコードとしてデータを記述する場合、テーブルで DESCRIBEまたは SELECTステートメントを実行すると、ディメンションとマルチメジャー属性が列としてテーブルに表示されます。したがって、クエリを記述するときは、同じクエリでディメンションとメジャーを自由に使用できます。ただし、データを取り込むように書き込みレコードを作成するときは、ディメンションとして指定される属性と測定値として指定される属性を選択する際に、次の点に注意してください。

- ディメンション名、値、メジャー名、タイムスタンプは、時系列データを一意に識別します。の Timestream LiveAnalytics は、この一意の識別子を使用してデータを自動的に重複排除します。つまり、 の Timestream がディメンション名、ディメンション値、メジャー名、タイムスタンプの値が同じ 2 つのデータポイント LiveAnalytics を受信した場合、値のバージョン番号が同じであれば、重複排除の LiveAnalytics Timestream になります。新しい書き込みリクエストのバージョンが、 の Timestream に既に存在するデータより低い場合 LiveAnalytics、書き込みリクエストは拒否されます。新しい書き込みリクエストのバージョンがより高い場合、新しい値は古い値を上書きします。したがって、ディメンション値の選択方法は、この重複排除動作に影響します。
- ディメンション名と値は更新できません。測定値は にできます。したがって、更新が必要なデータはすべて、測定値としてモデル化されます。例えば、工場の床に色を変更できるマシンがある場合、重複排除に必要な識別属性として色を使用する場合を除き、その色を測定値としてモデル化できます。つまり、測定値を使用して、時間の経過とともに変化が遅い属性を保存できます。

の Timestream のテーブル LiveAnalytics は、ディメンション名と値の一意の組み合わせの数を制限しないことに注意してください。例えば、このような一意の値の組み合わせを数十億個テーブルに保存できます。ただし、次の例に示すように、ディメンションとメジャーを慎重に選択することで、特にクエリの場合、リクエストのレイテンシーを大幅に最適化できます。

ディメンションIDsで一意

アプリケーションシナリオで、すべてのデータポイント (リクエスト ID、トランザクション ID、相関 ID など) に一意の識別子を保存する必要がある場合、ID 属性を測定値としてモデリングすると、クエリのレイテンシーが大幅に向上します。マルチメジャーレコードでデータをモデリングすると、ID は他のディメンションや時系列データとコンテキストで同じ行に表示されるため、クエリは引き続き効果的に使用できます。例えば、サーバーによって出力されるすべてのデータポイントDevOps に一意のリクエスト ID 属性がある場合、リクエスト ID をメジャー値としてモデリングすると、一意のリクエスト ID をディメンションとしてモデリングするのではなく、異なるクエリタイプ間でクエリレイテンシーが最大 4 倍低くなります。

すべてのデータポイントで完全に一意ではないが、数十万または数百万の一意の値を持つ属性に同様の類推を使用できます。これらの属性は、ディメンション値またはメジャー値としてモデル化できます。前述のように書き込みパスで重複排除に値が必要な場合、またはクエリでその属性の値に対して等価述語を持つ WHERE句 (例えば、アプリケーションが数百万のデバイスを追跡`device_id = 'abcde'`している場合など) として頻繁に使用する場合、ディメンションとしてモデル化します。

マルチメジャーレコードによるデータ型の豊富さ

マルチメジャーレコードは、データを効果的にモデル化するための柔軟性を提供します。マルチメジャーレコードに保存するデータは、ディメンションに似た列としてテーブルに表示されるため、ディメンションとメジャー値のクエリも同じように容易になります。これらのパターンのいくつかは、前述の例で確認しました。マルチメジャーレコードを効果的に使用してアプリケーションのユースケースに対応するための追加パターンを以下に示します。

マルチメジャーレコードは、データ型 DOUBLE、BIGINT、、VARCHARBOOLEAN、および の属性をサポートしますTIMESTAMP。したがって、これらはさまざまなタイプの属性に自然に適合します。

- 位置情報 : 例えば、場所を追跡する場合 (緯度と経度で表されます) 、マルチメジャー属性としてモデル化すると、特に緯度と経度を前提としている場合、VARCHAR属性として保存するよりもクエリレイテンシーが低くなります。
- レコード内の複数のタイムスタンプ : アプリケーションシナリオで時系列レコードの複数のタイムスタンプを追跡する必要がある場合は、マルチメジャーレコードの追加属性としてモデル化できます。このパターンを使用して、将来のタイムスタンプまたは過去のタイムスタンプを含むデータを保存できます。すべてのレコードは、時間列のタイムスタンプを使用して、レコードをパーティション化、インデックス化、一意に識別することに注意してください。

特に、クエリに述語がある数値データまたはタイムスタンプがある場合、ディメンションではなくマルチメジャー属性としてこれらの属性をモデリングすると、クエリのレイテンシーが低くなります。これは、マルチメジャーレコードでサポートされているリッチデータ型を使用してこのようなデータをモデル化する場合、ディメンションなどのデータをモデル化した場合、 から別のデータ型VARCHARに値をキャストする代わりに、ネイティブデータ型を使用して述語を表現できるためです。

マルチメジャーレコードでのメジャー名の使用

の Timestream のテーブルは、メジャー名と呼ばれる特別な属性 (または列) LiveAnalytics をサポートします。の Timestream に書き込むレコードごとに、この属性の値を指定します LiveAnalytics。単一メジャーレコードの場合、メトリクスの名前 (cpu、サーバーメトリクスのメモリ、センサーメトリクスの温度、圧力など) を使用することは自然なことです。マルチメジャーレコードを使用する場合、マルチメジャーレコードの属性には名前が付けられるため (これらの名前はテーブルの列名になるため) 、cpu、メモリ、または温度により、圧力はマルチメジャー属性名になる可能性があります。したがって、自然な質問は、メジャー名を効果的に使用する方法です。

の Timestream LiveAnalytics は、測定名属性の値を使用してデータをパーティション化およびインデックス化します。したがって、テーブルに複数の異なるメジャー名があり、クエリがそれらの値をクエリ述語として使用している場合、 の Timestream はカスタムパーティショニン

グとインデックス作成を使用して、クエリに関連しないデータをプルアウト LiveAnalytics できます。例えば、テーブルに cpu とメモリのメジャー名があり、クエリに述語 がある場合WHERE measure_name = 'cpu'、 の Timestream は、この例のメジャー名メモリを持つ行など、クエリに関連しないメジャー名のデータを効果的にプルーン LiveAnalytics できます。このプルーニングは、マルチメジャーレコードでメジャー名を使用する場合にも適用されます。メジャー名属性は、テーブルのパーティショニング属性として効果的に使用できます。測定名とディメンション名と値、および時間は、 LiveAnalytics テーブルの Timestream 内のデータをパーティション化するために使用します。 LiveAnalytics テーブルの Timestream で許可される一意のメジャー名の数の制限に注意してください。また、メジャー名はメジャー値データ型とも関連付けられます。例えば、単一のメジャー名は 1 つのタイプのメジャー値にのみ関連付けることができます。このタイプは、DOUBLE、、BIGINT、BOOLEANVARCHAR、および のいずれかですMULTI。メジャー名で保存されたマルチメジャーレコードのデータ型は になりますMULTI。1 つのマルチメジャーレコードは、異なるデータ型 (DOUBLE、BIGINT、、VARCHAR、TIMESTAMP) を持つ複数のメトリクスを保存できるためBOOLEAN、マルチメジャーレコードに異なるタイプのデータを関連付けることができます。

以下のセクションでは、測定名属性を使用して同じテーブル内の異なるタイプのデータをグループ化する方法のいくつかの異なる例について説明します。

品質と価値を報告する IoT センサー

IoT センサーからのアプリケーションモニタリングデータがあるとします。各センサーは、温度、圧力などのさまざまな測定値を追跡します。実際の値に加えて、センサーは測定値の品質も報告します。これは測定値の正確性の尺度であり、測定の単位です。品質、単位、および値は一緒に出力されるため、Device_id がディメンション、品質、値、および単位がマルチメジャー属性である以下のデータ例に示すように、マルチメジャーレコードとしてモデル化できます。

device_id	measure_name	時間	Quality	値	単位
sensor-sea478	temperature	2021-12-01 19:22:32	92	35	c
sensor-sea478	temperature	2021-12-01 18:07:51	93	34	c
sensor-sea478	pressure	2021-12-01 19:05:30	98	31	psi

device_id	measure_name	時間	Quality	値	単位
sensor-sea478	pressure	2021-12-01 19:00:01	24	132	psi

このアプローチにより、マルチメジャーレコードの利点と、メジャー名の値を使用したデータのパーティショニングとプルーニングを組み合わせることができます。クエリが温度などの単一のメジャーを参照している場合は、クエリにメジャー名の述語を含めることができます。以下に、品質が 90 を超える測定の単位も投影する、このようなクエリの例を示します。

```
SELECT device_id, time, value AS temperature, unit
FROM db.table
WHERE time > ago(1h)
    AND measure_name = 'temperature'
    AND quality > 90
```

クエリで measure_name 述語を使用すると LiveAnalytics 、 の Timestream はクエリに関連しないパーティションとデータを効果的にプルーンできるため、クエリのレイテンシーが向上します。

また、すべてのメトリクスが同じタイムスタンプで出力され、複数のメトリクスが同じクエリで一緒にクエリされている場合、すべてのメトリクスを同じマルチメジャーレコードに保存することもできます。例えば、属性 temperature_quality、 temperature_value、 temperature_unit、 pressure_quality、 pressure_value、 pressure_unit などを使用して、マルチメジャーレコードとして構築できます。単一メジャーレコードとマルチメジャーレコードを使用したデータのモデリングについて前述した点の多くは、データのモデリング方法の決定に適用されます。クエリアクセスパターンとデータの生成方法を考慮して、コスト、取り込みとクエリのレイテンシー、クエリの作成のしやすさを最適化するモデルを選択します。

同じテーブル内のさまざまなタイプのメトリクス

マルチメジャーレコードをメジャー名値と組み合わせることができるもう 1 つのユースケースは、同じデバイスから独立して出力されるさまざまなタイプのデータをモデル化することです。 DevOps モニタリングのユースケースサーバーが定期的に生成されるメトリクスと不規則なイベントという 2 種類のデータを送信していることを考慮してください。このアプローチの例は、 DevOps ユースケース をモデリングするデータジェネレーターで説明されているスキーマです。この場合、異なるメジャー名を使用して、同じサーバーから出力される異なるタイプのデータを同じテーブルに保存できます。例えば、同時に出力されるすべてのメトリクスは、測定名メトリクスとともに保存されま

す。メトリクスから別の時点で出力されるすべてのイベントは、メジャー名イベントとともに保存されます。テーブルのメジャースキーマ (SHOW MEASURESクエリの出力など) は次のとおりです。

measure_name	data_type	ディメンション
events	マルチ	[{「data_type」varch ar「dimension_name」 availability_zone」,{「data_t ype」varchar「dimens ion_name」microserv ice_name」,{「data_t ype」varchar「dimens ion_name」instance_name」, {「data_type」varchar「d imension_name」inst ance_name」,{「data_ type」valchar_name「 dimension_name」_name」, {「data_type」varchar_nam e_name」
メトリクス	マルチ	[{「data_type」、「dim ension_name」、「data _type」、「varchar」、 「dimension_name」、 「microservice_name」、 「data_type」、「varch ar」、「dimension_name」、 「data_type」、「varchar_typ e」、「dimension_name 」、「data_type」、「var char_type」、」、「varc har_type」、「dimensi on_name_name」、」、」、 」、「

この場合、イベントとメトリクスのディメンションのセットも異なることがわかります。イベントには異なるディメンション jdk_version と process_name があり、メトリクスにはディメンション instance_type と os_version があります。

異なるメジャー名を使用すると、メトリクスのみを取得するなど`WHERE measure_name = 'metrics'`、述語を使用してクエリを記述できます。また、同じテーブル内の同じインスタンスから出力されるすべてのデータがある場合、インスタンス_name述語を使用してより単純なクエリを記述して、そのインスタンスのすべてのデータを取得することもできます。例えば、measure_name述語`WHERE instance_name = 'instance-1234'`のないフォームの述語は、特定のサーバーインスタンスのすべてのデータを返します。

マルチメジャーレコードをパーティション分割するための推奨事項

> **Important**
>
> このセクションは非推奨です。
> これらのレコメンデーションは古くなっています。パーティショニングは、ユーザー定義のパーティションキー を使用してより適切に制御できるようになりました。

時系列エコシステムでは、大量のデータを取り込んで保存する必要があるワークロードが増え、同時にディメンション値のカーディナリティの高いセットでデータにアクセスするときに低レイテンシーのクエリレスポンスが必要になることがわかりました。

このような特性により、このセクションの推奨事項は、以下を含むお客様のワークロードに役立ちます。

- マルチメジャーレコードを採用または採用したい。
- システムに大量にデータが入り、長期間保存されることが予想されます。
- メインアクセス (クエリ) パターンには低レイテンシーの応答時間が必要です。
- 最も重要なクエリパターンには、述語で何らかのフィルタリング条件が含まれることに注意してください。このフィルタリング条件は、高いカーディナリティディメンションに基づいています。例えば、、、ServerID DeviceId UserId、ホスト名などによるイベントや集計を考えてみましょう。

このような場合、エンジンはマルチメジャー名を使用してデータをパーティション化し、単一の値を持つとパーティションの利点が制限されるため、すべてのマルチメジャーの単一名は役に立ちません。これらのレコードのパーティショニングは、主に 2 つのディメンションに基づいています。時

間が x 軸にあり、ディメンション名のハッシュと y 軸measure_nameのハッシュがあるとします。このような場合measure_name、 はほぼパーティショニングキーのように機能します。

推奨事項は次のとおりです。

- 前述のようなユースケースのためにデータをモデリングする場合は、メインクエリアクセスパターンの直接派生measure_nameである を使用します。例:
  - ユースケースでは、エンドユーザーの観点からアプリケーションのパフォーマンスと QoE を追跡する必要があります。これは、単一のサーバーまたは IoT デバイスの測定値を追跡する場合もあります。
  - によるクエリとフィルタリングを行う場合 UserIdは、取り込み時に measure_nameに関連付ける最適な方法を見つける必要があります UserId。
  - マルチメジャーテーブルは 8192 個の異なるメジャー名しか保持できないため、採用される式が 8192 個の異なる値以上を生成しないでください。
- 文字列値に成功して適用したアプローチの 1 つは、文字列値にハッシュアルゴリズムを適用することです。次に、ハッシュ結果の絶対値と 8192 を使用してモジュロオペレーションを実行します。

```
measure_name = getMeasureName(UserId)
int getMeasureName(value) {
    hash_value =  abs(hash(value))
    return hash_value % 8192
}
```

- また、 記号を削除abs()し、値が -8192 から 8192 の範囲になる可能性を排除するために を追加しました。これは、モジュロオペレーションの前に実行する必要があります。
- この方法を使用すると、クエリは、パーティション化されていないデータモデルで実行されるのにかかる時間のごく一部で実行できます。
- データをクエリするときは、新たに導出された measure_name の値を使用するフィルタリング条件を述語に含めるようにしてください。例:
  -

```
SELECT * FROM your_database.your_table
WHERE host_name = 'Host-1235' time BETWEEN '2022-09-01'
    AND '2022-09-18'
    AND measure_name = (SELECT
 cast(abs(from_big_endian_64(xxhash64(CAST('HOST-1235' AS varbinary))))%8192 AS
 varchar))
```

- これにより、時間の経過とともに高速クエリに変換されるデータを取得するためにスキャンされるパーティションの合計数が最小限になります。

このパーティションスキーマからメリットを得るには、クライアント側でハッシュを計算し、クエリエンジンへの静的値 LiveAnalytics として の Timestream に渡す必要があることに注意してください。前の例では、生成されたハッシュが必要に応じてエンジンによって解決できることを検証する方法を示しています。

時系	host_name	ロケーション	server_type	cpu_usage	available_memory	cpu_temp
2022-09-07 21:48:44 .000 0	host-1235	us-east1	5.8xl	55	16.2	78
R2022-09-07 21:48:44 .000 0	host-3587	us-west1	5.8xl	62	18.1	81
2022-09-07 21:48:45. 000000000	host-258743	eu-central	5.8xl	88	9.4	91
2022-09-07 21:48:45 .000 0	host-35654	us-east2	5.8xl	29	24	54
R2022-09-07 21:48:45 .000 0	host-254	us-west1	5.8xl	44	32	48

当社の推奨事項`measure_name`に従って関連する を生成するには、取り込みパターンに依存する 2 つのパスがあります。

1. 履歴データのバッチ取り込みの場合 — バッチプロセスに独自のコードを使用する場合は、書き込みコードに変換を追加できます。

   上記の例の上に構築します。

```
        List<String> hosts = new ArrayList<>();

        hosts.add("host-1235");
        hosts.add("host-3587");
        hosts.add("host-258743");
        hosts.add("host-35654");
        hosts.add("host-254");

        for (String h: hosts){
            ByteBuffer buf2 = ByteBuffer.wrap(h.getBytes());
            partition = abs(hasher.hash(buf2, 0L)) % 8192;
            System.out.println(h + " - " + partition);

        }
```

   出力

```
host-1235 - 6445
host-3587 - 6399
host-258743 - 640
host-35654 - 2093
host-254 - 7051
```

   結果データセット

時系	host_name	ロケーション	measure_name	server_type	cpu_usage	available_memory	cpu_temp
2022-09-07 21:48:44 .00	host-1235	us-east1	6445	5.8xl	55	16.2	78

時系	host_name	ロケーション	measure_name	server_type	cpu_usage	available_memory	cpu_temp
R2022-09-07 21:48:44 .00	host-3587	us-west1	6399	5.8xl	62	18.1	81
2022-09-07 21:48:45.000000000	host-258743	eu-central	640	5.8xl	88	9.4	91
2022-09-07 21:48:45 .00	host-35654	us-east2	2093	5.8xl	29	24	54
R2022-09-07 21:48:45 .00	host-254	us-west1	7051	5.8xl	44	32	48

2. リアルタイム取り込みの場合 — データが入るときにmeasure_name、実行中の を生成する必要があります。

どちらの場合も、両端 (取り込みとクエリ) でハッシュ生成アルゴリズムをテストして、同じ結果が得られることを確認することをお勧めします。

に基づいてハッシュ化された値を生成するためのコード例をいくつか示しますhost_name。

Example Python

```
>>> import xxhash
>>> from bitstring import BitArray
>>> b=xxhash.xxh64('HOST-ID-1235').digest()
>>> BitArray(b).int % 8192
### 3195
```

Example Go

```
package main

import (
    "bytes"
    "fmt"
    "github.com/cespare/xxhash"
)

func main() {
    buf := bytes.NewBufferString("HOST-ID-1235")
    x := xxhash.New()
    x.Write(buf.Bytes())
    // convert unsigned integer to signed integer before taking mod
    fmt.Printf("%f\n", abs(int64(x.Sum64())) % 8192)
}

func abs(x int64) int64 {
    if (x < 0) {
        return -x
    }
    return x
}
```

Example Java

```
import java.nio.ByteBuffer;

import net.jpountz.xxhash.XXHash64;

public class test {
    public static void main(String[] args) {
        XXHash64 hasher = net.jpountz.xxhash.XXHashFactory.fastestInstance().hash64();

        String host = "HOST-ID-1235";
        ByteBuffer buf = ByteBuffer.wrap(host.getBytes());

        Long result = Math.abs(hasher.hash(buf, 0L));
        Long partition = result % 8192;

        System.out.println(result);
        System.out.println(partition);
```

```
    }
}
```

Example Maven の依存関係

```
        <dependency>
            <groupId>net.jpountz.lz4</groupId>
            <artifactId>lz4</artifactId>
            <version>1.3.0</version>
        </dependency>
```

セキュリティ

- の Timestream に継続的にアクセスするには LiveAnalytics、暗号化キーが保護され、取り消されたりアクセスできなくなったりしていないことを確認します。
- からAPIアクセスログをモニタリングします AWS CloudTrail。権限のないユーザーからの異常なアクセスパターンを監査して取り消す。
- で説明されている追加のガイドラインに従ってくださいの Amazon Timestream のセキュリティのベストプラクティス LiveAnalytics。

の Amazon Timestream の設定 LiveAnalytics

データ処理、ストレージ、クエリパフォーマンス、コスト要件に合わせて、メモリストアとマグネティックストアのデータ保持期間を設定します。

- 遅延受信データを処理するためのアプリケーションの要件に合わせて、メモリストアのデータ保持を設定します。遅延受信データとは、現在の時刻より前のタイムスタンプを持つ受信データです。これは、 の Timestream にデータを送信する前にイベントをバッチ処理するリソースと LiveAnalytics、断続的にオンラインになっている IoT センサーなどの断続的な接続を持つリソースから出力されます。
- 遅延受信データがメモリストアの保持よりも早くタイムスタンプで到着することが予想される場合は、テーブルのマグネティックストア書き込みを有効にする必要があります。テーブル EnableMagneticStoreWrites MagneticStoreWritesProperties に を設定すると、テーブルはメモリストアの保持期間より前のタイムスタンプを持つデータを、磁気ストアの保持期間内に受け入れます。
- クエリのタイプ、頻度、時間範囲、パフォーマンス要件 LiveAnalytics など、Timestream で実行する予定のクエリの特性を考慮してください。これは、メモリストアとマグネティックストアが異な

るシナリオに合わせて最適化されているためです。メモリストアは、 の Timestream に送信された最近の少量のデータを処理する高速 point-in-timeクエリ用に最適化されています LiveAnalytics。マグネティックストアは、 の Timestream に送信される中規模から大量のデータを処理する高速分析クエリ用に最適化されています LiveAnalytics。

- データ保持期間は、システムのコスト要件にも影響されます。

  例えば、アプリケーションの遅延受信データしきい値が 2 時間で、アプリケーションが 1 日分、1 週間分、または 1 か月分分分のデータを処理する多くのクエリを送信するシナリオを考えてみましょう。その場合、高速分析クエリ用にマグネティックストアが最適化されているため、メモリストアの保持期間を短く (2～3 時間) 設定し、より多くのデータをマグネティックストアに流すことができます。

メモリストアと既存のテーブルのマグネティックストアのデータ保持期間を増減した場合の影響を理解します。

- メモリストアの保持期間を短くすると、データはメモリストアからマグネティックストアに移動され、このデータ転送は永続的です。の Timestream LiveAnalytics は、メモリストアに入力するためにマグネティックストアからデータを取得しません。マグネティックストアの保持期間を短くすると、データはシステムから削除され、データの削除は永続的です。
- メモリストアまたはマグネティックストアの保持期間を延長すると、その時点 LiveAnalytics からの Timestream に送信されるデータに変更が有効になります。の Timestream LiveAnalytics は、メモリストアに入力するために磁気ストアからデータを取得しません。例えば、メモリストアの保持期間が最初に 2 時間に設定され、その後 24 時間に延長された場合、メモリストアに 24 時間分のデータが含まれるまで 22 時間かかります。

書き込み

- 受信データのタイムスタンプが、メモリストア用に設定されたデータ保持より前でなく、 で定義されている将来の取り込み期間より前であることを確認しますクォータ。これらの範囲外のタイムスタンプを持つデータを送信すると、テーブルのマグネティックストア書き込みを有効に LiveAnalytics しない限り、 の Timestream によってデータが拒否されます。マグネティックストアの書き込みを有効にする場合は、受信データのタイムスタンプがマグネティックストアに設定されたデータ保持よりも前になっていないことを確認してください。
- データの到着が遅れると予想される場合は、テーブルのマグネティックストア書き込みをオンにします。これにより、メモリストアの保持期間を過ぎてもマグネティックストアの保持期間内であるタイムスタンプを持つデータの取り込みが可能になります。これ

は、MagneticStoreWritesPropertiesテーブルの のEnableMagneticStoreWritesフラグを更新することで設定できます。このプロパティはデフォルトで false です。マグネティックストアへの書き込みは、すぐにクエリできないことに注意してください。6 時間以内に利用可能になります。

- 取り込まれたデータのタイムスタンプがメモリストアの保持境界内に収まるようにすることで、高スループットワークロードをメモリストアにターゲットにします。マグネティックストアへの書き込みは、データベースの同時取り込みを受信できるアクティブなマグネティックストアパーティションの最大数に制限されます。このActiveMagneticStorePartitionsメトリクスは、 で確認できます CloudWatch。アクティブなマグネティックストアパーティションを減らすには、マグネティックストアの取り込みのために同時に取り込むシリーズ数と時間を減らすことを目指します。
- の Timestream にデータを送信するときに LiveAnalytics、1 回のリクエストで複数のレコードをバッチ処理して、データ取り込みのパフォーマンスを最適化します。
  - 同じ時系列のレコードと、同じメジャー名を持つレコードを一括処理することをお勧めします。
  - リクエストが で定義されているサービス制限内であれば、1 つのリクエストでできるだけ多くのレコードをバッチ処理しますクォータ。
  - データ転送と取り込みのコストを削減するには、可能な限り一般的な属性を使用します。詳細については、 WriteRecords API「」を参照してください。
- の Timestream へのデータの書き込み中にクライアント側の部分的な障害が発生した場合は LiveAnalytics、拒否の原因に対処した後、取り込みに失敗したレコードのバッチを再送信できます。
- タイムスタンプで順序付けされたデータの方が書き込みパフォーマンスが向上します。
- の Amazon Timestream LiveAnalytics は、アプリケーションのニーズに合わせて自動的にスケーリングするように設計されています。Timestream for LiveAnalytics Notice がアプリケーションからの書き込みリクエストで急増すると、アプリケーションにある程度の初期メモリストアスロットリングが発生する可能性があります。アプリケーションでメモリストアのスロットリングが発生した場合は、同じ (または増加した) レート LiveAnalytics で の Timestream にデータを送信し続け、 LiveAnalytics がアプリケーションのニーズに合わせて自動的にスケーリングできるようにします。マグネティックストアのスロットリングが表示された場合は、マグネティックストアの取り込み速度を下げて、その回数ActiveMagneticStorePartitionsが減るまで減らす必要があります。

バッチロード

バッチロードのベストプラクティスについては、「」を参照してくださいバッチロードのベストプラクティス。

クエリ

Amazon Timestream for を使用したクエリのベストプラクティスを次に示します LiveAnalytics。

- クエリに不可欠なメジャー名とディメンション名のみを含めます。外部列を追加すると、データスキャンが増加し、クエリのパフォーマンスに影響します。
- 本番環境にクエリをデプロイする前に、クエリインサイトを確認して、空間的および時間的なプルーニングが最適であることを確認することをお勧めします。詳細については、「クエリインサイトを使用して Amazon Timestream のクエリを最適化する」を参照してください。
- 可能であれば、データ計算を Timestream にプッシュして、必要に応じて SELECT句と WHERE句の組み込みの集計関数とスカラー関数 LiveAnalytics を使用して、クエリのパフォーマンスを向上させ、コストを削減します。「SELECT」および「集計関数」を参照してください。
- 可能であれば、おおよその関数を使用します。例えば、クエリのパフォーマンスを最適化し、クエリコストを削減するには、 COUNT (DISTINCT column_name) の代わりに APPROX_DISTINCT を使用します。「集計関数」を参照してください。
- CASE 式を使用して、同じテーブルから複数回選択するのではなく、複雑な集計を実行します。「CASE ステートメント」を参照してください。
- 可能であれば、クエリの WHERE句に時間範囲を含めます。これにより、クエリのパフォーマンスとコストが最適化されます。例えば、データセットに過去 1 時間のデータのみが必要な場合は、time > ago(1h) などの時間述語を含めます。「SELECT」および「間隔と期間」を参照してください。
- クエリがテーブル内のメジャーのサブセットにアクセスする場合、クエリの WHERE句にメジャー名を必ず含めます。
- 可能な場合は、クエリの WHERE句のディメンションとメジャーを比較するときに等価演算子を使用します。ディメンションとメジャー名の等価述語により、クエリのパフォーマンスが向上し、クエリコストが削減されます。
- 可能な限り、 WHERE句の関数を使用してコストを最適化しないでください。
- LIKE 句を複数回使用しないでください。文字列列で複数の値をフィルタリングする場合は、正規表現を使用します。「正規表現関数」を参照してください。
- クエリの GROUP BY 句に必要な列のみを使用してください。

- クエリ結果が特定の順序でなければならない場合は、最も外側のクエリの ORDER BY 句でその順序を明示的に指定します。クエリ結果に順序付けが必要ない場合は、ORDERBY 句を使用してクエリのパフォーマンスを向上させることは避けてください。
- クエリの最初の N 行のみが必要な場合は、 LIMIT句を使用します。
- ORDER BY 句を使用して上位 N 値または下位 N 値を調べる場合は、LIMIT句を使用してクエリコストを削減します。
- 返されたレスポンスのページ分割トークンを使用して、クエリ結果を取得します。詳細については、「クエリ」を参照してください。
- クエリの実行を開始し、クエリが探している結果を返さないことに気付いた場合は、クエリをキャンセルしてコストを削減します。詳細については、「」を参照してくださいCancelQuery。
- アプリケーションにスロットリングが発生した場合は、Amazon Timestream が LiveAnalyticsアプリケーションのクエリスループットニーズを満たすように LiveAnalytics 自動スケーリングできるように、Amazon Timestream に同じレートでデータを送信し続けます。
- アプリケーションのクエリ同時実行要件が の Timestream のデフォルト制限を超える場合は LiveAnalytics、 AWS Support に連絡して制限を引き上げてください。

スケジュールされたクエリ

スケジュールされたクエリは、フリート全体の集計統計を事前に計算してダッシュボードを最適化するのに役立ちます。したがって、自然な質問は、ユースケースをどのように取り上げ、どの結果を事前計算するか、派生テーブルに保存されているこれらの結果を使用してダッシュボードを作成する方法を特定することです。このプロセスの最初のステップは、事前計算するパネルを特定することです。以下は、大まかなガイドラインです。

- パネルの入力に使用されるクエリによってスキャンされたバイト数、ダッシュボードの再ロードの頻度、およびこれらのダッシュボードをロードする同時ユーザーの数を考慮してください。最も頻繁にロードされるダッシュボードから始め、大量のデータをスキャンする必要があります。集計ダッシュボードの例の最初の 2 つのダッシュボードとドリルダウン例の集計ダッシュボードは、このようなダッシュボードの良い例です。
- どの計算が繰り返し使用されているかを検討します。すべてのパネルとパネルで使用されるすべての変数値に対してスケジュールされたクエリを作成できますが、1 つの計算を使用して複数のパネルに必要なデータを事前に計算する方法を探すことで、コストとスケジュールされたクエリの数を大幅に最適化できます。
- スケジュールされたクエリの頻度を考慮して、派生テーブルのマテリアライズド結果を更新します。ダッシュボードが更新される頻度と、ダッシュボードでクエリされる時間枠と、計算前および

ダッシュボードのパネルで使用されるビニングの時間を比較します。例えば、過去数日間の時間集計をプロットしているダッシュボードが数時間に 1 回しか更新されない場合は、30 分または 1 時間に 1 回だけ更新するようにスケジュールされたクエリを設定することができます。一方、1 分あたりの集計をプロットし、1 分ごとに更新されるダッシュボードがある場合は、スケジュールされたクエリで 1 分または数分ごとに結果を更新する必要があります。

- スケジュールされたクエリを使用して、どのクエリパターンをさらに最適化できるか (クエリコストとクエリレイテンシーの両方の観点から) を検討します。例えば、ダッシュボードで変数として頻繁に使用される一意のディメンション値を計算したり、センサーから最後に出力されたデータポイントや、特定の日付以降にセンサーから出力された最初のデータポイントを返したりする場合などです。これらのパターンの例の一部については、このガイドで説明します。

上記の考慮事項は、ダッシュボードを移動して派生テーブルをクエリする場合、ダッシュボード内のデータの鮮度、およびスケジュールされたクエリによって発生するコストに大きな影響を与えます。

クライアントアプリケーションとサポートされている統合

の Timestream と同じリージョンからクライアントアプリケーションを実行して LiveAnalytics 、ネットワークのレイテンシーとデータ転送コストを削減します。他の サービスの操作の詳細については、「」を参照してください他の サービスでの使用。以下は、その他の役立つリンクです。

- を使用した AWS 開発のベストプラクティス AWS SDK for Java
- AWS Lambda 関数の使用に関するベストプラクティス
- Amazon Managed Service for Apache Flink のベストプラクティス
- Grafana でダッシュボードを作成するためのベストプラクティス

全般

- に Timestream を使用する場合は、AWS 「 Well-Architected フレームワーク」に従ってください LiveAnalytics。このホワイトペーパーでは、オペレーショナルエクセレンス、セキュリティ、信頼性、パフォーマンス効率、コスト最適化のベストプラクティスに関するガイダンスを提供します。

計測とコスト最適化

の Amazon Timestream では LiveAnalytics、使用した分に対してのみ料金が発生します。クエリによってスキャンされた書き込み、保存データ、およびデータについて、 LiveAnalytics メーターの

タイムストリームを個別に指定します。各計測ディメンションの料金は、料金ページ で指定します。Amazon Timestream for LiveAnalytics Pricing Calculator を使用して、毎月の請求額を見積もることができます。

このセクションでは、 の Timestream での書き込み、ストレージ、クエリの計測方法について説明します LiveAnalytics。シナリオ例と計算例も示されています。さらに、コスト最適化のベストプラクティスのリストも含まれています。トピックは以下で選択できます。

トピック

- 書き込み
- ストレージ
- クエリ
- コスト最適化
- Amazon によるモニタリング CloudWatch

書き込み

各時系列イベントの書き込みサイズは、タイムスタンプのサイズと、1 つ以上のディメンション名、ディメンション値、メジャー名、およびメジャー値の合計として計算されます。タイムスタンプのサイズは 8 バイトです。ディメンション名、ディメンション値、およびメジャー名のサイズは、各ディメンション名、ディメンション値、およびメジャー名を表す文字列の UTF-8 エンコードされたバイトの長さです。測定値のサイズは、データ型によって異なります。これは、ブールデータ型では 1 バイト、gigint と double では 8 バイト、文字列では UTF-8 エンコードされたバイトの長さです。各書き込みは 1 KiB の単位でカウントされます。

2 つの計算例を以下に示します。

トピック

- 時系列イベントの書き込みサイズの計算
- 書き込み数の計算

時系列イベントの書き込みサイズの計算

以下に示すように、EC2インスタンスのCPU使用率を表す時系列イベントを考えてみましょう。

時間	region	az	vpc	ホスト名	measure_name	measure_value::double
1602983435238563000	us-east-1	1 日	vpc-1a2b3c4d	host-24Gju	cpu_utilization	35.0

時系列イベントの書き込みサイズは次のように計算できます。

- 時間 = 8 バイト
- 最初のディメンション = 15 バイト (`region` + `us-east-1`)
- 2 番目のディメンション = 4 バイト (`az` + `1d`)
- 3 番目のディメンション = 15 バイト (`vpc` + `vpc-1a2b3c4d`)
- 4 番目のディメンション = 18 バイト (`hostname` + `host-24Gju`)
- メジャーの名前 = 15 バイト (`cpu_utilization`)
- メジャーの値 = 8 バイト

時系列イベントの書き込みサイズ = 83 バイト

書き込み数の計算

次に、「」で説明されているEC2インスタンスと同様に時系列イベントの書き込みサイズの計算、5 秒ごとにメトリクスを出力する 100 個のインスタンスを考えてみましょう。EC2 インスタンスの月間書き込みの合計は、書き込みあたりの時系列イベントの数と、時系列イベントのバッチ処理中に一般的な属性が使用されているかによって異なります。次のシナリオごとに、合計月間書き込みを計算する例を示します。

トピック

- 書き込みごとに 1 つの時系列イベント
- 書き込みでの時系列イベントのバッチ処理
- 時系列イベントのバッチ処理と書き込みでの共通属性の使用

書き込みごとに 1 つの時系列イベント

各書き込みに 1 つの時系列イベントのみが含まれている場合、合計月間書き込みは次のように計算されます。

- 100 回の時系列イベント = 5 秒ごとに 100 回の書き込み
- x 12 書き込み/分 = 1,200 書き込み
- x 60 分/時 = 72,000 書き込み
- x 24 時間/日 = 1,728,000 書き込み
- x 30 日/月 = 51,840,000 書き込み

月間書き込みの合計 = 51,840,000

書き込みでの時系列イベントのバッチ処理

各書き込みは 1 KB の単位で測定されるため、書き込みには 12 の時系列イベントのバッチ (998 バイト) を含めることができ、合計月間書き込みは次のように計算されます。

- 100 回の時系列イベント = 5 秒ごとに 9 回の書き込み (1 回の書き込みあたり 12 回の時系列イベント)
- x 12 書き込み/分 = 108 書き込み
- x 60 分/時 = 6,480 回の書き込み
- x 24 時間/日 = 155,520 回の書き込み
- x 30 日/月 = 4,665,600 書き込み

月間書き込みの合計 = 4,665,600

時系列イベントのバッチ処理と書き込みでの共通属性の使用

リージョン、az、vpc、およびメジャー名が 100 個のEC2インスタンスで共通である場合、共通値は書き込みごとに 1 回だけ指定でき、共通属性と呼ばれます。この場合、一般的な属性のサイズは 52 バイト、時系列イベントのサイズは 27 バイトです。各書き込みは 1 KiB の単位で測定されるため、書き込みには 36 の時系列イベントと一般的な属性を含めることができ、合計月間書き込みは次のように計算されます。

- 100 回の時系列イベント = 5 秒ごとに 3 回の書き込み (1 回の書き込みあたり 36 回の時系列イベント)

- x 12 書き込み/分 = 36 書き込み
- x 60 分/時 = 2,160 書き込み
- x 24 時間/日 = 51,840 回の書き込み
- x 30 日/月 = 1,555,200 書き込み

月間書き込みの合計 = 1,555,200

> **Note**
>
> バッチ処理の使用、一般的な属性、1KB 単位への書き込みの四捨五入により、時系列イベントのストレージサイズは書き込みサイズとは異なる場合があります。

ストレージ

メモリストアとマグネティックストアの各時系列イベントのストレージサイズは、タイムスタンプ、ディメンション名、ディメンション値、メジャー名、およびメジャー値のサイズの合計として計算されます。タイムスタンプのサイズは 8 バイトです。ディメンション名、ディメンション値、およびメジャー名のサイズは、ディメンション名、ディメンション値、およびメジャー名を表す各文字列の UTF-8 エンコードされたバイトの長さです。測定値のサイズは、データ型によって異なります。これは、ブール型データ型の場合は 1 バイト、ビギント型とダブル型の場合は 8 バイト、文字列の場合は UTF-8 エンコードされたバイトの長さです。各メジャーは、 の Amazon Timestream に個別のレコードとして保存されます。つまり LiveAnalytics、時系列イベントに 4 つのメジャーがある場合、その時系列イベントには 4 つのレコードがストレージに保存されます。

EC2 インスタンスのCPU使用率を表す時系列イベントの例 (「」を参照[時系列イベントの書き込みサイズの計算](#)) を考慮すると、時系列イベントのストレージサイズは次のように計算されます。

- 時間 = 8 バイト
- 最初のディメンション = 15 バイト (`region` + `us-east-1`)
- 2 番目のディメンション = 4 バイト (`az` + `1d`)
- 3 番目のディメンション = 15 バイト (`vpc` + `vpc-1a2b3c4d`)
- 4 番目のディメンション = 18 バイト (`hostname` + `host-24Gju`)
- メジャーの名前 = 15 バイト (`cpu_utilization`)
- メジャーの値 = 8 バイト

時系列イベントのストレージサイズ = 83 バイト

> **Note**
>
> メモリストアは GB 時間で計測され、マグネティックストアは GB 月で計測されます。

クエリ

クエリは、Amazon Timestream の料金ページで指定されているように、アプリケーションで使用されている Timestream コンピューティングユニット (TCUs) の期間に基づいて TCU時間単位で課金されます。Amazon Timestream for LiveAnalytics' クエリエンジンは、クエリの処理中に無関係なデータを削除します。時間範囲、測定名、ディメンション名などの射影や述語を含むクエリにより、クエリ処理エンジンは大量のデータをプルーニングし、クエリコストを削減できます。

コスト最適化

書き込み、ストレージ、クエリのコストを最適化するには、 の Amazon Timestream で次のベストプラクティスを使用します LiveAnalytics。

- 書き込みリクエストの数を減らすために、書き込みごとに複数の時系列イベントをバッチ処理します。
- マルチメジャーレコードを使用することを検討してください。これにより、1 回の書き込みリクエストで複数の時系列メジャーを書き込み、データをよりコンパクトな方法で保存できます。これにより、書き込みリクエストの数だけでなく、データストレージコストとクエリコストも削減されます。
- 書き込みリクエストの数をさらに減らすには、バッチ処理で一般的な属性を使用して、書き込みごとにより多くの時系列イベントをバッチ処理します。
- 遅延受信データを処理するためのアプリケーションの要件に合わせて、メモリストアのデータ保持を設定します。遅延受信データとは、現在の時刻より前のタイムスタンプがあり、メモリストアの保持期間外の受信データです。
- マグネティックストアのデータ保持を、長期的なデータストレージ要件に合わせて設定します。
- クエリを作成するときは、クエリに不可欠なメジャー名とディメンション名のみを含めます。無関係な列を追加すると、データスキャンが増加し、クエリコストも増加します。クエリインサイトを確認して、含まれているディメンションとメジャーのプルーニング効率を評価することをお勧めします。

- 可能であれば、クエリの WHERE句に時間範囲を含めます。例えば、データセットに過去 1 時間のデータのみが必要な場合は、 などのタイム述語を含めますtime > ago(1h)。
- クエリがテーブル内のメジャーのサブセットにアクセスする場合、クエリの WHERE句に必ずメジャー名を含めます。
- クエリの実行を開始し、クエリが探している結果を返さないことに気付いた場合は、クエリをキャンセルしてコストを節約します。

Amazon によるモニタリング CloudWatch

Amazon LiveAnalytics を使用して Timestream をモニタリングできます。Amazon は CloudWatch、Timestream LiveAnalyticsから未加工のデータを収集して、 near-real-time読み取り可能なメトリクスに処理します。これらの統計は 2 週間記録されるため、履歴情報にアクセスしてウェブアプリケーションまたはサービスのパフォーマンスをより的確に把握できます。デフォルトでは、 LiveAnalytics メトリクスデータの Timestream は 1 分または 15 分の期間 CloudWatch で自動的に に送信されます。詳細については、「Amazon ユーザーガイド」の「Amazon CloudWatchとは」を参照してください。 CloudWatch

トピック

- LiveAnalytics メトリクスに Timestream を使用する方法を教えてください。
- LiveAnalytics メトリクスとディメンションのタイムストリーム
- の Timestream をモニタリングする CloudWatch アラームの作成 LiveAnalytics

LiveAnalytics メトリクスに Timestream を使用する方法を教えてください。

Timestream によってレポートされるメトリクスは、さまざまな方法で分析できる情報 LiveAnalytics を提供します。以下のリストは、メトリクスの一般的な利用方法をいくつか示しています。ここで紹介するのは開始するための提案事項です。すべてを網羅しているわけではありません。

目的	関連するメトリクス
How can I determine if any system errors occurred?	SystemErrors をモニタリングすれば、サーバーエラーコードを発生させたリクエストがあるかどうかを判断できます。通常、このメトリクスはゼロであるべきです。そうでない場合は、調査することをお勧めします。

目的	関連するメトリクス
How can I monitor the amount of data in the memory store?	指定された期間MemoryCumulativeBytesMetered にわたってモニタリングして、メモリストアに保存されているデータ量をバイト単位でモニタリングできます。このメトリクスは 1 時間ごとに出力され、アカウントおよびデータベースの粒度に保存されているバイトを追跡できます。メモリストアは GB 時間 (1 時間分の 1GB のデータを保存するコスト) で計測されます。したがって、 の時間単位の値を リージョンの GB 時間料金MemoryCumulativeBytesMetered に乗算すると、1 時間あたりのコストが発生します。 ディメンション: オペレーション (ストレージ) DatabaseName、メトリクス名
How can I monitor the amount of data in the magnetic store?	指定された期間MagneticCumulativeBytesMetered にわたってモニタリングして、マグネティックストアに保存されているデータ量をバイト単位でモニタリングできます。このメトリクスは 1 時間ごとに出力され、アカウントおよびデータベースの粒度に保存されているバイトを追跡できます。メモリストアは GB 月単位で計測されます (1 か月間に 1GB のデータを保存するコスト) 。したがって、 の時間単位の値を リージョンの GB 月単位の料金MagneticCumulativeBytesMetered に乗算すると、1 時間あたりのコストが発生します。例えば、 の値が MagneticCumulativeBytesMetered 107374182400 バイト (100GB) の場合、マグネティックストアの 1GB のデータの時間あたりの料金 = (0.03) (us-east-1 料金) / (30.4*24)。この値を GB MagneticCumulativeBytesMetered の に乗算すると、その時間に対して約 $0.004 が与えられます。 ディメンション: オペレーション (ストレージ) DatabaseName、メトリクス名

目的	関連するメトリクス
How can I monitor the data scanned by queries?	指定された期間CumulativeBytesMetered にわたってモニタリングして、 の Timestream に送信されたクエリ (バイト単位) によってスキャンされたデータをモニタリングできます LiveAnalytics。このメトリクスはクエリの実行後に出力され、アカウントとデータベースの粒度でスキャンされたデータを追跡できます。特定の期間のクエリコストは、メトリクスの値を、リージョンのスキャンされた GB あたりの料金に乗算することで計算できます。スケジュールされたクエリによってスキャンされたバイトは、このメトリクスで考慮されます。 ディメンション: オペレーション (クエリ) DatabaseName、メトリクス名
How can I monitor the data scanned by scheduled queries?	指定された期間CumulativeBytesMetered にわたってモニタリングし、 の Timestream によって実行されるスケジュールされたクエリ (バイト単位) によってスキャンされたデータをモニタリングできます LiveAnalytics。このメトリクスはクエリの実行後に出力され、アカウントとデータベースの粒度でスキャンされたデータを追跡できます。特定の期間のクエリコストは、メトリクスの値を、リージョンのスキャンされた GB あたりの料金に乗算することで計算できます。 **Note** 計測されたバイトは、クエリ でも考慮されますCumulativeBytesMetered 。 ディメンション: オペレーション (TriggeredScheduledQuery) DatabaseName、メトリクス名

目的	関連するメトリクス
How can I monitor the number of records ingested?	指定された期間NumberOfRecords にわたってモニタリングして、取り込まれたレコードの数をモニタリングできます。アカウントとデータベースの粒度に保存されているバイトを追跡できます。このメトリクスを使用して、クエリ結果が別のテーブルに書き込まれるときにスケジュールされたクエリによって行われた書き込みを監視することもできます。 を使用する場合WriteRecords API、メトリクスはWriteRecords リクエストごとに出力され、 CloudWatch オペレーションディメンションは になりますWriteRecords 。BatchLoad または を使用するとScheduled Query APIs、タスクが完了するまで、サービスによって決定される間隔でメトリクスが出力されます。このメトリクスの CloudWatch オペレーションディメンションはScheduled Query 、使用する BatchLoad または のいずれかAPIです。 ディメンション: オペレーション (WriteRecords、 BatchLoad、または ScheduledQuery ） DatabaseName、メトリクス名

目的	関連するメトリクス
How can I monitor the cost of records ingested?	コストが発生する取り込みバイト数CumulativeBytesMetered をモニタリングできます。アカウントとデータベースの粒度に保存されているバイトを追跡できます。取り込まれたレコードは累積バイトで計測されます。の値をリージョンの書き込み料金CumulativeBytesMetered で乗算すると、発生した取り込みコストが発生します。 を使用する場合WriteRecords API、このメトリクスはWriteRecords リクエストごとに出力され CloudWatch 、オペレーションディメンションは になりますWriteRecords 。BatchLoad または を使用する場合ScheduledQuery API、メトリクスは、タスクが完了するまで、サービスによって決定される間隔で出力されます。このメトリクスの CloudWatch オペレーションディメンションはAPI、使用する BatchLoad または ScheduledQuery によって異なります。 ディメンション: オペレーション (WriteRecords、 BatchLoad、または ScheduledQuery ） DatabaseName、メトリクス名
How can I monitor the Timestream Compute Units (TCUs) used in my account?	指定された期間QueryTCUにわたってモニタリングし、アカウントのクエリワークロードに消費されるコンピューティングユニットをモニタリングできます。このメトリクスは、アカウントからのアクティブなクエリワークロード中、1 分ごとに最大および最小のコンピューティングユニットで出力されます。 単位: Count 有効な統計: 最小、最大 メトリクス: ResourceCount ディメンション: Service: Timestream 、Resource: QueryTCU、Type: Resource、 Class: OnDemand

LiveAnalytics メトリクスとディメンションのタイムストリーム

の Timestream を操作すると LiveAnalytics、以下のメトリクスとディメンションが Amazon に送信されます CloudWatch。メトリクス値はすべて、1 分ごとに集計されて報告されます。の Timestream のメトリクスを表示するには、次の手順を使用します LiveAnalytics。

CloudWatch コンソールを使用してメトリクスを表示するには

メトリクスはまずサービスの名前空間ごとにグループ化され、次に各名前空間内のさまざまなディメンションの組み合わせごとにグループ化されます。

1. で CloudWatch コンソールを開きますhttps://console.aws.amazon.com/cloudwatch/。
2. 必要に応じて、リージョンを変更します。ナビゲーションバーで、AWSリソースが存在するリージョンを選択します。詳細については、「AWS サービスエンドポイント」を参照してください。
3. ナビゲーションペインで [Metrics] (メトリクス) を選択します。
4. [All metrics] (すべてのメトリクス) タブで、`AWS/Timestream for LiveAnalytics.` を選択します。

を使用してメトリクスを表示するには AWS CLI

- コマンドプロンプトで、次のコマンドを使用します。

```
aws cloudwatch list-metrics --namespace "AWS/Timestream"
```

LiveAnalytics メトリクスの Timestream のディメンション

の Timestream のメトリクス LiveAnalytics は、アカウント、テーブル名、またはオペレーションの値によって修飾されます。 CloudWatch コンソールを使用して、次の表の任意のディメンションに沿った LiveAnalytics データの Timestream を取得できます。

ディメンション	説明
`DatabaseName`	このディメンションは、 LiveAnalytics データベースの特定の Timestream にデータを制限します。この値は、現在のリージョ

ディメンション	説明
	ンと現在の AWS アカウントの任意のデータベースにすることができます。
Operation	このディメンションは、、、、 BatchLoad などの LiveAnalytics オペレーションの Timestream Storage WriteRecords のいずれかにデータを制限しますScheduledQuery 。使用可能な値のリストについては、 LiveAnalytics 「クエリAPIリファレンスの Timestream」を参照してください。
TableName	このディメンションは、 LiveAnalyticss データベースの Timestream 内の特定のテーブルにデータを制限します。

Important

CumulativeBytesMetered、UserErrors、および SystemErrorsメトリクスには Operationディメンションのみがあります。 SuccessfulRequestLatencyメトリクスには、常に Operation ディメンションがありますが、 の値によっては DatabaseNameおよび TableNameディメンションも含まれる場合がありますOperation。これは、テーブルレベルのオペレーションの LiveAnalytics Timestream にはディメンションTableNameとして DatabaseNameと がありますが、アカウントレベルのオペレーションには がないためです。

LiveAnalytics メトリクスのタイムストリーム

Note

Amazon は、 LiveAnalytics メトリクスについて以下の Timestream をすべて 1 分間隔で CloudWatch 集計します。

一般的なメトリクス

メトリクス	説明
SuccessfulRequestLatency	指定された LiveAnalytics 期間中に の Timestream へのリクエストが成功しました。SuccessfulRequestLatency は、次の 2 種類の情報を提供します。 • 成功したリクエストの経過時間 (最小、最大、合計、または平均) 。 • 成功したリクエストの数 (SampleCount)。 SuccessfulRequestLatency は、 の Timestream 内のアクティビティのみを反映し LiveAnalytics 、ネットワークレイテンシーやクライアント側のアクティビティは考慮しません。 単位: Milliseconds ディメンション • DatabaseName • TableName • Operation 有効な統計: • Minimum • Maximum • Average • SampleCount • P10 • p50 • p90 • p95

メトリクス	説明
	• p99

書き込みメトリクスとストレージメトリクス

メトリクス	説明
MagneticStoreRejectedRecordCount	非同期的に拒否されたマグネティックストア書き込みレコードの数。これは、新しいレコードのバージョンが現在のバージョンよりも小さい場合、または新しいレコードのバージョンが現在のバージョンと等しいがデータが異なる場合に発生する可能性があります。 単位: Count ディメンション • DatabaseName • TableName • Operation 有効な統計: • Sum • SampleCount
MagneticStoreRejectedUpload UserFailures	ユーザーエラーが原因でアップロードされなかった磁気ストア拒否レコードレポートの数。これは、IAMアクセス許可が正しく設定されていないか、削除された S3 バケットが原因である可能性があります。 単位: Count

メトリクス	説明
	ディメンション • DatabaseName • TableName • Operation 有効な統計: • Sum • SampleCount
MagneticStoreRejectedUpload SystemFailures	システムエラーが原因でアップロードされなかった磁気ストア拒否レコードレポートの数。 単位: Count ディメンション • DatabaseName • TableName • Operation 有効な統計: • Sum • SampleCount

メトリクス	説明
ActiveMagneticStorePartitions	特定の時間にデータをアクティブに取り込むマグネティックストアパーティションの数。 単位: Count ディメンション • DatabaseName • Operation 有効な統計: • Sum • SampleCount

メトリクス	説明
MagneticStorePendingRecordsLatency	クエリに使用できないマグネティックストアへの最も古い書き込み。マグネティックストアに書き込まれたレコードは、6 時間以内にクエリできます。 単位: Milliseconds ディメンション • DatabaseName • TableName • Operation 有効な統計: • Minimum • Maximum • Average • SampleCount • P10 • p50 • p90 • p95 • p99
MemoryCumulativeBytesMetered	メモリストアに保存されているデータ量、バイト単位 単位: Bytes ディメンション: Operation 有効な統計: • Average

メトリクス	説明
MagneticCumulativeBytesMetered	マグネティックストアに保存されるデータ量、バイト単位 単位: Bytes ディメンション: Operation 有効な統計: • Average
CumulativeBytesMetered	の Timestream への取り込みによって計測されたデータ量 LiveAnalytics。バイト単位。 単位: Bytes ディメンション: Operation 有効な統計:Sum
NumberOfRecords	の Timestream に取り込まれたレコードの数 LiveAnalytics。 単位: Count ディメンション: Operation 有効な統計:Sum

クエリメトリクス

メトリクス	説明
CumulativeBytesMetered	の Timestream に送信されたクエリによってスキャンされたデータの量 LiveAnalytics。バイト単位。 単位: Bytes

メトリクス	説明
	ディメンション: Operation 有効な統計: • Sum
ResourceCount	アカウントのクエリワークロードに消費される Timestream コンピューティングユニット (TCUs) 。このメトリクスは、アカウントからのアクティブなクエリワークロード中、1 分ごとに最大および最小のコンピューティングユニットで出力されます。 単位: Count 有効な統計: 最小、最大 ディメンション: Service: Timestream 、Resource: QueryTCU、Type: Resource、 Class: OnDemand

エラーメトリクス

メトリクス	説明
SystemErrors	が指定された SystemError 期間中に LiveAnalytics を生成する Timestream へのリクエスト。SystemError は通常、内部サービスエラーを示します。 単位: Count ディメンション: Operation 有効な統計: • Sum

メトリクス	説明
	• SampleCount
UserErrors	指定された期間中に InvalidRequest エラー LiveAnalytics を生成する の Timestream へのリクエスト。 InvalidRequest は通常、パラメータの無効な組み合わせ、存在しないテーブルの更新の試み、不正なリクエスト署名など、クライアント側のエラーを示します。 UserErrors は、現在の AWS リージョンと現在の AWS アカウントの無効なリクエストの集計を表します。 単位: Count ディメンション: Operation 有効な統計: • Sum • SampleCount

> **Important**
>
> Average や Sum など、すべての統計が必ずしも常にすべてのメトリクスに適用可能であるとは限りません。ただし、これらの値はすべて、 LiveAnalytics コンソールの場合は Timestream を通じて、コンソールの場合は CloudWatch コンソールを使用して AWS CLI、AWS SDKsすべてのメトリクスで使用できます。

の Timestream をモニタリングする CloudWatch アラームの作成 LiveAnalytics

CloudWatch アラームの状態 LiveAnalytics が変わったときに Amazon Simple Notification Service (Amazon SNS) メッセージを送信する Timestream の Amazon アラームを作成できます。指定した期間中、1 つのアラームが 1 つのメトリクスを監視します。このアラームは、複数の期間にわたる一定のしきい値とメトリクスの値の関係性に基づき、1 つ以上のアクションを実行します。アクションは、Amazon SNSトピックまたは Auto Scaling ポリシーに送信される通知です。

アラームは、持続状態の変更に対してのみアクションを呼び出します。 CloudWatch アラームは、単に特定の状態にあるという理由だけではアクションを呼び出しません。状態が変わって、変わった状態が指定期間にわたって維持される必要があります。

CloudWatch アラームの作成の詳細については、「Amazon ユーザーガイド」の「Amazon CloudWatch アラームの使用」を参照してください。 CloudWatch

トラブルシューティング

このセクションでは、 の Timestream のトラブルシューティングについて説明します LiveAnalytics。

トピック

- WriteRecords スロットルの処理
- 拒否されたレコードの処理
- の Timestream UNLOADからのトラブルシューティング LiveAnalytics
- LiveAnalytics 特定のエラーコードのタイムストリーム

WriteRecords スロットルの処理

Timestream へのメモリストアの書き込みリクエストは、アプリケーションのデータ取り込みのニーズに合わせて Timestream がスケールするとスロットリングされる場合があります。アプリケーションでスロットリングの例外が発生した場合は、Timestream がアプリケーションのニーズに自動的にスケーリングできるように、同じ (またはそれ以上の) スループットでデータを送信し続ける必要があります。

Timestream へのマグネティックストアの書き込みリクエストは、取り込みを受信するマグネティックストアパーティションの最大制限が設定されている場合、スロットリングされる可能性があります。このデータベースの `ActiveMagneticStorePartitions` Cloudwatch メトリクスを確認するように指示するスロットルメッセージが表示されます。このスロットルの解決には最大 6 時間かかる場合があります。このスロットルを回避するには、高スループットの取り込みワークロードにメモリストアを使用する必要があります。マグネティックストア取り込みの場合、取り込むシリーズの数と期間を制限することで、より少ないパーティションへの取り込みをターゲットにできます。

データ取り込みのベストプラクティスの詳細については、「」を参照してください書き込み。

拒否されたレコードの処理

Timestream がレコードを拒否すると、拒否の詳細`RejectedRecordsException`を含む を受け取ります。 WriteRecords レスポンスからこの情報を抽出する方法の詳細については、書き込み失敗の処理を参照してください。

新しいレコードのバージョンが既存のレコードのバージョン 以下のマグネティックストアの更新を除き、すべての拒否がこのレスポンスに含まれます。この場合、Timestream は上位バージョンの既存のレコードを更新しません。Timestream は、以前のバージョンまたは同等のバージョンの新しいレコードを拒否し、これらのエラーを S3 バケットに非同期的に書き込みます。これらの非同期エラーレポートを受信するには、 テーブル`MagneticStoreWriteProperties`の`MagneticStoreRejectedDataLocation`プロパティを設定する必要があります。

の Timestream UNLOADからのトラブルシューティング LiveAnalytics

UNLOAD コマンドに関連するトラブルシューティングのガイダンスを次に示します。

カテゴリ	エラーメッセージ	トラブルシューティング方法
S3 キーの長さ	UNLOAD 送信先で指定された S3 プレフィックス [%s] を使用する場合の結果ファイルキーは、S3 で許可されているキーの長さを超えています。詳細については、ドキュメントを参照してください。	UNLOAD ステートメントを使用してクエリ結果をエクスポートする場合、S3 バケット名とプレフィックスの長さの合計で構成される S3 キー長は、サポートされている最大 S3 キー長を超えています。S3 プレフィックスまたはバケット名の長さを短くすることをお勧めします。
	UNLOAD partitioned_by [%s] を使用する場合の結果ファイルキーは、S3 で許可されているキー長を超えています。詳細については、ドキュメントを参照してください。	UNLOAD ステートメントを使用してクエリ結果をエクスポートする場合、partitioned_by 列を使用する S3 キーの長さが、サポートされている最大 S3 キーの長さ を超えています。代替列でパーティション分割するか、pa

カテゴリ	エラーメッセージ	トラブルシューティング方法
		rtitioned_column の長さを短くすることをお勧めします (可能な場合) 。
	UNLOAD S3 プレフィックス [%s] と partitioned_by [%s] を使用する場合の結果ファイルキーは、S3 で許可されているキー長を超えています。詳細については、ドキュメントを参照してください。	UNLOAD ステートメントを使用してクエリ結果をエクスポートする場合、S3 バケット名、プレフィックス、および partitioned_by 列名の長さの合計で構成される S3 キーの長さが、サポートされている最大 S3 キー長 を超えています。プレフィックス、バケット名の長さを減らすか、代替列を使用してデータをパーティション化することをお勧めします。
	生成された S3 オブジェクトキー: %s が長すぎます。詳細については、ドキュメントを参照してください。	UNLOAD ステートメントを使用してクエリを処理する際、パーティション化された列の値の 1 つが、サポートされている最大 S3 キー長 を超えています。パーティション列と値は、生成されたオブジェクトキーにあります。

カテゴリ	エラーメッセージ	トラブルシューティング方法
S3 スロットル	Amazon S3 がUNLOADコマンドからの書き込みをスロットリングしていることが検出されました。詳細については、「Amazon Timestream ドキュメント」を参照してください。	S3 のドキュメントについては、https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html「」を参照してください。複数のリーダー/ライターが同じフォルダにアクセスすると、S3 APIコールレートをスロットリングできます。指定されたバケットへの呼び出しボリュームを監査してください。複数の同時UNLOADクエリに同じバケットを使用している場合は、同じ に異なるバケットを使用してください。の Timestream 以外の複数のオペレーションに同じバケットを使用している場合は LiveAnalytics UNLOAD、UNLOAD結果を別のバケットに移動することを検討してください。

LiveAnalytics 特定のエラーコードのタイムストリーム

このセクションでは、 の Timestream の特定のエラーコードについて説明します LiveAnalytics。

LiveAnalytics 書き込みAPIエラーのタイムストリーム

InternalServerException

HTTP ステータスコード: 500

ThrottlingException

HTTP ステータスコード: 429

ValidationException

HTTP ステータスコード: 400

ConflictException

HTTP ステータスコード: 409

AccessDeniedException

このアクションを実行する十分なアクセス権限がありません。

HTTP ステータスコード: 403

ServiceQuotaExceededException

HTTP ステータスコード: 402

ResourceNotFoundException

HTTP ステータスコード: 404

RejectedRecordsException

HTTP ステータスコード: 419

InvalidEndpointException

HTTP ステータスコード: 421

LiveAnalytics クエリAPIエラーのタイムストリーム

ValidationException

HTTP ステータスコード: 400

QueryExecutionException

HTTP ステータスコード: 400

ConflictException

HTTP ステータスコード: 409

ThrottlingException

HTTP ステータスコード: 429

InternalServerException

HTTP ステータスコード: 500

InvalidEndpointException

HTTP ステータスコード: 421

クォータ

このトピックでは、 の Amazon Timestream 内で制限とも呼ばれる現在のクォータについて説明します LiveAnalytics。各クォータは、指定がない限り、リージョン単位で適用されます。

トピック

- デフォルトのクォータ
- サービス制限
- サポートされているデータ型
- バッチロード
- 命名に関する制約
- 予約済みキーワード
- システム識別子
- UNLOAD

デフォルトのクォータ

次の表に、 LiveAnalytics クォータの Timestream とデフォルト値を示します。

displayName	説明	defaultValue
アカウントあたりのデータベース	ごとに作成できるデータベースの最大数 AWS アカウント。	500
アカウントあたりのテーブル数	ごとに作成できるテーブルの最大数 AWS アカウント。	50000

displayName	説明	defaultValue
のスロットルレート CRUD APIs	現在のリージョンで、アカウントごとに 1 秒あたりに実行できるCreate/Update/List/Describe/Delete database/table/scheduledクエリAPIリクエストの最大数。	1
アカウントごとのスケジュールされたクエリ	ごとに作成できるスケジュールされたクエリの最大数 AWS アカウント。	10000
アクティブなマグネティックストアパーティションの最大数	データベースあたりのアクティブなマグネティックストアパーティションの最大数。パーティションは、取り込みを受信してから最大 6 時間アクティブのままになる場合があります。	250
maxQueryTCU	アカウントに対して設定TCUsできる最大クエリ。	1,000

サービス制限

次の表に、 LiveAnalytics サービス制限の Timestream とデフォルト値を示します。コンソールからテーブルのデータ保持を編集するには、「テーブルの編集」を参照してください。

displayName	説明	defaultValue
将来の取り込み期間 (分)	現在のシステム時間と比較した時系列データの最大リードタイム (分単位)。例えば、将来の取り込み期間が 15 分の場合、 LiveAnalytics の Timestream は、現在のシステ	15

displayName	説明	defaultValue
	ム時間より最大 15 分前のデータを受け入れます。	
メモリストアの最小保持期間 (時間)	テーブルごとにデータをメモリストアに保持する必要がある最小期間 (時間単位)。	1
メモリストアの最大保持期間 (時間)	テーブルごとにデータをメモリストアに保持できる最大期間 (時間単位)。	8766
マグネティックストアの最小保持期間 (日)	テーブルごとにデータをマグネティックストアに保持する必要がある最小期間 (日数)。	1
マグネティックストアの最大保持期間 (日)	マグネティックストアにデータを保持できる最大期間 (日数)。この値は 200 年と同等です。	73000
マグネティックストアのデフォルトの保持期間	テーブルごとにデータがマグネティックストアに保持されるデフォルト値 (日数) 。この値は 200 年と同等です。	73000
メモリストアのデフォルトの保持期間	メモリストアにデータが保持されるデフォルトの期間 (時間単位) 。	6
テーブルあたりのディメンション	テーブルあたりのディメンションの最大数。	128
テーブルあたりのメジャー名	テーブルあたりの一意のメジャー名の最大数。	8192

displayName	説明	defaultValue
シリーズあたりのディメンション名とディメンション値のペアのサイズ	シリーズあたりのディメンション名とディメンション値の最大サイズ。	2 KB
レコードの最大サイズ	レコードの最大サイズ。	2 KB
リクエストあたりのWriteRecords APIレコード数	リクエスト内の WriteRecords APIレコードの最大数。	100
ディメンション名の長さ	ディメンション名の最大バイト数。	60 バイト
メジャー名の長さ	メジャー名の最大バイト数。	256 バイト
データベース名の長さ	データベース名の最大バイト数。	256 バイト
テーブル名の長さ	テーブル名の最大バイト数。	256 バイト
QueryString KiB の長さ	クエリ文字列の最大長 (KiB 単位) 。クエリのエンコード文字数は UTF-8 文字です。	256
クエリの実行期間 (時間)	クエリの最大実行期間 (時間単位)。時間がかかるクエリはタイムアウトします。	1
クエリインサイト	現在のリージョンで、アカウントごとに 1 秒あたりにクエリインサイトを有効にして実行できるクエリAPIリクエストの最大数。	1
クエリ結果のメタデータのサイズ	クエリ結果の最大メタデータサイズ。	100 KB

displayName	説明	defaultValue
クエリ結果のデータサイズ	クエリ結果の最大データサイズ。	5 GB
複数メジャーレコードあたりのメジャー	複数メジャーレコードあたりのメジャーの最大数。	256
複数メジャーレコードあたりのメジャー値サイズ	複数メジャーレコードあたりのメジャー値の最大サイズ。	2048
テーブルごとの複数メジャーレコードにわたる一意のメジャー	1 つのテーブルに定義されているすべての複数メジャーレコードの一意のメジャー。	1024
アカウントあたりのタイムストリームコンピューティングユニット (TCUs)	アカウントTCUsあたりのデフォルトの最大値。	200

サポートされているデータ型

次の表は、測定値とディメンション値でサポートされているデータ型を示しています。

説明	LiveAnalytics 値のタイムストリーム
測定値でサポートされているデータ型。	Big int、ダブル、文字列、ブール値、MULTI、タイムスタンプ
ディメンション値でサポートされているデータ型。	文字列

バッチロード

バッチロード内の制限とも呼ばれる現在のクォータは次のとおりです。

説明	LiveAnalytics 値のタイムストリーム
最大バッチロードタスクサイズ	最大バッチロードタスクサイズは 100 GB を超えることはできません。
ファイル数	バッチロードタスクには 100 個を超えるファイルを含めることはできません。
最大ファイルサイズ	バッチロードタスクの最大ファイルサイズは 5 GB を超えることはできません。
CSV ファイル行サイズ	CSV ファイル内の行は 16 MB を超えることはできません。これは、引き上げることができないハードな制限です。
アクティブなバッチロードタスク	テーブルには 5 つ以上のアクティブなバッチロードタスクを含めることはできません。また、アカウントには 10 を超えるアクティブなバッチロードタスクを含めることはできません。の Timestream LiveAnalytics は、より多くのリソースが利用可能になるまで、新しいバッチロードタスクをスロットリングします。

命名に関する制約

次の表に、命名上の制約を示します。

説明	LiveAnalytics 値のタイムストリーム
ディメンション名の最大長。	60 バイト
メジャー名の最大長。	256 バイト
テーブル名またはデータベース名の最大長。	256 バイト
テーブル名とデータベース名	• は使用しないでくださいシステム識別子。 • a ~ z A ~ Z 0 ~ 9 _ (アンダースコア) - (ダッシュ) . (ドット) を含めることができます。

説明	LiveAnalytics 値のタイムストリーム
	• すべての名前は UTF-8 でエンコードする必要があり、大文字と小文字は区別されます。 **Note** テーブル名とデータベース名は、UTF-8 バイナリ表現を使用して比較されます。つまり、ASCII文字の比較では大文字と小文字が区別されます。
メジャー名	• システム識別子 またはコロン ':' を含めることはできません。 • 予約済みプレフィックス (ts_、) で開始することはできませんmeasure_value 。 **Note** テーブル名とデータベース名は、UTF-8 バイナリ表現を使用して比較されます。つまり、ASCII文字の比較では大文字と小文字が区別されます。
属性名	• システム識別子、コロン ':'、または二重引用符 (「) を含めることはできません。 • 予約済みプレフィックス (ts_、) で開始することはできませんmeasure_value 。 • ここにリストされている Unicode 文字 [0,31]、「\u2028」、「」「u2029」を含めることはできません。 **Note** ディメンション名とメジャー名は、UTF-8 バイナリ表現を使用して比較されます。つまり、ASCII文字の比較では大文字と小文字が区別されます。

説明	LiveAnalytics 値のタイムストリーム
すべての列名	列名を複製することはできません。マルチメジャーレコードはディメンションとメジャーを列として表すため、ディメンションの名前をメジャーの名前と同じにすることはできません。名前では、大文字と小文字が区別されます。

予約済みキーワード

以下のキーワードはすべて予約されています。

- ALTER
- AND
- AS
- BETWEEN
- BY
- CASE
- CAST
- CONSTRAINT
- CREATE
- CROSS
- CUBE
- CURRENT_DATE
- CURRENT_TIME
- CURRENT_TIMESTAMP
- CURRENT_USER
- DEALLOCATE
- DELETE
- DESCRIBE
- DISTINCT
- DROP
- ELSE

- END
- ESCAPE
- EXCEPT
- EXECUTE
- EXISTS
- EXTRACT
- FALSE
- FOR
- FROM
- FULL
- GROUP
- GROUPING
- HAVING
- IN
- INNER
- INSERT
- INTERSECT
- INTO
- IS
- JOIN
- LEFT
- LIKE
- LOCALTIME
- LOCALTIMESTAMP
- NATURAL
- NORMALIZE
- NOT
- NULL
- ON

- または
- ORDER
- OUTER
- PREPARE
- RECURSIVE
- RIGHT
- ROLLUP
- SELECT
- TABLE
- THEN
- TRUE
- UESCAPE
- UNION
- UNNEST
- USING
- VALUES
- WHEN
- WHERE
- WITH

システム識別子

システム識別子の LiveAnalytics列名「measure_value」、「ts_non_existent_col」、「time」を Timestream に予約します。さらに、列名は「ts_」または「measure_name」で始まることはできません。システム識別子は大文字と小文字が区別されます。UTF-8 バイナリ表現を使用して比較された識別子。つまり、識別子の比較では大文字と小文字が区別されます。

> **Note**
>
> システム識別子をディメンション名またはメジャー名に使用することはできません。データベース名やテーブル名にシステム識別子を使用しないことをお勧めします。

UNLOAD

UNLOAD コマンドに関連する制限については、「 を使用して Timestream から S3 にクエリ結果をエクスポートUNLOADする」を参照してください。

クエリ言語リファレンス

> **Note**
>
> このクエリ言語リファレンスには、Apache ライセンス、バージョン 2.0 でライセンスされている Trino Software Foundation (旧 Presto Software Foundation) からの以下のサードパーティードキュメントが含まれています。このライセンスに準拠している場合を除き、このファイルは使用できません。Apache ライセンスのバージョン 2.0 のコピーを取得するには、Apache ウェブサイト を参照してください。

の Timestream LiveAnalytics は、データを操作するための豊富なクエリ言語をサポートします。使用可能なデータ型、演算子、関数、およびコンストラクトを以下に示します。

サンプルクエリ セクションで Timestream のクエリ言語をすぐに使用開始することもできます。

トピック

- サポートされているデータ型
- 組み込みの時系列機能
- SQL サポート
- 論理演算子
- 比較演算子
- 比較関数
- 条件式
- 変換関数
- 算術演算子
- 数学関数
- 文字列演算子
- 文字列関数
- 配列演算子

- 配列関数
- ビット単位関数
- 正規表現関数
- 日付/時刻演算子
- 日付/時刻関数
- 集計関数
- Window 関数
- サンプルクエリ

サポートされているデータ型

LiveAnalyticsのクエリ言語の Timestream では、次のデータ型がサポートされています。

> **Note**
> 書き込みでサポートされているデータ型については、「データ型」で説明されています。

データ型	説明
`int`	32 ビット整数を表します。
`bigint`	64 ビット符号付き整数を表します。
`boolean`	ロジックの 2 つの真理値の 1 つ、 `True` と `False`。
`double`	64 ビットの可変精度データ型を表します。IEEE バイナリ浮動小数点算術 に Standard 754 を実装します。 **Note** クエリ言語はデータを読み取るためのものです。クエリで使用できる `Infinity`と の 2 `NaN`重値関数があります。ただし、これらの値を Timestream に書き込むことはできません。

データ型	説明
varchar	最大サイズが 2KB の可変長文字データ。
array[*T*,...]	指定されたデータ型の 1 つ以上の要素を含む *T*次のとおりです。*T* は、Timestream でサポートされている任意のデータ型です。
row(*T*,...)	データ型の 1 つ以上の名前付きフィールドを含む *T*。 フィールドは Timestream でサポートされている任意のデータ型で、ドットフィールドリファレンス演算子を使用してアクセスできます。 `.`
date	形式で日付を表します*YYYY*-*MM*-*DD*。ここで *YYYY* は 年、*MM* は月であり、*DD* は、それぞれ 日です。サポートされている範囲は から 1970-01-01 です2262-04-11 。 例: `1971-02-03`
time	の時刻を表しますUTC。time データ型は*HH*.*MM*.*SS*.*sssssssss* .、Support nanosecond precision の形式で表されます。 例: `17:02:07.496000000`

データ型	説明
timestamp	でナノ秒の精度時間を使用して、インスタンスを時間単位で表しますUTC。 *YYYY*-*MM*-*DD* *hh*:*mm*:*ss*.*sssssssss* クエリは1677-09-21 00:12:44.000000000 、 ~ の範囲のタイムスタンプをサポートします2262-04-11 23:47:16.854775807 。

データ型	説明

interval	2 つの部分*X*`t`で構成される文字列リテラル として時間間隔を表します。*X* また、*t*.
	
	X は 以上の数値であり`0`、*t* は 2 番目や 1 時間などの時間単位です。ユニットは複数化されません。時間単位 *t* は、次のいずれかの文字列リテラルである必要があります。
	
	• `nanosecond`
	• `microsecond`
	• `millisecond`
	• `second`
	• `minute`
	• `hour`
	• `day`
	• `ns` (と同じ`nanosecond`)
	• `us` (と同じ`microsecond`)
	• `ms` (と同じ`millisecond`)
	• `s` (と同じ`second`)
	• `m` (と同じ`minute`)
	• `h` (と同じ`hour`)
	• `d` (と同じ`day`)
	
	例:
	
	`17s`
	
	`12second`
	
	`21hour`

データ型	説明
	`2d`
`timeseries[row(timestamp, T,...)]`	オブジェクトarrayで構成される時間間隔で記録されたメジャーの値を表しますrow。各 には、データ型の timestamp と 1 つ以上の測定値rowが含まれます。*T*次のとおりです。*T* は、bigint、、booleandouble、または のいずれかになりますvarchar。行は、 によって昇順にソートされますtimestamp 。時系列データ型は、時間の経過に伴うメジャーの値を表します。
unknown	null データを表します。

組み込みの時系列機能

の Timestream LiveAnalytics は、時系列データをファーストクラスの概念として扱う組み込みの時系列機能を提供します。

組み込みの時系列機能は、ビューと関数の 2 つのカテゴリに分割できます。

各コンストラクトについては、以下を参照してください。

トピック

- 時系列ビュー
- 時系列関数

時系列ビュー

の Timestream は、データをtimeseriesデータ型に変換するための次の関数 LiveAnalytics をサポートしています。

トピック

- CREATE_TIME_SERIES
- UNNEST

CREATE_TIME_SERIES

CREATE_TIME_SERIES は、時系列のすべての raw 測定値 (時刻と測定値) を取得し、時系列データ型を返す集計関数です。この関数の構文は次のとおりです。

```
CREATE_TIME_SERIES(time, measure_value::<data_type>)
```

ここで、 <data_type>は測定値のデータ型であり、gigint、boolean、double、または varchar のいずれかになります。2 番目のパラメータを null にすることはできません。

次に示すように、メトリクスという名前のテーブルに保存されているEC2インスタンスのCPU使用率を検討します。

時間	region	az	vpc	instance_id	measure_name	measure_value::double
2019-12-04 19:00:00.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	cpu_utilization	35.0
2019-12-04 19:00:01.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	cpu_utilization	38.2
2019-12-04 19:00:02.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	cpu_utilization	45.3
2019-12-04 19:00:00.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef1	cpu_utilization	54.1
2019-12-04 19:00:01.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef1	cpu_utilization	42.5

時間	region	az	vpc	instance_id	measure_name	measure_value::double
2019-12-04 19:00:02.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef1	cpu_utilization	33.7

クエリの実行：

```
SELECT region, az, vpc, instance_id, CREATE_TIME_SERIES(time, measure_value::double) as
 cpu_utilization FROM metrics
    WHERE measure_name='cpu_utilization'
    GROUP BY region, az, vpc, instance_id
```

は、 をメジャー値cpu_utilizationとして持つすべてのシリーズを返します。この場合、2 つのシリーズがあります。

region	az	vpc	instance_id	cpu_utilization
us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	[{time: 2019-12-04 19:00:00.000000000, measure_value::double: 35.0}, {time: 2019-12-04 19:00:01.000000000, measure_value::double: 38.2}, {time: 2019-12-04 19:00:02.000000000, measure_v

region	az	vpc	instance_id	cpu_utilization
				alue::double: 45.3}]
us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567 890abcdef1	[{time: 2019-12-0 4 19:00:00. 000000000 , measure_v alue::double: 35.1}, {time: 2019-12-0 4 19:00:01. 000000000 , measure_v alue::double: 38.5}, {time: 2019-12-0 4 19:00:02. 000000000 , measure_v alue::double: 45.7}]

UNNEST

UNNEST は、timeseriesデータをフラットモデルに変換できるテーブル関数です。構文は次のとおりです。

UNNEST は、 timeseriesを timeと の 2 つの列に変換しますvalue。エイリアスは、以下 UNNESTに示すように でも使用できます。

```
UNNEST(timeseries) AS <alias_name> (time_alias, value_alias)
```

ここで、 <alias_name>はフラットテーブルのエイリアス、 time_alias はtime列のエイリアス、 value_aliasはvalue列のエイリアスです。

例えば、フリート内の一部のEC2インスタンスが 5 秒間隔でメトリクスを出力するように設定されていて、他のインスタンスが 15 秒間隔でメトリクスを出力し、過去 6 時間の 10 秒の粒度ですべてのインスタンスの平均メトリクスが必要であるシナリオを考えてみましょう。このデータを取得するには、CREATE_TIME_SERIES を使用してメトリクスを時系列モデルに変換します。その後、INTERPOLATE_LINEAR を使用して 10 秒の粒度で欠落値を取得できます。次に、 を使用してデータをフラットモデルに変換しUNNEST、 AVG を使用してすべてのインスタンスの平均メトリクスを取得します。

```
WITH interpolated_timeseries AS (
    SELECT region, az, vpc, instance_id,
        INTERPOLATE_LINEAR(
            CREATE_TIME_SERIES(time, measure_value::double),
                SEQUENCE(ago(6h), now(), 10s)) AS interpolated_cpu_utilization
    FROM timestreamdb.metrics
    WHERE measure_name= ‘cpu_utilization’ AND time >= ago(6h)
    GROUP BY region, az, vpc, instance_id
)
SELECT region, az, vpc, instance_id, avg(t.cpu_util)
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_cpu_utilization) AS t (time, cpu_util)
GROUP BY region, az, vpc, instance_id
```

上記のクエリは、 をエイリアスUNNESTで使用する方法を示しています。以下は、 にエイリアスを使用せずに同じクエリの例ですUNNEST。

```
WITH interpolated_timeseries AS (
    SELECT region, az, vpc, instance_id,
        INTERPOLATE_LINEAR(
            CREATE_TIME_SERIES(time, measure_value::double),
                SEQUENCE(ago(6h), now(), 10s)) AS interpolated_cpu_utilization
    FROM timestreamdb.metrics
    WHERE measure_name= ‘cpu_utilization’ AND time >= ago(6h)
    GROUP BY region, az, vpc, instance_id
)
SELECT region, az, vpc, instance_id, avg(value)
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_cpu_utilization)
GROUP BY region, az, vpc, instance_id
```

時系列関数

Amazon Timestream for LiveAnalytics は、派生、積分、相関などの時系列関数をサポートし、時系列データからより深いインサイトを取得します。このセクションでは、これらの各関数の使用情報とサンプルクエリについて説明します。詳細については、以下のトピックを選択してください。

トピック

補間関数

時系列データが特定の時点でイベントの値が欠落している場合は、補間を使用して欠落しているイベントの値を推定できます。Amazon Timestream は、線形補間、立方スプライン補間、最後の観測繰越 (locf) 補間、および一定の補間の 4 つの補間バリアントをサポートしています。このセクションでは、補間関数の LiveAnalytics Timestream の使用情報とサンプルクエリについて説明します。

使用状況の情報

機能	出力データ型	説明
`interpolate_linear (timeseries, array[timestamp])`	時系列	線形補間 を使用して欠落データを入力します。
`interpolate_linear (timeseries, timestamp)`	double	線形補間 を使用して欠落データを入力します。
`interpolate_spline _cubic(timeseries, array[timestamp])`	時系列	立方スプライン補間 を使用して欠落データを入力します。

機能	出力データ型	説明
interpolate_spline_cubic(timeseries, timestamp)	double	立方スプライン補間 を使用して欠落データを入力します。
interpolate_locf(timeseries, array[timestamp])	時系列	最後にサンプリングされた値を使用して欠落データを入力します。
interpolate_locf(timeseries, timestamp)	double	最後にサンプリングされた値を使用して欠落データを入力します。
interpolate_fill(timeseries, array[timestamp], double)	時系列	定数値を使用して欠落データを入力します。
interpolate_fill(timeseries, timestamp, double)	double	定数値を使用して欠落データを入力します。

クエリの例

Example

過去 2 時間の特定のEC2ホストの平均CPU使用率を 30 秒間隔でビニングし、線形補間を使用して欠落値を入力します。

```
WITH binned_timeseries AS (
SELECT hostname, BIN(time, 30s) AS binned_timestamp, ROUND(AVG(measure_value::double),
 2) AS avg_cpu_utilization
FROM "sampleDB".DevOps
WHERE measure_name = 'cpu_utilization'
    AND hostname = 'host-Hovjv'
    AND time > ago(2h)
GROUP BY hostname, BIN(time, 30s)
), interpolated_timeseries AS (
SELECT hostname,
    INTERPOLATE_LINEAR(
```

```
        CREATE_TIME_SERIES(binned_timestamp, avg_cpu_utilization),
            SEQUENCE(min(binned_timestamp), max(binned_timestamp), 15s)) AS
 interpolated_avg_cpu_utilization
FROM binned_timeseries
GROUP BY hostname
)
SELECT time, ROUND(value, 2) AS interpolated_cpu
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_avg_cpu_utilization)
```

Example

過去 2 時間における特定のEC2ホストの平均CPU使用率を 30 秒間隔でビニングし、最後に繰り越された観測値に基づいて補間を使用して欠落値を入力します。

```
WITH binned_timeseries AS (
SELECT hostname, BIN(time, 30s) AS binned_timestamp, ROUND(AVG(measure_value::double),
 2) AS avg_cpu_utilization
FROM "sampleDB".DevOps
WHERE measure_name = 'cpu_utilization'
    AND hostname = 'host-Hovjv'
    AND time > ago(2h)
GROUP BY hostname, BIN(time, 30s)
), interpolated_timeseries AS (
SELECT hostname,
    INTERPOLATE_LOCF(
        CREATE_TIME_SERIES(binned_timestamp, avg_cpu_utilization),
            SEQUENCE(min(binned_timestamp), max(binned_timestamp), 15s)) AS
 interpolated_avg_cpu_utilization
FROM binned_timeseries
GROUP BY hostname
)
SELECT time, ROUND(value, 2) AS interpolated_cpu
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_avg_cpu_utilization)
```

派生関数

派生は、特定のメトリクスの変更率を計算し、イベントにプロアクティブに応答するために使用できます。例えば、過去 5 分間のEC2インスタンスのCPU使用率の導関数を計算し、重要な正の導関数に気付いたとします。これはワークロードに対する需要の増加を示す可能性があるため、ワークロー

ドをより適切に処理するために、より多くのEC2インスタンスをスピンアップすることを決定できます。

Amazon Timestream は、2 つの派生関数のバリアントをサポートしています。このセクションでは、 LiveAnalytics 派生関数の Timestream の使用情報とサンプルクエリについて説明します。

使用状況の情報

機能	出力データ型	説明
`derivative_linear(timeseries, interval)`	時系列	指定された `timeseries` について、 の各ポイントの微分を計算します`interval`。
`non_negative_derivative_linear(timeseries, interval)`	時系列	と同じですが`derivative_linear(timeseries, interval)` 、正の値のみを返します。

クエリの例

Example

過去 1 時間の 5 分ごとのCPU使用率の変化率を見つけます。

```
SELECT DERIVATIVE_LINEAR(CREATE_TIME_SERIES(time, measure_value::double), 5m) AS
 result
FROM “sampleDB”.DevOps
WHERE measure_name = 'cpu_utilization'
AND hostname = 'host-Hovjv' and time > ago(1h)
GROUP BY hostname, measure_name
```

Example

1 つ以上のマイクロサービスによって生成されたエラーの増加率を計算します。

```
WITH binned_view as (
    SELECT bin(time, 5m) as binned_timestamp, ROUND(AVG(measure_value::double), 2) as
 value
    FROM “sampleDB”.DevOps
```

```
    WHERE micro_service = 'jwt'
    AND time > ago(1h)
    AND measure_name = 'service_error'
    GROUP BY bin(time, 5m)
)
SELECT non_negative_derivative_linear(CREATE_TIME_SERIES(binned_timestamp, value), 1m)
 as rateOfErrorIncrease
FROM binned_view
```

統合関数

整数を使用して、時系列イベントの時間単位あたりの曲線下面積を見つけることができます。例えば、アプリケーションが受信したリクエストのボリュームを単位時間ごとに追跡しているとします。このシナリオでは、積分関数を使用して、特定の期間に指定された間隔ごとに処理されたリクエストの合計ボリュームを決定できます。

Amazon Timestream は、統合関数の 1 つのバリアントをサポートしています。このセクションでは、 LiveAnalytics 統合関数の Timestream の使用情報とサンプルクエリについて説明します。

使用状況の情報

機能	出力データ型	説明

`integral_trapezoidal(timeseries(double))`
 	
`integral_trapezoidal(timeseries(double), interval day to second)`
 	
`integral_trapezoidal(timeseries(bigint))`
 	
`integral_trapezoidal(timeseries(bigint), interval day to second)`
	double	台形ルール を使用して、`timeseries` 指定された に従って積分を概算`interval day to second`します。間隔の日単位から秒単位のパラメータはオプションで、デフォルトはです`1s`。間隔の詳細については、「」を参照してください間隔と期間。

機能	出力データ型	説明
integral_trapezoidal(timeseries(integer), interval day to second) integral_trapezoidal(timeseries(integer))		

クエリの例

Example

特定のホストによって過去 1 時間に 5 分ごとに処理されたリクエストの合計量を計算します。

```
SELECT INTEGRAL_TRAPEZOIDAL(CREATE_TIME_SERIES(time, measure_value::double), 5m) AS
 result FROM sample.DevOps
WHERE measure_name = 'request'
AND hostname = 'host-Hovjv'
AND time > ago (1h)
GROUP BY hostname, measure_name
```

相関関数

2 つの同様の長さ時系列がある場合、相関関数は相関係数を提供し、2 つの時系列の経時的な傾向について説明します。相関係数の範囲は -1.0から です1.0。 -1.0は、2 つの時系列が同じレートで反対方向に傾向していることを示します。 は、2 つの時系列が同じレートで同じ方向に傾向している1.0ことを示します。の値は、2 つの時系列間に相関関係がない0ことを示します。例えば、石油の価格が上昇し、石油会社の株価が上昇すると、石油の価格上昇の傾向と石油会社の価格上昇の傾向には正の相関係数があります。正の相関係数が高い場合は、2 つの料金が同様の割合で傾向を示すことになります。同様に、債券価格と債券イールドの相関係数は負であり、これら 2 つの値は時間の経過とともに逆方向に推移することを示しています。

Amazon Timestream は、相関関数の 2 つのバリアントをサポートしています。このセクションでは、 LiveAnalytics 相関関数の Timestream の使用情報とサンプルクエリについて説明します。

使用状況の情報

機能	出力データ型	説明
`correlate_pearson(timeseries, timeseries)`	double	2 つの のピアソンの相関係数を計算しますtimeseries 。時系列には同じタイムスタンプが必要です。
`correlate_spearman(timeseries, timeseries)`	double	2 つの のスピアマンの相関係数を計算しますtimeseries 。時系列には同じタイムスタンプが必要です。

クエリの例

Example

```
WITH cte_1 AS (
    SELECT INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(time, measure_value::double),
        SEQUENCE(min(time), max(time), 10m)) AS result
    FROM sample.DevOps
    WHERE measure_name = 'cpu_utilization'
    AND hostname = 'host-Hovjv' AND time > ago(1h)
    GROUP BY hostname, measure_name
),
cte_2 AS (
    SELECT INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(time, measure_value::double),
        SEQUENCE(min(time), max(time), 10m)) AS result
    FROM sample.DevOps
    WHERE measure_name = 'cpu_utilization'
    AND hostname = 'host-Hovjv' AND time > ago(1h)
    GROUP BY hostname, measure_name
)
SELECT correlate_pearson(cte_1.result, cte_2.result) AS result
FROM cte_1, cte_2
```

関数のフィルタリングと削減

Amazon Timestream は、時系列データに対してフィルターを実行し、オペレーションを減らすための関数をサポートしています。このセクションでは、 LiveAnalytics フィルターおよび削減関数の Timestream の使用情報とサンプルクエリについて説明します。

使用状況の情報

機能	出力データ型	説明
`filter(timeseries(T), function(T, Boolean))`	時系列 (T)	渡された が をfunction返す値を使用して、入力時系列から時系列を作成しますtrue。
`reduce(timeseries(T), initialState S, inputFunction(S, T, S), outputFunction(S, R))`	R	時系列から減算された単一の値を返します。inputFunction は、各要素で時系列で順番に呼び出されます。現在の 要素に加えて、 は現在の状態 (最初は initialState) inputFunction を取得し、新しい状態を返します。outputFunction が呼び出され、最終状態が結果値になります。は ID 関数にoutputFunction することができます。

クエリの例

Example

測定が 70 を超えるホストポイントとフィルターポイントのCPU使用率の時系列を作成します。

```
WITH time_series_view AS (
    SELECT INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(time, ROUND(measure_value::double,2)),
```

```
            SEQUENCE(ago(15m), ago(1m), 10s)) AS cpu_user
    FROM sample.DevOps
    WHERE hostname = 'host-Hovjv' and measure_name = 'cpu_utilization'
        AND time > ago(30m)
    GROUP BY hostname
)
SELECT FILTER(cpu_user, x -> x.value > 70.0) AS cpu_above_threshold
from time_series_view
```

Example

ホストのCPU使用率の時系列を作成し、測定値の合計二乗を求めます。

```
WITH time_series_view AS (
    SELECT INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(time, ROUND(measure_value::double,2)),
            SEQUENCE(ago(15m), ago(1m), 10s)) AS cpu_user
    FROM sample.DevOps
    WHERE hostname = 'host-Hovjv' and measure_name = 'cpu_utilization'
        AND time > ago(30m)
    GROUP BY hostname
)
SELECT REDUCE(cpu_user,
    DOUBLE '0.0',
    (s, x) -> x.value * x.value + s,
    s -> s)
from time_series_view
```

Example

ホストのCPU使用率の時系列を作成し、CPUしきい値を超えるサンプルの割合を決定します。

```
WITH time_series_view AS (
    SELECT INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(time, ROUND(measure_value::double,2)),
            SEQUENCE(ago(15m), ago(1m), 10s)) AS cpu_user
    FROM sample.DevOps
    WHERE hostname = 'host-Hovjv' and measure_name = 'cpu_utilization'
        AND time > ago(30m)
    GROUP BY hostname
)
SELECT ROUND(
```

```
    REDUCE(cpu_user,
      -- initial state
      CAST(ROW(0, 0) AS ROW(count_high BIGINT, count_total BIGINT)),
      -- function to count the total points and points above a certain threshold
      (s, x) -> CAST(ROW(s.count_high + IF(x.value > 70.0, 1, 0), s.count_total + 1) AS
 ROW(count_high BIGINT, count_total BIGINT)),
      -- output function converting the counts to fraction above threshold
      s -> IF(s.count_total = 0, NULL, CAST(s.count_high AS DOUBLE) / s.count_total)),
    4) AS fraction_cpu_above_threshold
from time_series_view
```

SQL サポート

の Timestream は、いくつかの一般的なSQLコンストラクト LiveAnalytics をサポートしています。詳細については、以下を参照してください。

トピック

- SELECT
- サブクエリのサポート
- SHOW ステートメント
- DESCRIBE ステートメント
- UNLOAD

SELECT

SELECT ステートメントを使用して、1 つ以上のテーブルからデータを取得できます。Timestream のクエリ言語は、 SELECTステートメントの次の構文をサポートしています。

```
[ WITH with_query [, ...] ]
            SELECT [ ALL | DISTINCT ] select_expr [, ...]
            [ function (expression) OVER (
            [ PARTITION BY partition_expr_list ]
            [ ORDER BY order_list ]
            [ frame_clause ] )
            [ FROM from_item [, ...] ]
            [ WHERE condition ]
            [ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
            [ HAVING condition]
            [ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
```

```
            [ ORDER BY order_list ]
            [ LIMIT [ count | ALL ] ]
```

この場合、次のようになります。

- function (expression) は、サポートされているウィンドウ関数の 1 つです。
- partition_expr_listは以下の通りです。

```
expression | column_name [, expr_list ]
```

- order_listは以下の通りです。

```
expression | column_name [ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ]
[, order_list ]
```

- frame_clauseは以下の通りです。

```
ROWS | RANGE
{ UNBOUNDED PRECEDING | expression PRECEDING | CURRENT ROW } |
{BETWEEN
{ UNBOUNDED PRECEDING | expression { PRECEDING | FOLLOWING } |
CURRENT ROW}
AND
{ UNBOUNDED FOLLOWING | expression { PRECEDING | FOLLOWING } |
CURRENT ROW }}
```

- from_item は次のいずれかです。

```
table_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
from_item join_type from_item [ ON join_condition | USING ( join_column [, ...] ) ]
```

- join_type は次のいずれかです。

```
[ INNER ] JOIN
LEFT [ OUTER ] JOIN
RIGHT [ OUTER ] JOIN
FULL [ OUTER ] JOIN
```

- grouping_element は次のいずれかです。

```
()
```

```
expression
```

サブクエリのサポート

Timestream は、 EXISTSおよびIN述語のサブクエリをサポートしています。EXISTS 述語は、サブクエリが行を返すかどうかを決定します。IN 述語は、サブクエリによって生成された値が IN 句のの値または式と一致するかどうかを決定します。Timestream クエリ言語は、相関クエリやその他のサブクエリをサポートしています。

```
SELECT t.c1
FROM (VALUES 1, 2, 3, 4, 5) AS t(c1)
WHERE EXISTS
(SELECT t.c2
 FROM (VALUES 1, 2, 3) AS t(c2)
 WHERE t.c1= t.c2
)
ORDER BY t.c1
```

c1
1
2
3

```
SELECT t.c1
FROM (VALUES 1, 2, 3, 4, 5) AS t(c1)
WHERE t.c1 IN
(SELECT t.c2
 FROM (VALUES 2, 3, 4) AS t(c2)
)
ORDER BY t.c1
```

c1
2

c1
3
4

SHOW ステートメント

SHOW DATABASES ステートメントを使用して、アカウント内のすべてのデータベースを表示できます。構文は次のとおりです。

```
SHOW DATABASES [LIKE pattern]
```

ここで、 LIKE句を使用してデータベース名をフィルタリングできます。

SHOW TABLES ステートメントを使用して、アカウントのすべてのテーブルを表示できます。構文は次のとおりです。

```
SHOW TABLES [FROM database] [LIKE pattern]
```

ここで、 FROM句を使用してデータベース名をフィルタリングし、 LIKE句を使用してテーブル名をフィルタリングできます。

SHOW MEASURES ステートメントを使用して、テーブルのすべての測定値を表示できます。構文は次のとおりです。

```
SHOW MEASURES FROM database.table [LIKE pattern]
```

ここで、 FROM句を使用してデータベースとテーブル名を指定し、 LIKE句を使用してメジャー名をフィルタリングできます。

DESCRIBE ステートメント

DESCRIBE ステートメントを使用して、テーブルのメタデータを表示できます。構文は次のとおりです。

```
DESCRIBE database.table
```

にテーブル名tableが含まれています。describe ステートメントは、テーブルの列名とデータ型を返します。

UNLOAD

の Timestream は、SQLサポートの拡張機能として UNLOAD コマンド LiveAnalytics をサポートします。でサポートされているデータ型UNLOADについては、「」を参照してくださいサポートされているデータ型。time および unknownタイプは には適用されませんUNLOAD。

```
UNLOAD (SELECT statement)
 TO 's3://bucket-name/folder'
 WITH ( option = expression [, ...] )
```

オプションが の場合

```
{ partitioned_by = ARRAY[ col_name[,…] ]
 | format = [ '{ CSV | PARQUET }' ]
 | compression = [ '{ GZIP | NONE }' ]
 | encryption = [ '{ SSE_KMS | SSE_S3 }' ]
 | kms_key = '<string>'
 | field_delimiter ='<character>'
 | escaped_by = '<character>'
 | include_header = ['{true, false}']
 | max_file_size = '<value>'
}
```

SELECT ステートメント

LiveAnalytics テーブルの 1 つ以上の Timestream からデータを選択および取得するために使用されるクエリステートメント。

```
(SELECT column 1, column 2, column 3 from database.table
      where measure_name = "ABC" and timestamp between ago (1d) and now() )
```

TO 句

```
TO 's3://bucket-name/folder'
```

または

```
TO 's3://access-point-alias/folder'
```

UNLOAD ステートメントの TO句は、クエリ結果の出力先を指定します。の Timestream LiveAnalytics が出力ファイルオブジェクトを書き込む Amazon S3 上のフォルダの場所を持つ Amazon S3 access-point-alias バケット名または Amazon S3 を含む完全なパスを指定する必要があります。S3 バケットは、同じアカウントと同じリージョンで所有されている必要があります。クエリ結果セットに加えて、 Timestream for はマニフェストファイルとメタデータファイルを指定された宛先フォルダに LiveAnalytics 書き込みます。

PARTITIONED_BY 句

```
partitioned_by = ARRAY [col_name[,…] , (default: none)
```

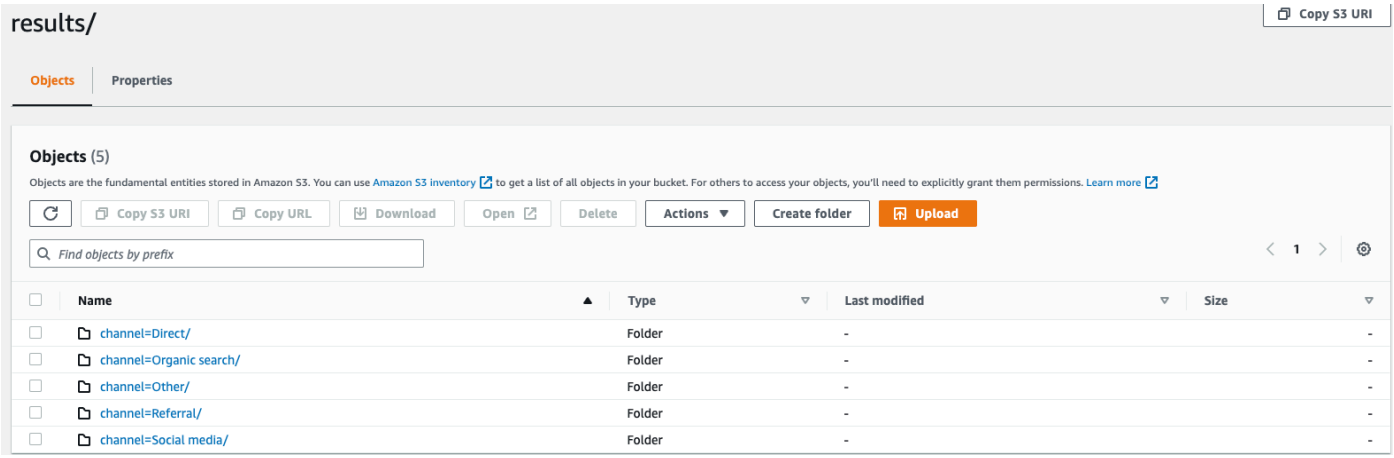
partitioned_by 句は、データをきめ細かなレベルでグループ化して分析するためのクエリに使用されます。クエリ結果を S3 バケットにエクスポートする場合、選択クエリの 1 つ以上の列に基づいてデータをパーティション化することを選択できます。データをパーティション化すると、エクスポートされたデータはパーティション列に基づいてサブセットに分割され、各サブセットは別々のフォルダに保存されます。エクスポートされたデータを含む結果フォルダ内に、サブフォルダfolder/results/partition column = partition value/が自動的に作成されます。ただし、パーティション化された列は出力ファイルに含まれないことに注意してください。

partitioned_by は構文の必須句ではありません。パーティショニングなしでデータを エクスポートする場合は、構文で 句を除外できます。

Example

ウェブサイトのクリックストリームデータをモニタリングしていて、トラフィックのチャネルが direct、、Social Media、Organic SearchOther、 の 5 つあるとしますReferral。データを エクスポートするときは、列 を使用してデータをパーティション化することを選択できますChannel。データフォルダ 内にはs3://bucketname/results、それぞれにチャネル名を持つ 5 つのフォルダがあります。例えば、このフォルダs3://bucketname/results/channel=Social Media/.内には、Social Mediaチャネルを介してウェブサイトに進出したすべての顧客のデータが表示されます。同様に、残りのチャネルには他のフォルダがあります。

Channel 列でパーティション分割されたエクスポートされたデータ



FORMAT

```
format = [ '{ CSV | PARQUET }' , default: CSV
```

S3 バケットに書き込まれたクエリ結果の形式を指定するキーワード。データは、デフォルトの区切り文字としてカンマ (,CSV) を使用してカンマ区切り値 () としてエクスポートするか、分析用の効率的なオープン列ストレージ形式である Apache Parquet 形式でエクスポートできます。

COMPRESSION

```
compression = [ '{ GZIP | NONE }' ], default: GZIP
```

エクスポートしたデータは、圧縮アルゴリズムを使用して圧縮GZIPすることも、 NONEオプションを指定して圧縮解除することもできます。

ENCRYPTION

```
encryption = [ '{ SSE_KMS | SSE_S3 }' ], default: SSE_S3
```

Amazon S3 の出力ファイルは、選択した暗号化オプションを使用して暗号化されます。データに加えて、マニフェストファイルとメタデータファイルは、選択した暗号化オプションに基づいて暗号化されます。現在、SSE_S3 および SSE_KMS 暗号化をサポートしています。SSE_S3 は、Amazon S3 が 256 ビットの高度な暗号化標準 (AES) 暗号化を使用してデータを暗号化するサーバー側の暗号化です。SSE_KMS は、カスタマーマネージドキーを使用してデータを暗号化するサーバー側の暗号化です。

KMS_KEY

```
kms_key = '<string>'
```

KMS キーは、エクスポートされたクエリ結果を暗号化するためのカスタマー定義のキーです。KMS キーは AWS Key Management Service (AWS KMS) によって安全に管理され、Amazon S3 のデータファイルの暗号化に使用されます。

FIELD_DELIMITER

```
field_delimiter ='<character>' , default: (,)
```

CSV データを形式でエクスポートする場合、このフィールドは、パイプASCII文字 (|)、カンマ (,)、タブ (/t) などの出力ファイルのフィールドを区切るために使用される 1 つの文字を指定します。CSV ファイルのデフォルトの区切り文字はカンマ文字です。データ内の値に選択した区切り文字が含まれている場合、区切り文字は引用符で囲まれます。例えば、データ内の値に が含まれている場合`Time,stream`、この値はエクスポートされたデータ`"Time,stream"`内の として引用符で囲まれます。Timestream が に使用する引用符文字 LiveAnalytics は、二重引用符 (「) です。

にヘッダーを含める`FIELD_DELIMITER`場合は、キャリッジリターン文字 (ASCII 13、16 進0D数 、テキスト '\r') または改行文字 (ASCII 10、16 進数 0A、テキスト '\n') を として指定しないでください。これによりCSV、多くのパーサーが結果のCSV出力でヘッダーを正しく解析できなくなるためです。

ESCAPED_BY

```
escaped_by = '<character>', default: (\)
```

データを CSV 形式でエクスポートする場合、このフィールドは S3 バケットに書き込まれたデータファイルでエスケープ文字として扱う文字を指定します。エスケープは、次のシナリオで発生します。

1. 値自体に引用符文字 (「) が含まれている場合、エスケープ文字を使用してエスケープされます。例えば、値が でTime"stream、 (\) が設定されたエスケープ文字である場合、エスケープは としてエスケープされます`Time\"stream`。
2. 値に設定されたエスケープ文字が含まれている場合、エスケープされます。例えば、値が の場合`Time\stream`、 としてエスケープされます`Time\\stream`。

> **Note**
>
> エクスポートされた出力に配列、行、時系列などの複雑なデータ型が含まれている場合、JSON文字列としてシリアル化されます。次に例を示します。

データ型	実際の値	値がエスケープされるCSV形式 [シリアル化されたJSON文字列〕
配列	`[ 23,24,25 ]`	`"[23,24,25]"`
行	`( x=23.0, y=hello )`	`"{\"x\":23.0,\"y\":\"hello\"}"`
時系列	`[ ( time=1970-01-01 00:00:00.000000010 , value=100.0 ), ( time=1970-01-01 00:00:00.000000012, value=120.0 ) ]`	`"[{\"time\":\"1970-01-01 00:00:00.000000010Z\",\"value\":100.0},{\"time\":\"1970-01-01 00:00:00.000000012Z\",\"value\":120.0}]"`

INCLUDE_HEADER

```
include_header = 'true' , default: 'false'
```

CSV データを形式でエクスポートする場合、このフィールドでは、エクスポートされたCSVデータファイルの最初の行として列名を含めることができます。

使用できる値は「true」と「false」で、デフォルト値は「false」です。escaped_by や などのテキスト変換オプションは、ヘッダーにもfield_delimiter適用されます。

> **Note**
>
> ヘッダーを含める場合、キャリッジリターン文字 (ASCII 13、16 進 0D、テキスト '\r') または改行文字 (ASCII 10、16 進 0A、テキスト '\n') を として選択しないことが重要です。これによりFIELD_DELIMITER、多くのパーサーが結果のCSV出力でヘッダーを正しく解析できなくなるためです。

MAX_FILE_SIZE

```
max_file_size = 'X[MB|GB]' , default: '78GB'
```

このフィールドは、UNLOADステートメントが Amazon S3 で作成するファイルの最大サイズを指定します。UNLOAD ステートメントは複数のファイルを作成できますが、Amazon S3 に書き込まれる各ファイルの最大サイズは、このフィールドで指定されたサイズとほぼ同じになります。

フィールドの値は 16 MB から 78 GB の間でなければなりません。などの整数、または 12GBなどの小数で指定できます0.5GB24.7MB。デフォルト値は 78 GB です。

実際のファイルサイズは、ファイルが書き込まれるときに概算されるため、実際の最大サイズは、指定した数と完全に等しくない場合があります。

論理演算子

の Timestream は、次の論理演算子 LiveAnalytics をサポートしています。

演算子	説明	例
AND	両方の値が true の場合は true	a AND b
または	いずれかの値が true の場合は true	a OR b
NOT	値が false の場合は true	NOT a

- AND 比較の結果は、式の一方または両側が NULLである場合に発生する可能性がありますNULL。
- AND 演算子の少なくとも 1 つの側が の場合FALSE、式は に評価されますFALSE。
- OR 比較の結果は、式の一方または両側が NULLである場合に発生する可能性がありますNULL。
- OR 演算子の少なくとも 1 つの側が の場合TRUE、式は に評価されますTRUE。
- の論理補数は NULLですNULL。

次の真理表は、 ANDおよび NULLでの の処理を示していますOR。

A	B	A と b	A または b
null	null	null	null
false	null	false	null
null	false	false	null
true	null	null	true
null	true	null	true
false	false	false	false
true	false	false	true
false	true	false	true
true	true	true	true

次の真理表は、 NULLでの の処理を示していますNOT。

A	ではない
null	null
true	false
false	true

比較演算子

の Timestream では、次の比較演算子 LiveAnalytics がサポートされています。

演算子	説明
<	Less than

演算子	説明
>	Greater than
<=	以下
>=	以上
=	Equal
<>	Not equal
!=	Not equal

Note

- BETWEEN 演算子は、値が指定された範囲内にあるかどうかをテストします。構文は次のとおりです。

  ```
  BETWEEN min AND max
  ```

  BETWEEN または NOT BETWEENステートメントNULLに が存在すると、 ステートメントはに評価されますNULL。
- IS NULL および IS NOT NULL 演算子は、値が null (未定義) であるかどうかをテストします。NULL で を使用すると true にIS NULL評価されます。
- ではSQL、NULL値は不明な値を示します。

比較関数

の Timestream は、次の比較関数 LiveAnalytics をサポートしています。

トピック

- greatest()
- 最小 ()
- ALL()、 ANY()、 SOME()

greatest()

greatest() 関数は、指定された値の最大を返します。指定された値のいずれかが NULLの場合、 が返されますNULL。構文は次のとおりです。

```
greatest(value1, value2, ..., valueN)
```

最小 ()

least() 関数は、指定された値の最小を返します。指定された値のいずれかが NULLの場合、 が返されますNULL。構文は次のとおりです。

```
least(value1, value2, ..., valueN)
```

ALL()、 ANY()、 SOME()

ALL、ANY、および SOME 量子は、次の方法で比較演算子とともに使用できます。

式	意味
A = ALL(...)	A がすべての値に等しい場合、true に評価されます。
<> ALL(...)	A が値と一致しない場合、true に評価されます。
A < ALL(...)	A が最小値より小さい場合、true に評価されます。
A = ANY(...)	A がいずれかの値に等しい場合、true に評価されます。
<> ANY(...)	A が 1 つ以上の値と一致しない場合、true に評価されます。
A < ANY(...)	A が最大値より小さい場合に true に評価されます。

例と使用上の注意

> **Note**
>
> 比較値がリテラルのリストである場合、ALL、、ANYまたは を使用する場合、SOMEキーワードVALUESを使用する必要があります。

例: **ANY()**

クエリステートメントANY()の の例を次に示します。

```
SELECT 11.7 = ANY (VALUES 12.0, 13.5, 11.7)
```

同じオペレーションの代替構文は次のとおりです。

```
SELECT 11.7 = ANY (SELECT 12.0 UNION ALL SELECT 13.5 UNION ALL SELECT 11.7)
```

この場合、 は にANY()評価されますTrue。

例: **ALL()**

クエリステートメントALL()の の例を次に示します。

```
SELECT 17 < ALL (VALUES 19, 20, 15);
```

同じオペレーションの代替構文は次のとおりです。

```
SELECT 17 < ALL (SELECT 19 UNION ALL SELECT 20 UNION ALL SELECT 15);
```

この場合、 は にALL()評価されますFalse。

例: **SOME()**

クエリステートメントSOME()の の例を次に示します。

```
SELECT 50 >= SOME (VALUES 53, 77, 27);
```

同じオペレーションの代替構文は次のとおりです。

```
SELECT 50 >= SOME (SELECT 53 UNION ALL SELECT 77 UNION ALL SELECT 27);
```

この場合、 は にSOME()評価されますTrue。

条件式

の Timestream は、次の条件式 LiveAnalytics をサポートしています。

トピック

- CASE ステートメント
- IF ステートメント
- COALESCE ステートメント
- NULLIF ステートメント
- TRY ステートメント

CASE ステートメント

CASE ステートメントは、 に等しい値が見つかるまで、左から右に各値式を検索しますexpression。一致が検出されると、一致する値の結果が返されます。一致が見つからない場合、ELSE句の結果は存在する場合は返され、存在しない場合は返nullされます。構文は次のとおりです。

```
CASE expression
    WHEN value THEN result
    [ WHEN ... ]
    [ ELSE result ]
END
```

Timestream は、CASEステートメントに対して次の構文もサポートしています。この構文では、「検索済み」フォームは、左から右に各ブール条件を評価し、一致する結果trueを返します。条件が でない場合true、 ELSE句の結果は存在する場合は返され、存在しない場合は返nullされます。代替構文については、以下を参照してください。

```
CASE
    WHEN condition THEN result
    [ WHEN ... ]
```

```
    [ ELSE result ]
END
```

IF ステートメント

IF ステートメントは、条件を true または false と評価し、適切な値を返します。Timestream は、IF に対して次の 2 つの構文表現をサポートしています。

```
if(condition, true_value)
```

この構文は、条件が true_valueの場合に評価して返しますtrue。それ以外の場合は nullが返され、評価true_valueされません。

```
if(condition, true_value, false_value)
```

この構文は、条件が true_valueの場合に評価して返しますtrue。それ以外の場合は、 を評価して返しますfalse_value。

例

```
SELECT
  if(true, 'example 1'),
  if(false, 'example 2'),
  if(true, 'example 3 true', 'example 3 false'),
  if(false, 'example 4 true', 'example 4 false')
```

_col0	_col1	_col2	_col3
example 1	-	example 3 true	example 4 false
	null		

COALESCE ステートメント

COALESCE は、引数リストの最初の NULL 以外の値を返します。構文は次のとおりです。

```
coalesce(value1, value2[,...])
```

NULLIF ステートメント

IF ステートメントは、条件を true または false として評価し、適切な値を返します。Timestream は、IF に対して次の 2 つの構文表現をサポートしています。

NULLIF は value1と等しい場合は null を返しvalue2、そうでない場合は を返しますvalue1。構文は次のとおりです。

```
nullif(value1, value2)
```

TRY ステートメント

このTRY関数は式を評価し、 を返すことで特定のタイプのエラーを処理しますnull。構文は次のとおりです。

```
try(expression)
```

変換関数

の Timestream は、次の変換関数 LiveAnalytics をサポートしています。

トピック

- cast()
- try_cast()

cast()

値をタイプとして明示的にキャストするキャスト関数の構文は次のとおりです。

```
cast(value AS type)
```

try_cast()

の Timestream は、キャストに似ているが、キャストが失敗すると null を返す try_cast 関数 LiveAnalytics もサポートします。構文は次のとおりです。

```
try_cast(value AS type)
```

算術演算子

の Timestream は、次の数学演算子 LiveAnalytics をサポートしています。

演算子	説明
+	加算
-	減算
*	乗算
/	分割 (整数分割は切り捨てを実行します)
%	モジュラス (リマインダー)

数学関数

の Timestream は、次の数学関数 LiveAnalytics をサポートしています。

機能	出力データ型	説明
abs(x)	〔入力と同じ〕	x の絶対値を返します。
cbrt(x)	double	x のキューブルートを返します。
ceiling(x) または ceil(x)	〔入力と同じ〕	x を最も近い整数に切り上げて返します。
度 (x)	double	角度 x をラジアン単位で度に変換します。
e()	double	定数 Euler の数値を返します。
exp(x)	double	x のパワーに上げられた Euler の数値を返します。

機能	出力データ型	説明
floor(x)	〔入力と同じ〕	x を最も近い整数に切り捨てて返します。
from_base(文字列,radix)	bigint	base-radix 番号として解釈された文字列の値を返します。
ln(x)	double	x の自然対数を返します。
log2(x)	double	x のベース 2 対数を返します。
log10(x)	double	x のベース 10 対数を返します。
mod(n,m)	〔入力と同じ〕	n のモジュラス (リマインダー) を m で割った値を返します。
pi()	double	定数 Pi を返します。
pow(x, p) または power(x, p)	double	p のパワーに上げられた x を返します。
ラジアン (x)	double	角度 x を度単位でラジアンに変換します。
rand() または random()	double	0.0 1.0 の範囲の擬似ランダム値を返します。
random(n)	〔入力と同じ〕	0 から n (排他的) までの擬似ランダムな数値を返します。
round(x)	〔入力と同じ〕	x を最も近い整数に四捨五入して返します。
round(x,d)	〔入力と同じ〕	x を小数点以下第 3 位に四捨五入して返します。

機能	出力データ型	説明
sign(x)	〔入力と同じ〕	x のシグナム関数を返します。つまり、 • 引数が 0 の場合、0 • 引数が 0 より大きい場合は 1 • 引数が 0 未満の場合は -1。 二重引数の場合、関数はさらに以下を返します。 • 引数が NaN の場合の NaN • 引数が +Infinity の場合は 1 • 引数が -Infinity の場合、-1。
sqrt(x)	double	x の平方根を返します。
to_base(x、radix)	varchar	x の base-radix 表現を返します。
truncate(x)	double	小数点以下の桁を削除して x を整数に四捨五入します。
acos(x)	double	x のアークコサインを返します。
asin(x)	double	x のアークサインを返します。
atan(x)	double	x のアークタンジェントを返します。
atan2 (y, x)	double	y/x のアークタンジェントを返します。
cos(x)	double	x のコサインを返します。

機能	出力データ型	説明
cosh(x)	double	x の双曲線コサインを返します。
sin(x)	double	x のサインを返します。
tan(x)	double	x の接線を返します。
tanh(x)	double	x の双曲線タンジェントを返します。
infinity()	double	正の無限を表す定数を返します。
is_finite(x)	ブール値	x が有限であるかどうかを判断します。
is_infinite(x)	ブール値	x が無限であるかどうかを判断します。
is_nan(x)	ブール値	x が であるかどうかを判断します not-a-number。
nan()	double	を表す定数を返します not-a-number。

文字列演算子

の Timestream は、1 つ以上の文字列を連結するための | | 演算子 LiveAnalytics をサポートします。

文字列関数

> **Note**
>
> これらの関数の入力データ型は、特に指定がない限り varchar と見なされます。

機能	出力データ型	説明
chr(n)	varchar	Unicode コードポイント n を varchar として返します。
コードポイント (x)	integer	str の唯一の文字の Unicode コードポイントを返します。
concat(x1、...、xN)	varchar	x1、x2、...、xN の連結を返します。
hamming_distance(x1,x2)	bigint	x1 と x2 のハミング距離を返します。つまり、対応する文字が異なる位置の数を返します。2 つの varchar 入力の長さは同じである必要があることに注意してください。
length(x)	bigint	x の長さを文字で返します。
levenshtein_distance(x1、x2)	bigint	x1 と x2 の Levenshtein 編集距離を返します。つまり、x1 を x2 に変更するために必要な 1 文字編集 (挿入、削除、置換) の最小数を返します。
lower(x)	varchar	x を小文字に変換します。
lpad(x1、ビギントサイズ、x2)	varchar	左パッド x1 からサイズ文字に x2。サイズが x1 の長さより小さい場合、結果はサイズ文字に切り捨てられます。サイズは負にできず、x2 は空にできません。
ltrim(x)	varchar	x から先頭の空白を削除します。

機能	出力データ型	説明
replace(x1, x2)	varchar	x1 から x2 のすべてのインスタンスを削除します。
replace(x1、x2、x3)	varchar	x2 のすべてのインスタンスを x1 の x3 に置き換えます。
リバース (x)	varchar	x を逆の順序で返します。
rpad(x1、ビギントサイズ、x2)	varchar	x1 を右パッドで、x2 で文字のサイズを変更します。サイズが x1 の長さより小さい場合、結果はサイズ文字に切り捨てられます。サイズは負にできず、x2 は空にできません。
rtrim(x)	varchar	x から末尾の空白を削除します。
split(x1, x2)	array(varchar)	区切り文字 x2 で x1 を分割し、配列を返します。
split(x1, x2, bigint の制限)	array(varchar)	区切り文字 x2 で x1 を分割し、配列を返します。配列の最後の要素には、常に x1 に残っているすべてが含まれます。 制限は正の数である必要があります。
split_part(x1、x2、gigint pos)	varchar	区切り文字 x2 で x1 を分割し、 位置で varchar フィールドを返します。フィールドインデックスは 1 で始まります。pos がフィールド数より大きい場合、null が返されます。

機能	出力データ型	説明
strpos(x1、x2)	bigint	x1 内の x2 の最初のインスタンスの開始位置を返します。位置は 1 で始まります。見つからない場合、0 が返されます。
strpos(x1、x2、bigint インスタンス)	bigint	x1 の x2 の N 番目のインスタンスの位置を返します。インスタンスは正の数である必要があります。位置は 1 で始まります。見つからない場合、0 が返されます。
strrpos(x1、x2)	bigint	x1 の x2 の最後のインスタンスの開始位置を返します。位置は 1 で始まります。見つからない場合、0 が返されます。
strrpos(x1、x2、gigint インスタンス)	bigint	x1 の末尾から x1 内の x2 の N 番目のインスタンスの位置を返します。インスタンスは正の数である必要があります。位置は 1 で始まります。見つからない場合、0 が返されます。
position(x2 IN x1)	bigint	x1 内の x2 の最初のインスタンスの開始位置を返します。位置は 1 で始まります。見つからない場合、0 が返されます。

機能	出力データ型	説明
substr(x, bigint start)	varchar	開始位置の開始から x の残りを返します。位置は 1 で始まります。負の開始位置は、x の終わりを基準として解釈されます。
substr(x, bigint start, bigint len)	varchar	開始位置の開始から長さ len の x から部分文字列を返します。位置は 1 で始まります。負の開始位置は、x の終わりを基準として解釈されます。
trim(x)	varchar	x から先頭と末尾の空白を削除します。
upper(x)	varchar	x を大文字に変換します。

配列演算子

の Timestream は、次の配列演算子 LiveAnalytics をサポートしています。

演算子	説明
[]	最初のインデックスが 1 で始まる配列の 要素にアクセスします。
||	配列を同じタイプの別の配列または要素と連結します。

配列関数

の Timestream は、次の配列関数 LiveAnalytics をサポートしています。

機能	出力データ型	説明
array_distinct(x)	array	配列 x から重複する値を削除します。 `SELECT array_distinct(ARRAY[1,2,2,3])` 結果の例: `[ 1,2,3 ]`
array_intersect(x, y)	array	x と y の交差にある要素の配列を重複せずに返します。 `SELECT array_intersect(ARRAY[1,2,3], ARRAY[3,4,5])` 結果の例: `[ 3 ]`
array_union(x, y)	array	x と y の結合内の要素の配列を重複せずに返します。 `SELECT array_union(ARRAY[1,2,3], ARRAY[3,4,5])` 結果の例: `[ 1,2,3,4,5 ]`
array_except(x, y)	array	重複なしで、x で要素の配列を返しますが、y では返しません。 `SELECT array_except(ARRAY[1,2,3], ARRAY[3,4,5])` 結果の例: `[ 1,2 ]`

機能	出力データ型	説明
array_join(x、区切り文字、null_replacement)	varchar	区切り文字とオプションの文字列を使用して特定の配列の要素を連結し、null を置き換えます。 `SELECT array_join(ARRAY[1,2,3], ';', '')` 結果の例: 1;2;3
array_max(x)	配列要素と同じ	入力配列の最大値を返します。 `SELECT array_max(ARRAY[1,2,3])` 結果の例: 3
array_min(x)	配列要素と同じ	入力配列の最小値を返します。 `SELECT array_min(ARRAY[1,2,3])` 結果の例: 1
array_position(x, 要素)	bigint	配列 x (見つからない場合は 0) 内の 要素の最初の出現の位置を返します。 `SELECT array_position(ARRAY[3,4,5,9], 5)` 結果の例: 3

機能	出力データ型	説明
array_remove(x, 要素)	array	配列 x から要素に等しいすべての要素を削除します。 `SELECT array_remove(ARRAY[3,4,5,9], 4)` 結果の例: `[ 3,5,9 ]`
array_sort(x)	array	配列 x をソートして返します。x の要素は順序付け可能である必要があります。Null 要素は、返された配列の最後に配置されます。 `SELECT array_sort(ARRAY[6,8,2,9,3])` 結果の例: `[ 2,3,6,8,9 ]`
arrays_overlap(x, y)	ブール値	配列 x と y に NULL 以外の要素が共通しているかどうかをテストします。共通の NULL 以外の要素がなく、いずれかの配列に null が含まれている場合、null を返します。 `SELECT arrays_overlap(ARRAY[6,8,2,9,3], ARRAY[6,8])` 結果の例: `true`

機能	出力データ型	説明
カーディナリティ (x)	bigint	配列 x のサイズを返します。 `SELECT cardinality(ARRAY[6,8,2,9,3])` 結果の例: 5
concat(array1, array2, ..., arrayN)	array	arrays array1、配列2、...、arrayNを連結します。 `SELECT concat(ARRAY[6,8,2,9,3], ARRAY[11,32], ARRAY[6,8,2,0,14])` 結果の例: `[ 6,8,2,9,3,11,32,6,8,2,0,14 ]`
element_at(array(E)、インデックス)	E	指定されたインデックスで配列の要素を返します。index < 0 の場合、 element_at は最後の から最初の までの要素にアクセスします。 `SELECT element_at(ARRAY[6,8,2,9,3], 1)` 結果の例: 6
repeat(要素、カウント)	array	カウント時間に 要素を繰り返します。 `SELECT repeat(1, 3)` 結果の例: `[ 1,1,1 ]`

機能	出力データ型	説明
reverse(x)	array	配列 x の逆の順序を持つ配列を返します。 `SELECT reverse(ARRAY[6,8,2,9,3])` 結果の例: `[ 3,9,2,8,6 ]`
sequence(開始、停止)	array(bigint)	開始から終了までの整数シーケンスを生成し、開始が停止以下の場合は 1 ずつ増分します。それ以外の場合は -1 です。 `SELECT sequence(3, 8)` 結果の例: `[ 3,4,5,6,7,8 ]`
sequence(開始、停止、ステップ)	array(bigint)	開始から停止まで整数のシーケンスを生成し、ステップごとに増分します。 `SELECT sequence(3, 15, 2)` 結果の例: `[ 3,5,7,9,11,13,15 ]`

機能	出力データ型	説明
sequence(開始、停止)	array(タイムスタンプ)	開始日から終了日までのタイムスタンプのシーケンスを生成し、1 日ずつ増分します。 `SELECT sequence( '2023-04-02 19:26:12.941000000', '2023-04-06 19:26:12.941000000', 1d)` 結果の例: [2023-04-02 19:26:12.941000000 ,2023-04-03 19:26:12.941000000 ,2023-04-04 19:26:12.941000000 ,2023-04-05 19:26:12.941000000 ,2023-04-06 19:26:12.941000000]

機能	出力データ型	説明
sequence(開始、停止、ステップ)	array(タイムスタンプ)	開始から停止までの一連のタイムスタンプを生成し、ステップごとに増分します。ステップのデータ型は間隔です。 `SELECT sequence( '2023-04-02 19:26:12.941000000', '2023-04-10 19:26:12.941000000', 2d)` 結果の例: `[ 2023-04-02 19:26:12.941000000 ,2023-04-04 19:26:12.941000000 ,2023-04-06 19:26:12.941000000 ,2023-04-08 19:26:12.941000000 ,2023-04-10 19:26:12.941000000 ]`
shuffle(x)	array	指定された配列 x のランダムな置換を生成します。 `SELECT shuffle(ARRAY[6,8,2,9,3])` 結果の例: `[ 6,3,2,9,8 ]`

機能	出力データ型	説明
slice(x、開始、長さ)	array	長さが のインデックス開始から (または開始が負の場合は終了から) 配列 x をサブセットします。 `SELECT slice(ARRAY[6,8,2,9,3], 1, 3)` 結果の例: `[ 6,8,2 ]`
zip(array1, array2[, ...])	array(行)	指定された配列を要素ごとに 1 つの行配列にマージします。引数の長さが不均等である場合、欠損値は で埋められますNULL。 `SELECT zip(ARRAY[6,8,2,9,3], ARRAY[15, 24])` 結果の例: `[ ( 6, 15 ), ( 8, 24 ),( 2, - ), ( 9, - ),( 3, - ) ]`

ビット単位関数

の Timestream は、次のビット単位関数 LiveAnalytics をサポートしています。

機能	出力データ型	説明
bit_count(bigint, bigint)	bigint (2 の補数)	2 番目のパラメータが 8 や 64 などのビット符号付き整数である場合、最初の bigint パラメータのビット数を返します。

機能	出力データ型	説明

		`SELECT bit_count(19, 8)`
		
		結果の例: 3
		
		`SELECT bit_count(19, 2)`
		
		結果の例: `Number must be representable with the bits specified . 19 can not be represented with 2 bits`

bitwise_and (bigint、bigint)	bigint (2 の補数)	bigint パラメータANDのビット単位を返します。 `SELECT bitwise_and(12, 7)` 結果の例: 4
bitwise_not(bigint)	bigint (2 の補数)	bigint パラメータNOTのビット単位を返します。 `SELECT bitwise_not(12)` 結果の例: -13
bitwise_or (bigint、bigint)	bigint (2 の補数)	bigint パラメータのビット単位の OR を返します。 `SELECT bitwise_or(12, 7)` 結果の例: 15

機能	出力データ型	説明
bitwise_xor(bigint、bigint)	bigint (2 の補数)	bigint パラメータXORのビット単位を返します。 `SELECT bitwise_xor(12, 7)` 結果の例: 11

正規表現関数

の Timestream の正規表現関数は、Java パターン構文 LiveAnalytics をサポートします。の Timestream は、次の正規表現関数 LiveAnalytics をサポートしています。

機能	出力データ型	説明
regexp_extract_all(文字列、パターン)	array(varchar)	文字列の正規表現パターンで一致した部分文字列 (複数可) を返します。 `SELECT regexp_extract_all('example expect complex', 'ex\w')` 結果の例: `[ exa,exp ]`
regexp_extract_all(文字列、パターン、グループ)	array(varchar)	文字列内の正規表現パターンのすべての出現を検出し、キャプチャグループ番号グループを返します。 `SELECT regexp_extract_all('example expect complex', '(ex)(\w)', 2)`

機能	出力データ型	説明
		結果の例: `[ a,p ]`
regexp_extract(文字列、パターン)	varchar	文字列内の正規表現パターンで一致させた最初の部分文字列を返します。 `SELECT regexp_extract('example expect', 'ex\w')` 結果の例: `exa`
regexp_extract(文字列、パターン、グループ)	varchar	文字列内の正規表現パターンの最初の出現を検出し、キャプチャグループ番号グループを返します。 `SELECT regexp_extract('example expect', '(ex)(\w)', 2)` 結果の例: a

機能	出力データ型	説明
regexp_like(文字列、パターン)	ブール値	正規表現パターンを評価し、文字列内に含まれているかどうかを判断します。この関数は LIKE演算子に似ていますが、パターンはすべての文字列と一致するのではなく、文字列内に含めるだけで済みます。つまり、一致オペレーションではなく、 を含むオペレーションを実行します。^ と $ を使用してパターンを固定することで、文字列全体を一致させることができます。 `SELECT regexp_like('example', 'ex')` 結果の例: `true`
regexp_replace(文字列、パターン)	varchar	文字列から正規表現パターンに一致する部分文字列のすべてのインスタンスを削除します。 `SELECT regexp_replace('example expect', 'expect')` 結果の例: `example`

機能	出力データ型	説明
regexp_replace(文字列、パターン、置換)	varchar	文字列内の正規表現パターンで一致した部分文字列のすべてのインスタンスを置き換えます。キャプチャグループは、番号付きグループの $g または名前付きグループの ${name} を使用して置き換えで参照できます。バックスラッシュ (\$) でエスケープすることで、置き換えにドル記号 ($) を含めることができます。 `SELECT regexp_replace('example expect', 'expect', 'surprise')` 結果の例: `example surprise`

機能	出力データ型	説明
regexp_replace(文字列、パターン、関数)	varchar	関数を使用して、文字列内の正規表現パターンで一致した部分文字列のすべてのインスタンスを置き換えます。Lambda 式関数は、配列として渡されたキャプチャグループとの一致ごとに呼び出されます。グループ番号の取得は 1 から始まり、一致全体のグループはありません (これが必要な場合は、式全体を括弧で囲みます)。 `SELECT regexp_replace('example', '(\w)', x -> upper(x[1]))` 結果の例: EXAMPLE
regexp_split(文字列、パターン)	array(varchar)	正規表現パターンを使用して文字列を分割し、配列を返します。末尾の空の文字列は保持されます。 `SELECT regexp_split('example', 'x')` 結果の例: `[ e,ample ]`

日付/時刻演算子

> **Note**
>
> の Timestream LiveAnalytics は負の時間値をサポートしていません。負の時間になる操作はエラーになります。

の Timestream は、`timestamps`、、`dates`および で次のオペレーション LiveAnalytics をサポートしますintervals。

演算子	説明
+	加算
-	減算

トピック

オペレーション

オペレーションの結果タイプは、オペランドに基づいています。1day や などの間隔リテラル3sを使用できます。

```
SELECT date '2022-05-21' + interval '2' day
```

```
SELECT date '2022-05-21' + 2d
```

```
SELECT date '2022-05-21' + 2day
```

それぞれの結果の例: 2022-05-23

間隔単位には、second、、minutehour、、month、dayweekおよび が含まれますyear。ただし、場合によっては、すべてが適用できるとは限りません。例えば、秒、分、および時間は日付に追加したり、日付から減算したりすることはできません。

```
SELECT interval '4' year + interval '2' month
```

結果の例: 4-2

```
SELECT typeof(interval '4' year + interval '2' month)
```

結果の例: interval year to month

間隔オペレーションの結果タイプは、オペランド'interval day to second'に応じて'interval year to month'または になります。間隔は、 dates および に追加または減算できますtimestamps。ただし、 dateまたは を dateまたは に追加したり、 から減算timestampしたりすることはできませんtimestamp。日付またはタイムスタンプに関連する間隔または期間を確認するには、「」のdate_diff「 および関連する関数」を参照してください間隔と期間。

加算

Example

```
SELECT date '2022-05-21' + interval '2' day
```

結果の例: 2022-05-23

Example

```
SELECT typeof(date '2022-05-21' + interval '2' day)
```

結果の例: date

Example

```
SELECT interval '2' year + interval '4' month
```

結果の例: 2-4

Example

```
SELECT typeof(interval '2' year + interval '4' month)
```

結果の例: `interval year to month`

減算

Example

```
SELECT timestamp '2022-06-17 01:00' - interval '7' hour
```

結果の例: `2022-06-16 18:00:00.000000000`

Example

```
SELECT typeof(timestamp '2022-06-17 01:00' - interval '7' hour)
```

結果の例: `timestamp`

Example

```
SELECT interval '6' day - interval '4' hour
```

結果の例: `5 20:00:00.000000000`

Example

```
SELECT typeof(interval '6' day - interval '4' hour)
```

結果の例: `interval day to second`

日付/時刻関数

> **Note**
>
> の Timestream LiveAnalytics は負の時間値をサポートしていません。負の時間になる操作はエラーになります。

の Timestream LiveAnalytics は、日付と時刻のUTCタイムゾーンを使用します。Timestream は、日付と時刻に対して次の関数をサポートしています。

トピック

全般と変換

の Timestream は、日付と時刻の次の一般関数と変換関数 LiveAnalytics をサポートしています。

機能	出力データ型	説明
current_date	date	で現在の日付を返します UTC。括弧は使用されません。 `SELECT current_date` 結果の例: `2022-07-07` **Note** これはリザーブドキーワードでもあります。予約キーワードのリストについては、「」を参照してください予約済みキーワード。
current_time	時系	で現在の時刻を返します UTC。括弧は使用されません。

機能	出力データ型	説明
		`SELECT current_time` 結果の例: `17:41:52.827000000` **Note** これはリザーブドキーワードでもあります。予約キーワードのリストについては、「」を参照してください予約済みキーワード。
current_timestamp または now()	timestamp	で現在のタイムスタンプを返しますUTC。 `SELECT current_timestamp` 結果の例: `2022-07-07 17:42:32.939000000` **Note** これはリザーブドキーワードでもあります。予約キーワードのリストについては、「」を参照してください予約済みキーワード。

機能	出力データ型	説明
current_timezone()	varchar 値は「」になりますUTC。	Timestream は、日付と時刻にUTCタイムゾーンを使用します。 `SELECT current_timezone()` 結果の例: UTC
date(varchar(x))、date(timestamp)	date	`SELECT date(TIMESTAMP '2022-07-07 17:44:43.771000000')` 結果の例: 2022-07-07
last_day_of_month(タイムスタンプ)、last_day_of_month(日付)	date	`SELECT last_day_of_month(TIMESTAMP '2022-07-07 17:44:43.771000000')` 結果の例: 2022-07-31
from_iso8601_timestamp(文字列)	timestamp	8601 ISO タイムスタンプを内部タイムスタンプ形式に解析します。 `SELECT from_iso8601_timestamp('2022-06-17T08:04:05.000000000+05:00')` 結果の例: 2022-06-17 03:04:05.000000000

機能	出力データ型	説明
from_iso8601_date(文字列)	date	8601 ISO 日付文字列を、指定された日付の 00:00:00 UTC の内部タイムスタンプ形式に解析します。 `SELECT from_iso8601_date('2022-07-17')` 結果の例: 2022-07-17
to_iso8601 (タイムスタンプ)、to_iso8601 (日付)	varchar	入力の ISO 8601 形式の文字列を返します。 `SELECT to_iso8601(from_iso8601_date('2022-06-17'))` 結果の例: 2022-06-17
from_milliseconds(bigint)	timestamp	`SELECT from_milliseconds(1)` 結果の例: 1970-01-01 00:00:00.001000000
from_nanoseconds(bigint)	timestamp	`select from_nanoseconds(300000001)` 結果の例: 1970-01-01 00:00:00.300000001

機能	出力データ型	説明
from_unixtime(ダブル)	timestamp	指定された unixtime に対応するタイムスタンプを返します。 `SELECT from_unixtime(1)` 結果の例: `1970-01-01 00:00:01.000000000`
localtime	時系	で現在の時刻を返します UTC。括弧は使用されません。 `SELECT localtime` 結果の例: `17:58:22.654000000` **Note** これは予約キーワードでもあります。予約キーワードのリストについては、「」を参照してください予約済みキーワード。

機能	出力データ型	説明
localtimestamp	timestamp	で現在のタイムスタンプを返しますUTC。括弧は使用されません。 `SELECT localtimestamp` 結果の例: `2022-07-07 17:59:04.368000000` **Note** これは予約キーワードでもあります。予約キーワードのリストについては、「」を参照してください予約済みキーワード。

to_milliseconds (間隔日 ~ 秒) 、to_milliseconds (タイムスタンプ)	bigint	`SELECT to_milliseconds(INTERVAL '2' DAY + INTERVAL '3' HOUR)`
		結果の例: `183600000`
		`SELECT to_milliseconds(TIMESTAMP '2022-06-17 17:44:43.771000000')`
		結果の例: `1655487883771`

機能	出力データ型	説明

to_nanoseconds (間隔日から秒)、to_nanoseconds (タイムスタンプ)	bigint	`SELECT to_nanoseconds(INTERVAL '2' DAY + INTERVAL '3' HOUR)`
		
		結果の例: 183600000000000
		
		`SELECT to_nanoseconds(TIMESTAMP '2022-06-17 17:44:43.771000678')`
		
		結果の例: 1655487883771000678

to_unixtime(タイムスタンプ)	double	指定されたタイムスタンプの unixtime を返します。 `SELECT to_unixtime('2022-06-17 17:44:43.771000000')` 結果の例: 1.655487883771E9

機能	出力データ型	説明
date_trunc(単位、タイムスタンプ)	timestamp	単位に切り捨てられたタイムスタンプを返します。単位は [秒、分、時間、日、週、月、四半期、年] のいずれかです。 `SELECT date_trunc('minute', TIMESTAMP '2022-06-17 17:44:43.771000000')` 結果の例: `2022-06-17 17:44:00.000000000`

間隔と期間

の Timestream は、日付と時刻の以下の間隔と期間関数 LiveAnalytics をサポートします。

機能	出力データ型	説明
date_add(unit、bigint、date)、date_add(unit、bigint、time)、date_add(varchar(x)、bigint、timestamp)	timestamp	単位が [秒、分、時間、日、週、月、四半期、または年] のいずれかである単位の重みを追加します。 `SELECT date_add('hour', 9, TIMESTAMP '2022-06-17 00:00:00')` 結果の例: `2022-06-17 09:00:00.000000000`
date_diff (単位、日付、日付) 、date_diff (単位、時刻、時刻) 、date_diff (単位、タイムスタンプ、タイムスタンプ)	bigint	単位が [秒、分、時間、日、週、月、四半期、年] のいずれかである差を返します。

機能	出力データ型	説明
		`SELECT date_diff('day', DATE '2020-03-01', DATE '2020-03-02')` 結果の例: `1`

parse_duration(文字列)	interval	入力文字列を解析してinterval同等の を返します。
		
		`SELECT parse_duration('42.8ms')`
		
		結果の例: `0 00:00:00.042800000`
		
		`SELECT typeof(parse_duration('42.8ms'))`
		
		結果の例: `interval day to second`

機能	出力データ型	説明

bin (タイムスタンプ、間隔)	timestamp	`timestamp` パラメータの整数値を、`interval`パラメータの整数値の最も近い倍数に切り捨てます。
		
		この戻り値の意味は明らかでない場合があります。これは、最初にタイムスタンプ整数を間隔整数で割ってから、結果を間隔整数で乗算することで整数算術を使用して計算されます。
		
		タイムスタンプは、POSIXエポック (1970 年 1 月 1 日) から経過した 1 秒の分数としてUTC時点を指定することに留意すると、戻り値はカレンダー単位とほとんど一致しません。例えば、30 日間の間隔を指定すると、エポックからのすべての日が 30 日単位で分割され、直近の 30 日単位の開始が返されます。これは暦月とは関係ありません。
		
		次に例を示します。
		
		`bin(TIMESTAMP '2022-06-17 10:15:20', 5m)`
		`==> 2022-06-17 10:15:00.000000000`
		`bin(TIMESTAMP '2022-06-17 10:15:20', 1d)`
		`==> 2022-06-17 00:00:00.000000000`

機能	出力データ型	説明
		`bin(TIMESTAMP '2022-06-17 10:15:20', 10day) ==> 2022-06-17 00:00:00.000000000` `bin(TIMESTAMP '2022-06-17 10:15:20', 30day) ==> 2022-05-28 00:00:00.000000000`
ago(間隔)	timestamp	current_timestamp に対応する値を返しますinterval。 `SELECT ago(1d)` 結果の例: `2022-07-06 21:08:53.245000000`
1h、1d、30m などの間隔リテラル	interval	間隔リテラルは、parse_duration(文字列) の利便性です。たとえば、1d は `parse_duration('1d')` と同じです。これにより、間隔が使用されている場所でリテラルを使用できます。例えば、ago(1d) と bin(*<timestamp>* , 1m) です。

一部の間隔リテラルは、parse_duration の略語として機能します。例えば、`parse_duration('1day')`、`parse_duration('1d')`、1day、およびタイプ`1 00:00:00.000000000`が である`1d`各戻り`interval day to second`値。スペースは、 に提供される形式で許可されます`parse_duration`。例えば、 `parse_duration('1day')`も を返します`00:00:00.000000000`。ただし、 `1 day`は間隔リテラルではありません。

に関連する単位`interval day to second`は、ns、nanosecond、us、microsecond、ms、ms、ms、ms、s、second、m、min、h、hour、d、です。

もありますinterval year to month。間隔の年と月に関連する単位は、y、年、月です。例えば、 は SELECT 1yearを返します1-0。 SELECT 12monthも を返します1-0。 は SELECT 8monthを返します0-8。

の単位quarterは、 date_truncや などの一部の関数でも使用できますquarterがdate_add、間隔リテラルの一部としては使用できません。

フォーマットと解析

の Timestream では、日付と時刻の次の書式設定と解析関数 LiveAnalytics がサポートされています。

機能	出力データ型	説明
date_format(タイムスタンプ、varchar(x))	varchar	この関数で使用される形式指定子の詳細については、https://trino.io/docs/current/functions/datetime.html「#mysql-date-functions」を参照してください。 `SELECT date_format(TIMESTAMP '2019-10-20 10:20:20', '%Y-%m-%d %H:%i:%s')` 結果の例: `2019-10-20 10:20:20`
date_parse(varchar(x)、varchar(y))	timestamp	この関数で使用される形式指定子の詳細については、https://trino.io/docs/current/functions/datetime.html「#mysql-date-functions」を参照してください。

機能	出力データ型	説明
		`SELECT date_parse('2019-10-20 10:20:20', '%Y-%m-%d %H:%i:%s')` 結果の例: `2019-10-20 10:20:20.000000000`
format_datetime(タイムスタンプ、varchar(x))	varchar	この関数で使用される形式文字列の詳細については、http://joda-time.sourceforge.net/apidocs/org/joda/time/format/DateTimeFormat.html を参照してください。 `SELECT format_datetime(parse_datetime('1968-01-13 12', 'yyyy-MM-dd HH'), 'yyyy-MM-dd HH')` 結果の例: `1968-01-13 12`

機能	出力データ型	説明
parse_datetime(varchar(x)、varchar(y))	timestamp	この関数で使用される形式文字列の詳細については、http://joda-time.sourceforge.net/apidocs/org/joda/time/format/DateTimeFormat.html を参照してください。 `SELECT parse_datetime('2019-12-29 10:10 PST', 'uuuu-LL-dd HH:mm z')` 結果の例: `2019-12-29 18:10:00.000000000`

抽出

の Timestream は、日付と時刻に対して次の抽出関数 LiveAnalytics をサポートします。抽出関数は、残りの利便性関数の基礎です。

機能	出力データ型	説明
extract	bigint	タイムスタンプからフィールドを抽出します。ここで、フィールドは [YEAR、、QUARTER、MONTHWEEK、DAY、、DAY_OF_MONTH、DAY_OF_WEEK、DOW、DAY_OF_YEAR、DOY、YEAR_OF_WEEK、YOW、、MINUTE、、または] HOURのいずれかですSECOND。

機能	出力データ型	説明
		`SELECT extract(YEAR FROM '2019-10-12 23:10:34.000000000')` 結果の例: 2019
day (タイムスタンプ)、day (日付)、day (間隔日 ~ 秒)	bigint	`SELECT day('2019-10-12 23:10:34.000000000')` 結果の例: 12
day_of_month(タイムスタンプ)、day_of_month(日付)、day_of_month(間隔日 ~ 秒)	bigint	`SELECT day_of_month('2019-10-12 23:10:34.000000000')` 結果の例: 12
day_of_week(タイムスタンプ)、day_of_week(日付)	bigint	`SELECT day_of_week('2019-10-12 23:10:34.000000000')` 結果の例: 6
day_of_year(タイムスタンプ)、day_of_year(日付)	bigint	`SELECT day_of_year('2019-10-12 23:10:34.000000000')` 結果の例: 285
dow (タイムスタンプ)、dow (日付)	bigint	day_of_week のエイリアス
doy (タイムスタンプ)、doy (日付)	bigint	day_of_year のエイリアス

機能	出力データ型	説明
hour (タイムスタンプ)、hour (時間)、 hour (間隔日 ~ 秒)	bigint	`SELECT hour('2019-10-12 23:10:34.000000000')` 結果の例: 23
ミリ秒 (タイムスタンプ)、ミリ秒 (時間)、ミリ秒 (間隔日 ~ 秒)	bigint	`SELECT millisecond('2019-10-12 23:10:34.000000000')` 結果の例: 0
minute(タイムスタンプ)、minute(時間)、minute(間隔日 ~ 秒)	bigint	`SELECT minute('2019-10-12 23:10:34.000000000')` 結果の例: 10
month (タイムスタンプ)、month (日付)、month (年々の間隔)	bigint	`SELECT month('2019-10-12 23:10:34.000000000')` 結果の例: 10
nanosecond (タイムスタンプ)、nanosecond (時間)、nanosecond (日間隔から秒間隔)	bigint	`SELECT nanosecond(current_timestamp)` 結果の例: 162000000
quarter(タイムスタンプ)、quarter(日付)	bigint	`SELECT quarter('2019-10-12 23:10:34.000000000')` 結果の例: 4

機能	出力データ型	説明
second (タイムスタンプ)、second (時間)、second (間隔日 ~ 秒)	bigint	`SELECT second('2019-10-12 23:10:34.000000000')` 結果の例: 34
week (タイムスタンプ)、week (日付)	bigint	`SELECT week('2019-10-12 23:10:34.000000000')` 結果の例: 41
week_of_year(タイムスタンプ)、week_of_year(日付)	bigint	週のエイリアス
year (タイムスタンプ)、year (日付)、year (年と月間隔)	bigint	`SELECT year('2019-10-12 23:10:34.000000000')` 結果の例: 2019
year_of_week(タイムスタンプ)、year_of_week(日付)	bigint	`SELECT year_of_week('2019-10-12 23:10:34.000000000')` 結果の例: 2019
yow(タイムスタンプ)、yow(日付)	bigint	year_of_week のエイリアス

集計関数

の Timestream は、次の集計関数 LiveAnalytics をサポートしています。

機能	出力データ型	説明
任意の (x)	〔入力と同じ〕	存在する場合は、任意の NULL 以外の値 x を返します。 `SELECT arbitrary(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 結果の例: 1
array_agg(x)	array<[入力と同じ〕	入力 x 要素から作成された配列を返します。 `SELECT array_agg(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 結果の例: [1,2,3,4]
avg(x)	double	すべての入力値の平均 (算術平均) を返します。 `SELECT avg(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 結果の例: 2.5
bool_and(ブール値) every(ブール値)	ブール値	すべての入力値が TRUEの場合は をTRUE返し、それ以外の場合は を返しますFALSE。 `SELECT bool_and(t.c) FROM (VALUES true, true, false, true) AS t(c)`

機能	出力データ型	説明
		結果の例: `false`
bool_or (ブール値)	ブール値	入力値が TRUEの場合は を TRUE返し、それ以外の場合は を返しますFALSE。 `SELECT bool_or(t.c) FROM (VALUES true, true, false, true) AS t(c)` 結果の例: `true`
count(*) count(x)	bigint	count(*) は入力行数を返します。 count(x) は NULL 以外の入力値の数を返します。 `SELECT count(t.c) FROM (VALUES true, true, false, true) AS t(c)` 結果の例: 4
count_if(x)	bigint	TRUE 入力値の数を返します。 `SELECT count_if(t.c) FROM (VALUES true, true, false, true) AS t(c)` 結果の例: 3

機能	出力データ型	説明
geometric_mean(x)	double	すべての入力値の幾何平均を返します。 `SELECT geometric_mean(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 結果の例: 2.213363839400643
max_by(x, y)	[x と同じ〕	すべての入力値で y の最大値に関連付けられた x の値を返します。 `SELECT max_by(t.c1, t.c2) FROM (VALUES (('a', 1)), (('b', 2)), (('c', 3)), (('d', 4))) AS t(c1, c2)` 結果の例: d
max_by(x、y、n)	array<[x と同じ]>	y のすべての入力値の n の最大値に関連付けられた x の n 値を y の降順で返します。 `SELECT max_by(t.c1, t.c2, 2) FROM (VALUES (('a', 1)), (('b', 2)), (('c', 3)), (('d', 4))) AS t(c1, c2)` 結果の例: [d,c]

機能	出力データ型	説明
min_by(x, y)	[x と同じ〕	すべての入力値にわたって y の最小値に関連付けられた x の値を返します。 `SELECT min_by(t.c1, t.c2) FROM (VALUES (('a', 1)), (('b', 2)), (('c', 3)), (('d', 4))) AS t(c1, c2)` 結果の例: a
min_by(x、y、n)	array<[x と同じ]>	y のすべての入力値の最小 n 値に関連付けられた x の n 値を y の昇順で返します。 `SELECT min_by(t.c1, t.c2, 2) FROM (VALUES (('a', 1)), (('b', 2)), (('c', 3)), (('d', 4))) AS t(c1, c2)` 結果の例: [a,b]
最大 (x)	〔入力と同じ〕	すべての入力値の最大値を返します。 `SELECT max(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 結果の例: 4

機能	出力データ型	説明
max(x、n)	array<[x と同じ]>	x のすべての入力値の最大値 n を返します。 `SELECT max(t.c, 2) FROM (VALUES 1, 2, 3, 4) AS t(c)` 結果の例: [4,3]
min(x)	〔入力と同じ〕	すべての入力値の最小値を返します。 `SELECT min(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 結果の例: 1
min(x、n)	array<[x と同じ]>	x のすべての入力値の最小値 n を返します。 `SELECT min(t.c, 2) FROM (VALUES 1, 2, 3, 4) AS t(c)` 結果の例: [1,2]
sum(x)	〔入力と同じ〕	すべての入力値の合計を返します。 `SELECT sum(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 結果の例: 10

機能	出力データ型	説明
bitwise_and_agg(x)	bigint	2 ANDの補数表現のすべての入力値のビット単位を返します。 `SELECT bitwise_and_agg(t.c) FROM (VALUES 1, -3) AS t(c)` 結果の例: 1
bitwise_or_agg(x)	bigint	2 の補数表現のすべての入力値のビット単位の OR を返します。 `SELECT bitwise_or_agg(t.c) FROM (VALUES 1, -3) AS t(c)` 結果の例: -3

機能	出力データ型	説明
approx_distinct(x)	bigint	個別の入力値の概算数を返します。この関数は count(DISTINCTx) の近似を提供します。すべての入力値が null の場合、ゼロが返されます。この関数は、2.3% の標準誤差を生成する必要があります。これは、すべての可能なセットに対する (ほぼ正規の) エラー分布の標準偏差です。特定の入力セットのエラーの上限は保証されません。 `SELECT approx_distinct(t.c) FROM (VALUES 1, 2, 3, 4, 8) AS t(c)` 結果の例: 5

機能	出力データ型	説明
approx_distinct(x, e)	bigint	個別の入力値の概算数を返します。この関数は count(DISTINCTx) の近似を提供します。すべての入力値が null の場合、ゼロが返されます。この関数は、e 以下の標準誤差を生成する必要があります。これは、すべての可能なセットに対する (ほぼ正規の) エラー分布の標準偏差です。特定の入力セットのエラーの上限は保証されません。この関数の現在の実装では、e が [0.0040625, 0.26000] の範囲内にある必要があります。 `SELECT approx_distinct(t.c, 0.2) FROM (VALUES 1, 2, 3, 4, 8) AS t(c)` 結果の例: 5

機能	出力データ型	説明
approx_percentile(x, percentage)	[x と同じ〕	指定されたパーセンテージで x のすべての入力値のおおよそのパーセンタイルを返します。パーセンテージの値は 0 から 1 の間で、すべての入力行で一定である必要があります。 `SELECT approx_percentile(t.c, 0.4) FROM (VALUES 1, 2, 3, 4) AS t(c)` 結果の例: 2
approx_percentile(x、パーセンテージ)	array<[x と同じ]>	指定された各パーセンテージで x のすべての入力値のおおよそのパーセンタイルを返します。percentages 配列の各要素は 0 から 1 の間であり、配列はすべての入力行で一定である必要があります。 `SELECT approx_percentile(t.c, ARRAY[0.1, 0.8, 0.8]) FROM (VALUES 1, 2, 3, 4) AS t(c)` 結果の例: `[ 1,4,4 ]`

機能	出力データ型	説明
approx_percentile(x、w、パーセンテージ)	[x と同じ〕	項目ごとの重み w をパーセンテージ p で使用して、x のすべての入力値のおおよその重み付けされたパーセンタイルを返します。重みは少なくとも 1 つの整数値である必要があります。実質的には、パーセンタイルセットの値 x のレプリケーション数です。p の値は 0 から 1 の間で、すべての入力行で一定である必要があります。 `SELECT approx_percentile(t.c, 1, 0.1) FROM (VALUES 1, 2, 3, 4) AS t(c)` 結果の例: 1

機能	出力データ型	説明
approx_percentile(x、w、パーセンテージ)	array<[x と同じ]>	配列で指定された特定のパーセンテージのそれぞれで項目ごとの重み w を使用して、x のすべての入力値のおおよその重み付きパーセンタイルを返します。重みは少なくとも 1 つの整数値である必要があります。実質的には、パーセンタイルセットの値 x のレプリケーション数です。配列の各要素は 0 から 1 の間でなければならず、配列はすべての入力行で一定である必要があります。 `SELECT approx_percentile(t.c, 1, ARRAY[0.1, 0.8, 0.8]) FROM (VALUES 1, 2, 3, 4) AS t(c)` 結果の例: `[ 1,4,4 ]`

機能	出力データ型	説明
approx_percentile(x、w、パーセンテージ、精度)	[x と同じ〕	最大ランク誤差の精度で、項目ごとの重み w をパーセンテージ p で使用して、x のすべての入力値のおおよその重み付きパーセンタイルを返します。重みは少なくとも 1 つの整数値である必要があります。実質的には、パーセンタイルセットの値 x のレプリケーション数です。p の値は 0 から 1 の間で、すべての入力行で一定である必要があります。精度は 0 より大きく 1 より小さい値で、すべての入力行で一定である必要があります。 `SELECT approx_percentile(t.c, 1, 0.1, 0.5) FROM (VALUES 1, 2, 3, 4) AS t(c)` 結果の例: 1
corr (y、x)	double	入力値の相関係数を返します。 `SELECT corr(t.c1, t.c2) FROM (VALUES ((1, 1)), ((2, 2)), ((3, 3)), ((4, 4))) AS t(c1, c2)` 結果の例: 1.0

機能	出力データ型	説明
covar_pop(y、x)	double	入力値の母集団共分散を返します。 `SELECT covar_pop(t.c1, t.c2) FROM (VALUES ((1, 1)), ((2, 2)), ((3, 3)), ((4, 4))) AS t(c1, c2)` 結果の例: `1.25`
covar_samp(y, x)	double	入力値のサンプル共分散を返します。 `SELECT covar_samp(t.c1, t.c2) FROM (VALUES ((1, 1)), ((2, 2)), ((3, 3)), ((4, 4))) AS t(c1, c2)` 結果の例: `1.6666666666666667`
regr_intercept(y, x)	double	入力値の線形回帰インターセプトを返します。y は従属値です。x は独立値です。 `SELECT regr_intercept(t.c1, t.c2) FROM (VALUES ((1, 1)), ((2, 2)), ((3, 3)), ((4, 4))) AS t(c1, c2)` 結果の例: `0.0`

機能	出力データ型	説明
regr_slope(y, x)	double	入力値の線形回帰スロープを返します。y は従属値です。x は独立値です。 `SELECT regr_slope(t.c1, t.c2) FROM (VALUES ((1, 1)), ((2, 2)), ((3, 3)), ((4, 4))) AS t(c1, c2)` 結果の例: `1.0`
スキューネス (x)	double	すべての入力値の歪みを返します。 `SELECT skewness(t.c1) FROM (VALUES 1, 2, 3, 4, 8) AS t(c1)` 結果の例: `0.8978957037987335`
stddev_pop(x)	double	すべての入力値の母集団標準偏差を返します。 `SELECT stddev_pop(t.c1) FROM (VALUES 1, 2, 3, 4, 8) AS t(c1)` 結果の例: `2.4166091947189146`

機能	出力データ型	説明
stddev_samp(x) stddev(x)	double	すべての入力値のサンプル標準偏差を返します。 `SELECT stddev_samp(t.c1) FROM (VALUES 1, 2, 3, 4, 8) AS t(c1)` 結果の例: 2.701851217221259
var_pop(x)	double	すべての入力値の母集団分散を返します。 `SELECT var_pop(t.c1) FROM (VALUES 1, 2, 3, 4, 8) AS t(c1)` 結果の例: 5.840000000000001
var_samp(x) 分散(x)	double	すべての入力値の分散例を返します。 `SELECT var_samp(t.c1) FROM (VALUES 1, 2, 3, 4, 8) AS t(c1)` 結果の例: 7.300000000000001

Window 関数

ウィンドウ関数は、クエリ結果の行間で計算を実行します。これらは HAVING句の後、ORDERBY句の前に実行されます。ウィンドウ関数を呼び出すには、 OVER句を使用してウィンドウを指定する特別な構文が必要です。ウィンドウには 3 つのコンポーネントがあります。

- 入力行を異なるパーティションに分割するパーティション仕様。これは、GROUPBY 句が集計関数のために行を異なるグループに分割する方法に似ています。
- ウィンドウ関数によって入力行が処理される順序を決定する順序指定。
- ウィンドウフレーム。特定の行の関数によって処理される行のスライディングウィンドウを指定します。フレームが指定されていない場合、デフォルトで RANGE UNBOUNDED に設定されます。これは PRECEDINGと同じです RANGEBETWEENUNBOUNDEDPRECEDINGANDCURRENTROW。このフレームには、パーティションの先頭から現在の行の最後のピアまでのすべての行が含まれます。

OVER 句を追加することで、すべての集計関数をウィンドウ関数として使用できます。集計関数は、現在の行のウィンドウフレーム内の行上の各行に対して計算されます。Timestream for では、集計関数に加えて、次のランキング関数と値関数 LiveAnalytics をサポートしています。

機能	出力データ型	説明
cume_dist()	bigint	値のグループ内の値の累積分布を返します。結果は、ウィンドウパーティションのウィンドウ順序の行の前または前の行とピアの数を、ウィンドウパーティションの行の合計数で割ったものです。したがって、順序内のタイ値は同じ分布値に評価されます。
dense_rank()	bigint	値のグループ内の値のランクを返します。これは rank() に似ていますが、タイ値ではシーケンスにギャップが発生しません。
ntile(n)	bigint	各ウィンドウパーティションの行を 1 から最大 n までの n バケットに分割します。バケット値は最大 1 だけ異なります。パーティション内の行数がバケット数に均等に分

機能	出力データ型	説明
		割されない場合、残りの値は最初のバケットから始めてバケットごとに 1 つずつ分散されます。
percent_rank()	double	値のグループ内の値のパーセンテージランキングを返します。結果は (r - 1) / (n - 1) で、r は行の rank() で、n はウィンドウパーティションの行の合計数です。
rank()	bigint	値のグループ内の値のランクを返します。ランクは 1 に、行とピアリングされていない行の前の行数を加えたものです。したがって、順序の結合値により、シーケンスにギャップが発生します。ランク付けは、ウィンドウパーティションごとに実行されます。
row_number()	bigint	ウィンドウパーティション内の行の順序に従って、1 から始まる各行の一意のシーケンシャル番号を返します。
first_value(x)	〔入力と同じ〕	ウィンドウの最初の値を返します。この関数はウィンドウフレームにスコープされます。関数は、式またはターゲットをパラメータとして受け取ります。

機能	出力データ型	説明
last_value(x)	〔入力と同じ〕	ウィンドウの最後の値を返します。この関数はウィンドウフレームにスコープされます。関数は、式またはターゲットをパラメータとして受け取ります。
nth_value(x、オフセット)	〔入力と同じ〕	ウィンドウの先頭から指定されたオフセットの値を返します。オフセットは 1 から始まります。オフセットは任意のスカラー式にすることができます。オフセットが null であるか、ウィンドウ内の値の数より大きい場合、null が返されます。オフセットがゼロまたは負のエラーです。関数は、式またはターゲットを最初のパラメータとして受け取ります。

機能	出力データ型	説明
lead(x[, offset[, default_value]])	〔入力と同じ〕	ウィンドウ内の現在の行の後にオフセット行の値を返します。オフセットは、現在の行である 0 から始まります。オフセットは任意のスカラー式にすることができます。デフォルトのオフセットは 1 です。オフセットが null またはウィンドウより大きい場合、default_value が返されるか、指定されていない場合は null が返されます。関数は、式またはターゲットを最初のパラメータとして受け取ります。
lag(x[, offset[, default_value]])	〔入力と同じ〕	ウィンドウの現在の行のオフセットが現在の行である 0 で始まる前のオフセット行の値を返します。オフセットは任意のスカラー式にすることができます。デフォルトのオフセットは 1 です。オフセットが null またはウィンドウより大きい場合、default_value が返されるか、指定されていない場合は null が返されます。関数は、式またはターゲットを最初のパラメータとして受け取ります。

サンプルクエリ

このセクションでは、 LiveAnalyticsのクエリ言語の Timestream のユースケースの例を示します。

トピック

- シンプルなクエリ
- 時系列関数を使用したクエリ
- 集計関数を使用したクエリ

シンプルなクエリ

以下は、テーブルに最近追加された 10 個のデータポイントを取得します。

```
SELECT * FROM <database_name>.<table_name>
ORDER BY time DESC
LIMIT 10
```

以下は、特定のメジャーの最も古い 5 つのデータポイントを取得します。

```
SELECT * FROM <database_name>.<table_name>
WHERE measure_name = '<measure_name>'
ORDER BY time ASC
LIMIT 5
```

以下は、ナノ秒の粒度タイムスタンプで機能します。

```
SELECT now() AS time_now
, now() - (INTERVAL '12' HOUR) AS twelve_hour_earlier -- Compatibility with ANSI SQL
, now() - 12h AS also_twelve_hour_earlier -- Convenient time interval literals
, ago(12h) AS twelve_hours_ago -- More convenience with time functionality
, bin(now(), 10m) AS time_binned -- Convenient time binning support
, ago(50ns) AS fifty_ns_ago -- Nanosecond support
, now() + (1h + 50ns) AS hour_fifty_ns_future
```

マルチメジャーレコードの測定値は、列名によって識別されます。単一測定レコードの測定値は によって識別されます。ここでmeasure_value::*<data_type>*、 *<data_type>* はdouble、、、bigintboolean、または のいずれかvarcharですサポートされているデータ型。測定値のモデル化方法の詳細については、「単一テーブル」と「複数テーブル」を参照してください。

以下は、 measure_nameの を持つマルチメジャーレコードspeedから というメジャーの値を取得しますIoTMulti-stats。

```
SELECT speed FROM <database_name>.<table_name> where measure_name = 'IoTMulti-stats'
```

以下は、 measure_name の を持つ単一測定レコードからdouble値を取得しますload。

```
SELECT measure_value::double FROM <database_name>.<table_name> WHERE measure_name =
 'load'
```

時系列関数を使用したクエリ

トピック

- データセットとクエリの例

データセットとクエリの例

の Timestream を使用して LiveAnalytics 、サービスとアプリケーションのパフォーマンスと可用性を理解し、改善できます。以下は、テーブルの例と、そのテーブルで実行されるサンプルクエリです。

テーブルには、EC2インスタンスのCPU使用率やその他のメトリクスなどのテレメトリデータがec2_metrics保存されます。以下の表を表示できます。

時間	region	az	ホスト名	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:00.000000000	us-east-1	us-east-1a	frontend01	cpu_utilization	35.1	null
2019-12-04 19:00:00.000000000	us-east-1	us-east-1a	frontend01	memory_utilization	55.3	null
2019-12-04 19:00:00.000000000	us-east-1	us-east-1a	frontend01	network_bytes_in	null	1,500

時間	region	az	ホスト名	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:00.000000000	us-east-1	us-east-1a	frontend01	network_bytes_out	null	6,700
2019-12-04 19:00:00.000000000	us-east-1	us-east-1b	frontend02	cpu_utilization	38.5	null
2019-12-04 19:00:00.000000000	us-east-1	us-east-1b	frontend02	memory_utilization	58.4	null
2019-12-04 19:00:00.000000000	us-east-1	us-east-1b	frontend02	network_bytes_in	null	23,000
2019-12-04 19:00:00.000000000	us-east-1	us-east-1b	frontend02	network_bytes_out	null	12,000
2019-12-04 19:00:00.000000000	us-east-1	us-east-1c	frontend03	cpu_utilization	45.0	null
2019-12-04 19:00:00.000000000	us-east-1	us-east-1c	frontend03	memory_utilization	65.8	null
2019-12-04 19:00:00.000000000	us-east-1	us-east-1c	frontend03	network_bytes_in	null	15,000
2019-12-04 19:00:00.000000000	us-east-1	us-east-1c	frontend03	network_bytes_out	null	836,000

時間	region	az	ホスト名	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:05.000000000	us-east-1	us-east-1a	frontend01	cpu_utilization	55.2	null
2019-12-04 19:00:05.000000000	us-east-1	us-east-1a	frontend01	memory_utilization	75.0	null
2019-12-04 19:00:05.000000000	us-east-1	us-east-1a	frontend01	network_bytes_in	null	1,245
2019-12-04 19:00:05.000000000	us-east-1	us-east-1a	frontend01	network_bytes_out	null	68,432
2019-12-04 19:00:08.000000000	us-east-1	us-east-1b	frontend02	cpu_utilization	65.6	null
2019-12-04 19:00:08.000000000	us-east-1	us-east-1b	frontend02	memory_utilization	85.3	null
2019-12-04 19:00:08.000000000	us-east-1	us-east-1b	frontend02	network_bytes_in	null	1,245
2019-12-04 19:00:08.000000000	us-east-1	us-east-1b	frontend02	network_bytes_out	null	68,432
2019-12-04 19:00:20.000000000	us-east-1	us-east-1c	frontend03	cpu_utilization	12.1	null

時間	region	az	ホスト名	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:20.000000000	us-east-1	us-east-1c	frontend03	memory_utilization	32.0	null
2019-12-04 19:00:20.000000000	us-east-1	us-east-1c	frontend03	network_bytes_in	null	1,400
2019-12-04 19:00:20.000000000	us-east-1	us-east-1c	frontend03	network_bytes_out	null	345
2019-12-04 19:00:10.000000000	us-east-1	us-east-1a	frontend01	cpu_utilization	15.3	null
2019-12-04 19:00:10.000000000	us-east-1	us-east-1a	frontend01	memory_utilization	35.4	null
2019-12-04 19:00:10.000000000	us-east-1	us-east-1a	frontend01	network_bytes_in	null	23
2019-12-04 19:00:10.000000000	us-east-1	us-east-1a	frontend01	network_bytes_out	null	0
2019-12-04 19:00:16.000000000	us-east-1	us-east-1b	frontend02	cpu_utilization	44.0	null
2019-12-04 19:00:16.000000000	us-east-1	us-east-1b	frontend02	memory_utilization	64.2	null

時間	region	az	ホスト名	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:16.000000000	us-east-1	us-east-1b	frontend02	network_bytes_in	null	1,450
2019-12-04 19:00:16.000000000	us-east-1	us-east-1b	frontend02	network_bytes_out	null	200
2019-12-04 19:00:40.000000000	us-east-1	us-east-1c	frontend03	cpu_utilization	66.4	null
2019-12-04 19:00:40.000000000	us-east-1	us-east-1c	frontend03	memory_utilization	86.3	null
2019-12-04 19:00:40.000000000	us-east-1	us-east-1c	frontend03	network_bytes_in	null	300
2019-12-04 19:00:40.000000000	us-east-1	us-east-1c	frontend03	network_bytes_out	null	423

過去 2 時間の特定のEC2ホストの平均CPU使用率、p90、p95、p99 を見つけます。

```
SELECT region, az, hostname, BIN(time, 15s) AS binned_timestamp,
    ROUND(AVG(measure_value::double), 2) AS avg_cpu_utilization,
    ROUND(APPROX_PERCENTILE(measure_value::double, 0.9), 2) AS p90_cpu_utilization,
    ROUND(APPROX_PERCENTILE(measure_value::double, 0.95), 2) AS p95_cpu_utilization,
    ROUND(APPROX_PERCENTILE(measure_value::double, 0.99), 2) AS p99_cpu_utilization
FROM "sampleDB".DevOps
WHERE measure_name = 'cpu_utilization'
    AND hostname = 'host-Hovjv'
    AND time > ago(2h)
```

```
GROUP BY region, hostname, az, BIN(time, 15s)
ORDER BY binned_timestamp ASC
```

過去 2 時間のフリート全体の平均CPU使用率と比較して、CPU使用率が 10 % 以上高いEC2ホストを特定します。

```
WITH avg_fleet_utilization AS (
    SELECT COUNT(DISTINCT hostname) AS total_host_count, AVG(measure_value::double) AS
 fleet_avg_cpu_utilization
    FROM "sampleDB".DevOps
    WHERE measure_name = 'cpu_utilization'
        AND time > ago(2h)
), avg_per_host_cpu AS (
    SELECT region, az, hostname, AVG(measure_value::double) AS avg_cpu_utilization
    FROM "sampleDB".DevOps
    WHERE measure_name = 'cpu_utilization'
        AND time > ago(2h)
    GROUP BY region, az, hostname
)
SELECT region, az, hostname, avg_cpu_utilization, fleet_avg_cpu_utilization
FROM avg_fleet_utilization, avg_per_host_cpu
WHERE avg_cpu_utilization > 1.1 * fleet_avg_cpu_utilization
ORDER BY avg_cpu_utilization DESC
```

過去 2 時間の特定のEC2ホストの平均CPU使用率を 30 秒間隔でビニングします。

```
SELECT BIN(time, 30s) AS binned_timestamp, ROUND(AVG(measure_value::double), 2) AS
 avg_cpu_utilization
FROM "sampleDB".DevOps
WHERE measure_name = 'cpu_utilization'
    AND hostname = 'host-Hovjv'
    AND time > ago(2h)
GROUP BY hostname, BIN(time, 30s)
ORDER BY binned_timestamp ASC
```

過去 2 時間の特定のEC2ホストの平均CPU使用率を 30 秒間隔でビニングし、線形補間を使用して欠損値を入力します。

```
WITH binned_timeseries AS (
    SELECT hostname, BIN(time, 30s) AS binned_timestamp,
 ROUND(AVG(measure_value::double), 2) AS avg_cpu_utilization
```

```
    FROM "sampleDB".DevOps
    WHERE measure_name = 'cpu_utilization'
        AND hostname = 'host-Hovjv'
        AND time > ago(2h)
    GROUP BY hostname, BIN(time, 30s)
), interpolated_timeseries AS (
    SELECT hostname,
        INTERPOLATE_LINEAR(
            CREATE_TIME_SERIES(binned_timestamp, avg_cpu_utilization),
                SEQUENCE(min(binned_timestamp), max(binned_timestamp), 15s)) AS
 interpolated_avg_cpu_utilization
    FROM binned_timeseries
    GROUP BY hostname
)
SELECT time, ROUND(value, 2) AS interpolated_cpu
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_avg_cpu_utilization)
```

過去 2 時間における特定のEC2ホストの平均CPU使用率を 30 秒間隔でビニングし、最後に繰り越された観測値に基づいて補間を使用して欠落値を入力します。

```
WITH binned_timeseries AS (
    SELECT hostname, BIN(time, 30s) AS binned_timestamp,
 ROUND(AVG(measure_value::double), 2) AS avg_cpu_utilization
    FROM "sampleDB".DevOps
    WHERE measure_name = 'cpu_utilization'
        AND hostname = 'host-Hovjv'
        AND time > ago(2h)
    GROUP BY hostname, BIN(time, 30s)
), interpolated_timeseries AS (
    SELECT hostname,
        INTERPOLATE_LOCF(
            CREATE_TIME_SERIES(binned_timestamp, avg_cpu_utilization),
                SEQUENCE(min(binned_timestamp), max(binned_timestamp), 15s)) AS
 interpolated_avg_cpu_utilization
    FROM binned_timeseries
    GROUP BY hostname
)
SELECT time, ROUND(value, 2) AS interpolated_cpu
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_avg_cpu_utilization)
```

集計関数を使用したクエリ

以下は、集計関数を使用したクエリを示す IoT IoT シナリオデータセットの例です。

トピック

データの例

Timestream を使用すると、1 つ以上のトラックフリートの場所、燃料消費量、速度、負荷容量などの IoT センサーデータを保存および分析して、効果的なフリート管理が可能になります。以下は、トラックの場所、燃料消費量、速度、負荷容量などのテレメトリを保存するテーブル iot_trucks のスキーマとデータの一部です。

時間	truck_id	Make	モデル	フリート	fuel_capacity	load_capacity	measure_name	measure_value::double	measure_value::varchar
2019-12-4 19:00:00.0000000	1234567	GMC	アストロ	[Alpha] (アルファ)	100	500	fuel_reading	65.2	null
2019-12-4 19:00:00.0000000	1234567	GMC	アストロ	[Alpha] (アルファ)	100	500	ロード	400.0	null
2019-12-4 19:00:00.0000000	1234567	GMC	アストロ	[Alpha] (アルファ)	100	500	速度	90.2	null
2019-12-4 19:00:00.0000000	1234567	GMC	アストロ	[Alpha] (アルファ)	100	500	ロケーション	null	47.6062 N、122.3321 W

時間	truck_id	Make	モデル	フリート	fuel_capacity	load_capacity	measure_name	measure_value::double	measure_value::varchar
2019-12-4 19:00:00.0000000	1234567	ケンワース	W900	[Alpha] (アルファ)	150	1,000	fuel_reading	10.1	null
2019-12-4 19:00:00.0000000	1234567	ケンワース	W900	[Alpha] (アルファ)	150	1,000	ロード	950.3	null
2019-12-4 19:00:00.0000000	1234567	ケンワース	W900	[Alpha] (アルファ)	150	1,000	速度	50.8	null
2019-12-4 19:00:00.0000000	1234567	ケンワース	W900	[Alpha] (アルファ)	150	1,000	ロケーション	null	40.7128 度 N、74.0060 度 W

クエリの例

フリート内の各トラックでモニタリングされているすべてのセンサー属性と値のリストを取得します。

```
SELECT
    truck_id,
    fleet,
    fuel_capacity,
    model,
    load_capacity,
    make,
    measure_name
FROM "sampleDB".IoT
```

```
GROUP BY truck_id, fleet, fuel_capacity, model, load_capacity, make, measure_name
```

過去 24 時間のフリート内の各トラックの最新の燃料測定値を取得します。

```
WITH latest_recorded_time AS (
    SELECT
        truck_id,
        max(time) as latest_time
    FROM "sampleDB".IoT
    WHERE measure_name = 'fuel-reading'
    AND time >= ago(24h)
    GROUP BY truck_id
)
SELECT
    b.truck_id,
    b.fleet,
    b.make,
    b.model,
    b.time,
    b.measure_value::double as last_reported_fuel_reading
FROM
latest_recorded_time a INNER JOIN "sampleDB".IoT b
ON a.truck_id = b.truck_id AND b.time = a.latest_time
WHERE b.measure_name = 'fuel-reading'
AND b.time > ago(24h)
ORDER BY b.truck_id
```

過去 48 時間に低燃料 (10 % 未満) で稼働しているトラックを特定します。

```
WITH low_fuel_trucks AS (
    SELECT time, truck_id, fleet, make, model, (measure_value::double/
cast(fuel_capacity as double)*100) AS fuel_pct
    FROM "sampleDB".IoT
    WHERE time >= ago(48h)
    AND (measure_value::double/cast(fuel_capacity as double)*100) < 10
    AND measure_name = 'fuel-reading'
),
other_trucks AS (
SELECT time, truck_id, (measure_value::double/cast(fuel_capacity as double)*100) as
 remaining_fuel
    FROM "sampleDB".IoT
    WHERE time >= ago(48h)
    AND truck_id IN (SELECT truck_id FROM low_fuel_trucks)
```

```
    AND (measure_value::double/cast(fuel_capacity as double)*100) >= 10
    AND measure_name = 'fuel-reading'
),
trucks_that_refuelled AS (
    SELECT a.truck_id
    FROM low_fuel_trucks a JOIN other_trucks b
    ON a.truck_id = b.truck_id AND b.time >= a.time
)
SELECT DISTINCT truck_id, fleet, make, model, fuel_pct
FROM low_fuel_trucks
WHERE truck_id NOT IN (
    SELECT truck_id FROM trucks_that_refuelled
)
```

過去 1 週間の各トラックの平均負荷と最大速度を見つけます。

```
SELECT
    bin(time, 1d) as binned_time,
    fleet,
    truck_id,
    make,
    model,
    AVG(
        CASE WHEN measure_name = 'load' THEN measure_value::double ELSE NULL END
    ) AS avg_load_tons,
    MAX(
        CASE WHEN measure_name = 'speed' THEN measure_value::double ELSE NULL END
    ) AS max_speed_mph
FROM "sampleDB".IoT
WHERE time >= ago(7d)
AND measure_name IN ('load', 'speed')
GROUP BY fleet, truck_id, make, model, bin(time, 1d)
ORDER BY truck_id
```

過去 1 週間のトラックごとの負荷効率を取得します。

```
WITH average_load_per_truck AS (
    SELECT
        truck_id,
        avg(measure_value::double)  AS avg_load
    FROM "sampleDB".IoT
    WHERE measure_name = 'load'
    AND time >= ago(7d)
```

```
    GROUP BY truck_id, fleet, load_capacity, make, model
),
truck_load_efficiency AS (
    SELECT
        a.truck_id,
        fleet,
        load_capacity,
        make,
        model,
        avg_load,
        measure_value::double,
        time,
        (measure_value::double*100)/avg_load as load_efficiency -- ,
 approx_percentile(avg_load_pct, DOUBLE '0.9')
    FROM "sampleDB".IoT a JOIN average_load_per_truck b
    ON a.truck_id = b.truck_id
    WHERE a.measure_name = 'load'
)
SELECT
    truck_id,
    time,
    load_efficiency
FROM truck_load_efficiency
ORDER BY truck_id, time
```

API リファレンス

このセクションでは、Amazon Timestream のAPIリファレンスドキュメントについて説明します。

Timestream には、クエリと書き込みAPIsの 2 つの があります。

- 書き込みAPIを使用すると、テーブルの作成、リソースのタグ付け、Timestream へのレコードの書き込みなどのオペレーションを実行できます。
- クエリAPIを使用すると、クエリオペレーションを実行できます。

> **Note**
>
> 両方に DescribeEndpoints アクションAPIsが含まれます。クエリと書き込みの両方で、DescribeEndpoints アクションは同じです。

API 以下の各項目の詳細と、データ型、一般的なエラー、パラメータを確認できます。

> **Note**
>
> すべての AWS サービスに共通するエラーコードについては、 AWS サポートセクション を参照してください。

トピック

アクション

Amazon Timestream Write では、次のアクションがサポートされています。

- CreateBatchLoadTask
- CreateDatabase
- CreateTable
- DeleteDatabase
- DeleteTable
- DescribeBatchLoadTask
- DescribeDatabase
- DescribeEndpoints
- DescribeTable
- ListBatchLoadTasks
- ListDatabases
- ListTables
- ListTagsForResource
- ResumeBatchLoadTask
- TagResource

- UntagResource
- UpdateDatabase
- UpdateTable
- WriteRecords

Amazon Timestream クエリでは、次のアクションがサポートされています。

- CancelQuery
- CreateScheduledQuery
- DeleteScheduledQuery
- DescribeAccountSettings
- DescribeEndpoints
- DescribeScheduledQuery
- ExecuteScheduledQuery
- ListScheduledQueries
- ListTagsForResource
- PrepareQuery
- Query
- TagResource
- UntagResource
- UpdateAccountSettings
- UpdateScheduledQuery

Amazon Timestream 書き込み

Amazon Timestream Write では、次のアクションがサポートされています。

- CreateBatchLoadTask
- CreateDatabase
- CreateTable
- DeleteDatabase
- DeleteTable

- DescribeBatchLoadTask
- DescribeDatabase
- DescribeEndpoints
- DescribeTable
- ListBatchLoadTasks
- ListDatabases
- ListTables
- ListTagsForResource
- ResumeBatchLoadTask
- TagResource
- UntagResource
- UpdateDatabase
- UpdateTable
- WriteRecords

CreateBatchLoadTask
サービス：Amazon Timestream Write

新しい Timestream バッチロードタスクを作成します。バッチロードタスクは、S3 ロケーションの CSVソースからのデータを処理し、Timestream テーブルに書き込みます。ソースからターゲットへのマッピングは、バッチロードタスクで定義されます。エラーとイベントは S3 ロケーションのレポートに書き込まれます。レポートでは、 AWS KMS キーが指定されていない場合、 SSE_S3 が オプションの場合、レポートは S3 マネージドキーで暗号化されます。それ以外の場合は、エラーがスローされます。詳細については、「AWS マネージドキー」を参照してください。サービスクォータが適用されます。詳細については、「コードサンプル」を参照してください。

リクエストの構文

```
{
   "ClientToken": "string",
   "DataModelConfiguration": {
      "DataModel": {
         "DimensionMappings": [
            {
               "DestinationColumn": "string",
               "SourceColumn": "string"
            }
         ],
         "MeasureNameColumn": "string",
         "MixedMeasureMappings": [
            {
               "MeasureName": "string",
               "MeasureValueType": "string",
               "MultiMeasureAttributeMappings": [
                  {
                     "MeasureValueType": "string",
                     "SourceColumn": "string",
                     "TargetMultiMeasureAttributeName": "string"
                  }
               ],
               "SourceColumn": "string",
               "TargetMeasureName": "string"
            }
         ],
         "MultiMeasureMappings": {
            "MultiMeasureAttributeMappings": [
               {
```

```
                    "MeasureValueType": "string",
                    "SourceColumn": "string",
                    "TargetMultiMeasureAttributeName": "string"
                 }
              ],
              "TargetMultiMeasureName": "string"
           },
           "TimeColumn": "string",
           "TimeUnit": "string"
        },
        "DataModelS3Configuration": {
           "BucketName": "string",
           "ObjectKey": "string"
        }
     },
     "DataSourceConfiguration": {
        "CsvConfiguration": {
           "ColumnSeparator": "string",
           "EscapeChar": "string",
           "NullValue": "string",
           "QuoteChar": "string",
           "TrimWhiteSpace": boolean
        },
        "DataFormat": "string",
        "DataSourceS3Configuration": {
           "BucketName": "string",
           "ObjectKeyPrefix": "string"
        }
     },
     "RecordVersion": number,
     "ReportConfiguration": {
        "ReportS3Configuration": {
           "BucketName": "string",
           "EncryptionOption": "string",
           "KmsKeyId": "string",
           "ObjectKeyPrefix": "string"
        }
     },
     "TargetDatabaseName": "string",
     "TargetTableName": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

ClientToken

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

必須：いいえ

DataModelConfiguration

型: DataModelConfiguration オブジェクト

必須：いいえ

DataSourceConfiguration

バッチロードタスクのデータソースに関する設定の詳細を定義します。

型: DataSourceConfiguration オブジェクト

必須：はい

RecordVersion

型: Long

必須：いいえ

ReportConfiguration

バッチロードタスクの設定をレポートします。これには、エラーレポートの保存先に関する詳細が含まれます。

型: ReportConfiguration オブジェクト

必須：はい

TargetDatabaseName

バッチロードタスクのターゲット Timestream データベース。

型: 文字列

Pattern: [a-zA-Z0-9_.-]+

必須：はい

TargetTableName

バッチロードタスクのターゲットタイムストリームテーブル。

型: 文字列

Pattern: [a-zA-Z0-9_.-]+

必須：はい

レスポンスの構文

```
{
   "TaskId": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

TaskId

バッチロードタスクの ID。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 32 です。

パターン: [A-Z0-9]+

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

ConflictException

Timestream には、既に存在するリソースが含まれているため、このリクエストを処理できませんでした。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、Timestream はこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 500

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

オペレーションは存在しないリソースにアクセスしようとしました。リソースが正しく指定されていないか、そのステータスが ではない可能性がありますACTIVE。

HTTP ステータスコード: 400

ServiceQuotaExceededException

このアカウントのリソースのインスタンスクォータを超えました。

HTTP ステータスコード: 400

ThrottlingException

ユーザーによって行われたリクエストが多すぎて、サービスクォータを超えました。リクエストがスロットリングされました。

HTTP ステータスコード: 400

ValidationException

無効なリクエストまたは不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

CreateDatabase
サービス：Amazon Timestream Write

新しい Timestream データベースを作成します。 AWS KMS キーが指定されていない場合、データベースはアカウントにある Timestream マネージド AWS KMS キーで暗号化されます。詳細については、「AWS マネージドキー」を参照してください。サービスクォータが適用されます。詳細については、「コードサンプル」を参照してください。

リクエストの構文

```
{
   "DatabaseName": "string",
   "KmsKeyId": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次の JSON 形式のデータを受け入れます。

DatabaseName

Timestream データベースの名前。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 256 です。

Pattern: [a-zA-Z0-9_.-]+

必須：はい

KmsKeyId

データベースの AWS KMS キー。 AWS KMS キーが指定されていない場合、データベースはアカウントにある Timestream マネージド AWS KMS キーで暗号化されます。詳細については、「AWS マネージドキー」を参照してください。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：いいえ

Tags

テーブルにラベルを付けるキーと値のペアのリスト。

型: Tag オブジェクトの配列

配列メンバー: 最小数は 0 項目です。最大数は 200 項目です。

必須：いいえ

レスポンスの構文

```
{
   "Database": {
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "KmsKeyId": "string",
      "LastUpdatedTime": number,
      "TableCount": number
   }
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

Database

新しく作成された Timestream データベース。

型: Database オブジェクト

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

ConflictException

Timestream には、既に存在するリソースが含まれているため、このリクエストを処理できませんでした。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、Timestream はこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 500

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ServiceQuotaExceededException

このアカウントのリソースのインスタンスクォータを超えました。

HTTP ステータスコード: 400

ThrottlingException

ユーザーによって行われたリクエストが多すぎて、サービスクォータを超えました。リクエストがスロットリングされました。

HTTP ステータスコード: 400

ValidationException

無効なリクエストまたは不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

CreateTable

サービス：Amazon Timestream Write

アカウントの既存のデータベースに新しいテーブルを追加します。では AWS アカウント、テーブル名が同じデータベースにある場合、各リージョン内で少なくとも一意である必要があります。テーブルが別のデータベースにある場合、同じリージョンに同じテーブル名がある場合があります。テーブルの作成時に、テーブル名、データベース名、保持プロパティを指定する必要があります。サービスクォータが適用されます。詳細については、コードサンプルを参照してください。

リクエストの構文

```
{
   "DatabaseName": "string",
   "MagneticStoreWriteProperties": {
      "EnableMagneticStoreWrites": boolean,
      "MagneticStoreRejectedDataLocation": {
         "S3Configuration": {
            "BucketName": "string",
            "EncryptionOption": "string",
            "KmsKeyId": "string",
            "ObjectKeyPrefix": "string"
         }
      }
   },
   "RetentionProperties": {
      "MagneticStoreRetentionPeriodInDays": number,
      "MemoryStoreRetentionPeriodInHours": number
   },
   "Schema": {
      "CompositePartitionKey": [
         {
            "EnforcementInRecord": "string",
            "Name": "string",
            "Type": "string"
         }
      ]
   },
   "TableName": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
```

```
    ]
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

DatabaseName

Timestream データベースの名前。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 256 です。

Pattern: `[a-zA-Z0-9_.-]+`

必須：はい

MagneticStoreWriteProperties

磁気ストアの書き込みを有効にするときに表に設定するプロパティが含まれます。

型: MagneticStoreWriteProperties オブジェクト

必須：いいえ

RetentionProperties

時系列データをメモリストアとマグネティックストアに保存する必要がある期間。

型: RetentionProperties オブジェクト

必須：いいえ

Schema

テーブルのスキーマ。

型: Schema オブジェクト

必須：いいえ

TableName

Timestream テーブルの名前。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 256 です。

Pattern: [a-zA-Z0-9_.-]+

必須：はい

Tags

テーブルにラベルを付けるキーと値のペアのリスト。

型: Tag オブジェクトの配列

配列メンバー: 最小数は 0 項目です。最大数は 200 項目です。

必須：いいえ

レスポンスの構文

```
{
   "Table": {
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "LastUpdatedTime": number,
      "MagneticStoreWriteProperties": {
         "EnableMagneticStoreWrites": boolean,
         "MagneticStoreRejectedDataLocation": {
            "S3Configuration": {
               "BucketName": "string",
               "EncryptionOption": "string",
               "KmsKeyId": "string",
               "ObjectKeyPrefix": "string"
            }
         }
      },
      "RetentionProperties": {
         "MagneticStoreRetentionPeriodInDays": number,
         "MemoryStoreRetentionPeriodInHours": number
```

```
      },
      "Schema": {
         "CompositePartitionKey": [
            {
               "EnforcementInRecord": "string",
               "Name": "string",
               "Type": "string"
            }
         ]
      },
      "TableName": "string",
      "TableStatus": "string"
   }
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

Table

新しく作成された Timestream テーブル。

型: Table オブジェクト

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

ConflictException

Timestream には、既に存在するリソースが含まれているため、このリクエストを処理できませんでした。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、Timestream はこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 500

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

オペレーションは存在しないリソースにアクセスしようとしました。リソースが正しく指定されていないか、そのステータスが ではない可能性がありますACTIVE。

HTTP ステータスコード: 400

ServiceQuotaExceededException

このアカウントのリソースのインスタンスクォータを超えました。

HTTP ステータスコード: 400

ThrottlingException

ユーザーによって行われたリクエストが多すぎて、サービスクォータを超えました。リクエストがスロットリングされました。

HTTP ステータスコード: 400

ValidationException

無効なリクエストまたは不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 用
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

DeleteDatabase
サービス：Amazon Timestream Write

特定の Timestream データベースを削除します。これは取り返しのつかないオペレーションです。データベースを削除すると、テーブルの時系列データは復元できません。

> **Note**
>
> データベース内のすべてのテーブルを最初に削除する必要があります。削除すると、ValidationException エラーがスローされます。
> 分散再試行の性質上、 オペレーションは成功または を返す可能性があります ResourceNotFoundException。クライアントはそれらと同等であると見なす必要があります。

詳細については、コードサンプルを参照してください。

リクエストの構文

```
{
   "DatabaseName": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

DatabaseName

削除する Timestream データベースの名前。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 256 です。

必須：はい

レスポンス要素

アクションが成功すると、サービスは空のHTTP本文で 200 HTTP レスポンスを送り返します。

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、Timestream はこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 500

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

オペレーションは存在しないリソースにアクセスしようとしました。リソースが正しく指定されていないか、そのステータスが ではない可能性がありますACTIVE。

HTTP ステータスコード: 400

ThrottlingException

ユーザーによって行われたリクエストが多すぎて、サービスクォータを超えました。リクエストがスロットリングされました。

HTTP ステータスコード: 400

ValidationException

無効なリクエストまたは不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

DeleteTable
サービス：Amazon Timestream Write

特定の Timestream テーブルを削除します。これは不可逆的なオペレーションです。Timestream データベーステーブルが削除されると、テーブルに保存されている時系列データは復元できません。

> **Note**
>
> 分散再試行の性質上、 オペレーションは成功または を返す可能性があります ResourceNotFoundException。クライアントはそれらを同等と見なす必要があります。

詳細については、 コードサンプルを参照してください。

リクエストの構文

```
{
   "DatabaseName": "string",
   "TableName": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

DatabaseName

Timestream データベースを削除するデータベースの名前。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 256 です。

必須：はい

TableName

削除する Timestream テーブルの名前。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 256 です。

必須：はい

レスポンス要素

アクションが成功すると、サービスは空のHTTP本文で 200 HTTP レスポンスを送り返します。

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、Timestream はこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 500

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

オペレーションは存在しないリソースにアクセスしようとしました。リソースが正しく指定されていないか、そのステータスが ではない可能性がありますACTIVE。

HTTP ステータスコード: 400

ThrottlingException

ユーザーによって行われたリクエストが多すぎて、サービスクォータを超えました。リクエストがスロットリングされました。

HTTP ステータスコード: 400

ValidationException

無効なリクエストまたは不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 用
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

DescribeBatchLoadTask
サービス：Amazon Timestream Write

設定、マッピング、進行状況、その他の詳細など、バッチロードタスクに関する情報を返します。サービスクォータが適用されます。詳細については、コードサンプルを参照してください。

リクエストの構文

```
{
   "TaskId": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

TaskId

バッチロードタスクの ID。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 32 です。

Pattern: [A-Z0-9]+

必須：はい

レスポンスの構文

```
{
   "BatchLoadTaskDescription": {
      "CreationTime": number,
      "DataModelConfiguration": {
         "DataModel": {
            "DimensionMappings": [
               {
                  "DestinationColumn": "string",
                  "SourceColumn": "string"
               }
```

```
            ],
            "MeasureNameColumn": "string",
            "MixedMeasureMappings": [
               {
                  "MeasureName": "string",
                  "MeasureValueType": "string",
                  "MultiMeasureAttributeMappings": [
                     {
                        "MeasureValueType": "string",
                        "SourceColumn": "string",
                        "TargetMultiMeasureAttributeName": "string"
                     }
                  ],
                  "SourceColumn": "string",
                  "TargetMeasureName": "string"
               }
            ],
            "MultiMeasureMappings": {
               "MultiMeasureAttributeMappings": [
                  {
                     "MeasureValueType": "string",
                     "SourceColumn": "string",
                     "TargetMultiMeasureAttributeName": "string"
                  }
               ],
               "TargetMultiMeasureName": "string"
            },
            "TimeColumn": "string",
            "TimeUnit": "string"
         },
         "DataModelS3Configuration": {
            "BucketName": "string",
            "ObjectKey": "string"
         }
      },
      "DataSourceConfiguration": {
         "CsvConfiguration": {
            "ColumnSeparator": "string",
            "EscapeChar": "string",
            "NullValue": "string",
            "QuoteChar": "string",
            "TrimWhiteSpace": boolean
         },
         "DataFormat": "string",
```

```
            "DataSourceS3Configuration": {
               "BucketName": "string",
               "ObjectKeyPrefix": "string"
            }
         },
         "ErrorMessage": "string",
         "LastUpdatedTime": number,
         "ProgressReport": {
            "BytesMetered": number,
            "FileFailures": number,
            "ParseFailures": number,
            "RecordIngestionFailures": number,
            "RecordsIngested": number,
            "RecordsProcessed": number
         },
         "RecordVersion": number,
         "ReportConfiguration": {
            "ReportS3Configuration": {
               "BucketName": "string",
               "EncryptionOption": "string",
               "KmsKeyId": "string",
               "ObjectKeyPrefix": "string"
            }
         },
         "ResumableUntil": number,
         "TargetDatabaseName": "string",
         "TargetTableName": "string",
         "TaskId": "string",
         "TaskStatus": "string"
      }
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

BatchLoadTaskDescription

バッチロードタスクの説明。

型: BatchLoadTaskDescription オブジェクト

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、Timestream はこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 500

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

オペレーションは存在しないリソースにアクセスしようとしました。リソースが正しく指定されていないか、そのステータスが ではない可能性がありますACTIVE。

HTTP ステータスコード: 400

ThrottlingException

ユーザーによって行われたリクエストが多すぎて、サービスクォータを超えました。リクエストがスロットリングされました。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用

- AWS SDK Go v2 用
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

DescribeDatabase
サービス：Amazon Timestream Write

データベース名、データベースが作成された時刻、データベース内で見つかったテーブルの合計数など、データベースに関する情報を返します。サービスクォータが適用されます。詳細については、コードサンプルを参照してください。

リクエストの構文

```
{
   "DatabaseName": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次の JSON 形式のデータを受け入れます。

DatabaseName

Timestream データベースの名前。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 256 です。

必須：はい

レスポンスの構文

```
{
   "Database": {
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "KmsKeyId": "string",
      "LastUpdatedTime": number,
      "TableCount": number
   }
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

Database

Timestream テーブルの名前。

型: Database オブジェクト

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、Timestream はこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 500

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

オペレーションは存在しないリソースにアクセスしようとしました。リソースが正しく指定されていないか、そのステータスが ではない可能性がありますACTIVE。

HTTP ステータスコード: 400

ThrottlingException

ユーザーによって行われたリクエストが多すぎて、サービスクォータを超えました。リクエストがスロットリングされました。

HTTP ステータスコード: 400

ValidationException

無効なリクエストまたは不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

DescribeEndpoints
サービス：Amazon Timestream Write

Timestream APIコールを行う使用可能なエンドポイントのリストを返します。このAPIオペレーションは、書き込み とクエリ の両方で使用できますAPIs。

Timestream SDKsは、サービスエンドポイントの管理やマッピングなど、サービスのアーキテクチャと透過的に連携するように設計されているため、 の場合を除き、このAPIオペレーションを使用することはお勧めしません。

- VPC Timestream でエンドポイント (AWS PrivateLink) を使用している
- アプリケーションが、まだSDKサポートされていないプログラミング言語を使用している
- クライアント側の実装をより適切に制御する必要がある

を使用および実装する方法と時期の詳細については DescribeEndpoints、「エンドポイント検出パターン」を参照してください。

レスポンスの構文

```
{
   "Endpoints": [
      {
         "Address": "string",
         "CachePeriodInMinutes": number
      }
   ]
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

Endpoints

Endpoints オブジェクトは、DescribeEndpointsリクエストが行われたときに返されます。

型: Endpoint オブジェクトの配列

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InternalServerException

内部サーバーエラーのため、Timestream はこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 500

ThrottlingException

ユーザーによって行われたリクエストが多すぎて、サービスクォータを超えました。リクエストがスロットリングされました。

HTTP ステータスコード: 400

ValidationException

無効なリクエストまたは不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 用
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

DescribeTable
サービス：Amazon Timestream Write

テーブル名、データベース名、メモリストアの保持期間、マグネティックストアなど、テーブルに関する情報を返します。サービスクォータが適用されます。詳細については、コードサンプルを参照してください。

リクエストの構文

```
{
   "DatabaseName": "string",
   "TableName": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

DatabaseName

Timestream データベースの名前。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 256 です。

必須：はい

TableName

Timestream テーブルの名前。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 256 です。

必須：はい

レスポンスの構文

```
{
```

```
   "Table": {
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "LastUpdatedTime": number,
      "MagneticStoreWriteProperties": {
         "EnableMagneticStoreWrites": boolean,
         "MagneticStoreRejectedDataLocation": {
            "S3Configuration": {
               "BucketName": "string",
               "EncryptionOption": "string",
               "KmsKeyId": "string",
               "ObjectKeyPrefix": "string"
            }
         }
      },
      "RetentionProperties": {
         "MagneticStoreRetentionPeriodInDays": number,
         "MemoryStoreRetentionPeriodInHours": number
      },
      "Schema": {
         "CompositePartitionKey": [
            {
               "EnforcementInRecord": "string",
               "Name": "string",
               "Type": "string"
            }
         ]
      },
      "TableName": "string",
      "TableStatus": "string"
   }
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

Table

Timestream テーブル。

型: Table オブジェクト

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、Timestream はこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 500

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

オペレーションは存在しないリソースにアクセスしようとしました。リソースが正しく指定されていないか、そのステータスが ではない可能性がありますACTIVE。

HTTP ステータスコード: 400

ThrottlingException

ユーザーによって行われたリクエストが多すぎて、サービスクォータを超えました。リクエストがスロットリングされました。

HTTP ステータスコード: 400

ValidationException

無効なリクエストまたは不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

ListBatchLoadTasks
サービス：Amazon Timestream Write

バッチロードタスクのリストと、名前、ステータス、タスクがいつまで再開可能になるかなどの詳細を提供します。詳細については、コードサンプルを参照してください。

リクエストの構文

```
{
   "MaxResults": number,
   "NextToken": "string",
   "TaskStatus": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次の JSON 形式のデータを受け入れます。

MaxResults

出力で返される項目の合計数。使用可能な項目の総数が指定された値を超える場合、出力には NextToken が表示されます。ページ分割を再開するには、 NextToken 値を後続のAPI呼び出しの引数として指定します。

型: 整数

有効範囲: 最小値 は 1 です。最大値は 100 です。

必須：いいえ

NextToken

ページ分割を始める場所を指定するトークン。これは、以前に切り捨てられたレスポンス NextToken からの です。

型: 文字列

必須: いいえ

TaskStatus

バッチロードタスクのステータス。

型: 文字列

有効な値: CREATED | IN_PROGRESS | FAILED | SUCCEEDED | PROGRESS_STOPPED | PENDING_RESUME

必須：いいえ

レスポンスの構文

```
{
   "BatchLoadTasks": [
      {
         "CreationTime": number,
         "DatabaseName": "string",
         "LastUpdatedTime": number,
         "ResumableUntil": number,
         "TableName": "string",
         "TaskId": "string",
         "TaskStatus": "string"
      }
   ],
   "NextToken": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

BatchLoadTasks

バッチロードタスクの詳細のリスト。

型: BatchLoadTask オブジェクトの配列

NextToken

ページ分割を始める場所を指定するトークン。次の を指定します ListBatchLoadTasksRequest。

型: 文字列

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、Timestream はこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 500

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ThrottlingException

ユーザーによって行われたリクエストが多すぎて、サービスクォータを超えました。リクエストがスロットリングされました。

HTTP ステータスコード: 400

ValidationException

無効なリクエストまたは不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET

- AWS SDK C++ 用
- AWS SDK Go v2 用
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

ListDatabases
サービス：Amazon Timestream Write

Timestream データベースのリストを返します。サービスクォータが適用されます。詳細については、 コードサンプルを参照してください。

リクエストの構文

```
{
   "MaxResults": number,
   "NextToken": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

MaxResults

出力で返される項目の合計数。使用可能な項目の総数が指定された値より大きい場合、出力には NextToken が指定されます。ページ分割を再開するには、 NextToken 値を後続のAPI呼び出しの引数として指定します。

型: 整数

有効範囲: 最小値 は 1 です。最大値は 20 です。

必須：いいえ

NextToken

ページ分割トークン。ページ分割を再開するには、 NextToken 値を後続のAPI呼び出しの引数として指定します。

型: 文字列

必須: いいえ

レスポンスの構文

```
{
   "Databases": [
      {
         "Arn": "string",
         "CreationTime": number,
         "DatabaseName": "string",
         "KmsKeyId": "string",
         "LastUpdatedTime": number,
         "TableCount": number
      }
   ],
   "NextToken": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

Databases

データベース名のリスト。

型: Database オブジェクトの配列

NextToken

ページ分割トークン。このパラメータは、レスポンスが切り捨てられたときに返されます。

型: 文字列

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、Timestream はこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 500

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ThrottlingException

ユーザーによって行われたリクエストが多すぎて、サービスクォータを超えました。リクエストがスロットリングされました。

HTTP ステータスコード: 400

ValidationException

無効なリクエストまたは不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

ListTables
サービス：Amazon Timestream Write

テーブルのリストと、各テーブルの名前、ステータス、保持プロパティを提供します。詳細については、 コードサンプルを参照してください。

リクエストの構文

```
{
   "DatabaseName": "string",
   "MaxResults": number,
   "NextToken": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次の JSON 形式のデータを受け入れます。

DatabaseName

Timestream データベースの名前。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 256 です。

必須：いいえ

MaxResults

出力で返される項目の合計数。使用可能な項目の総数が指定された値を超える場合、出力には NextToken が表示されます。ページ分割を再開するには、 NextToken 値を後続のAPI呼び出しの引数として指定します。

型: 整数

有効範囲: 最小値 は 1 です。最大値は 20 です。

必須：いいえ

NextToken

ページ分割トークン。ページ分割を再開するには、 NextToken 値を後続のAPI呼び出しの引数として指定します。

型: 文字列

必須: いいえ

レスポンスの構文

```
{
   "NextToken": "string",
   "Tables": [
      {
         "Arn": "string",
         "CreationTime": number,
         "DatabaseName": "string",
         "LastUpdatedTime": number,
         "MagneticStoreWriteProperties": {
            "EnableMagneticStoreWrites": boolean,
            "MagneticStoreRejectedDataLocation": {
               "S3Configuration": {
                  "BucketName": "string",
                  "EncryptionOption": "string",
                  "KmsKeyId": "string",
                  "ObjectKeyPrefix": "string"
               }
            }
         },
         "RetentionProperties": {
            "MagneticStoreRetentionPeriodInDays": number,
            "MemoryStoreRetentionPeriodInHours": number
         },
         "Schema": {
            "CompositePartitionKey": [
               {
                  "EnforcementInRecord": "string",
                  "Name": "string",
                  "Type": "string"
               }
            ]
         },
```

```
            "TableName": "string",
            "TableStatus": "string"
        }
    ]
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

NextToken

ページ分割を始める場所を指定するトークン。これは、以前に切り捨てられたレスポンス NextToken からの です。

型: 文字列

Tables

テーブルのリスト。

型: Table オブジェクトの配列

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、Timestream はこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 500

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

オペレーションは存在しないリソースにアクセスしようとしました。リソースが正しく指定されていないか、そのステータスが ではない可能性がありますACTIVE。

HTTP ステータスコード: 400

ThrottlingException

ユーザーによって行われたリクエストが多すぎて、サービスクォータを超えました。リクエストがスロットリングされました。

HTTP ステータスコード: 400

ValidationException

無効なリクエストまたは不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

ListTagsForResource
サービス：Amazon Timestream Write

Timestream リソースのすべてのタグを一覧表示します。

リクエストの構文

```
{
   "ResourceARN": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

ResourceARN

リストされるタグを持つ Timestream リソース。この値は Amazon リソースネーム () ですARN。

型: 文字列

長さの制限：最小長は 1 です。最大長は 1011 です。

必須：はい

レスポンスの構文

```
{
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

Tags

Timestream リソースに現在関連付けられているタグ。

型: Tag オブジェクトの配列

配列メンバー: 最小数は 0 項目です。最大数は 200 項目です。

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

オペレーションは存在しないリソースにアクセスしようとしました。リソースが正しく指定されていないか、そのステータスが ではない可能性がありますACTIVE。

HTTP ステータスコード: 400

ThrottlingException

ユーザーによって行われたリクエストが多すぎて、サービスクォータを超えました。リクエストがスロットリングされました。

HTTP ステータスコード: 400

ValidationException

無効なリクエストまたは不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

ResumeBatchLoadTask
サービス：Amazon Timestream Write

リクエストの構文

```
{
   "TaskId": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

TaskId

再開するバッチロードタスクの ID。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 32 です。

Pattern: [A-Z0-9]+

必須：はい

レスポンス要素

アクションが成功すると、サービスは空のHTTP本文で 200 HTTP レスポンスを送り返します。

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、Timestream はこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 500

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

オペレーションは存在しないリソースにアクセスしようとしました。リソースが正しく指定されていないか、そのステータスが ではない可能性がありますACTIVE。

HTTP ステータスコード: 400

ThrottlingException

ユーザーによって行われたリクエストが多すぎて、サービスクォータを超えました。リクエストがスロットリングされました。

HTTP ステータスコード: 400

ValidationException

無効なリクエストまたは不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3

- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

TagResource
サービス：Amazon Timestream Write

一連のタグを Timestream リソースに関連付けます。その後、これらのユーザー定義タグをアクティブ化して、コスト配分の追跡のために請求とコスト管理コンソールに表示されるようにすることができます。

リクエストの構文

```
{
   "ResourceARN": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

ResourceARN

タグを追加する Timestream リソースを識別します。この値は Amazon リソースネーム () です ARN。

型: 文字列

長さの制限：最小長は 1 です。最大長は 1011 です。

必須：はい

Tags

Timestream リソースに割り当てるタグ。

型: Tag オブジェクトの配列

配列メンバー: 最小数は 0 項目です。最大数は 200 項目です。

必須：はい

レスポンス要素

アクションが成功すると、サービスは空のHTTP本文で 200 HTTP レスポンスを送り返します。

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

オペレーションは存在しないリソースにアクセスしようとしました。リソースが正しく指定されていないか、そのステータスが ではない可能性がありますACTIVE。

HTTP ステータスコード: 400

ServiceQuotaExceededException

このアカウントのリソースのインスタンスクォータを超えました。

HTTP ステータスコード: 400

ThrottlingException

ユーザーによって行われたリクエストが多すぎて、サービスクォータを超えました。リクエストがスロットリングされました。

HTTP ステータスコード: 400

ValidationException

無効なリクエストまたは不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

UntagResource
サービス：Amazon Timestream Write

Timestream リソースからタグの関連付けを削除します。

リクエストの構文

```
{
   "ResourceARN": "string",
   "TagKeys": [ "string" ]
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

ResourceARN

タグを削除する Timestream リソース。この値は Amazon リソースネーム () ですARN。

型: 文字列

長さの制限：最小長は 1 です。最大長は 1011 です。

必須：はい

TagKeys

タグキーのリスト。このリストのメンバーであるキーを持つリソースの既存のタグは、Timestream リソースから削除されます。

型: 文字列の配列

配列メンバー：最小数は 0 項目です。最大数は 200 項目です。

長さの制限：最小長は 1 です。最大長は 128 です。

必須：はい

レスポンス要素

アクションが成功すると、サービスは空のHTTP本文で 200 HTTP レスポンスを送り返します。

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

オペレーションは存在しないリソースにアクセスしようとしました。リソースが正しく指定されていないか、そのステータスが ではない可能性がありますACTIVE。

HTTP ステータスコード: 400

ServiceQuotaExceededException

このアカウントのリソースのインスタンスクォータを超えました。

HTTP ステータスコード: 400

ThrottlingException

ユーザーによって行われたリクエストが多すぎて、サービスクォータを超えました。リクエストがスロットリングされました。

HTTP ステータスコード: 400

ValidationException

無効なリクエストまたは不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

UpdateDatabase
サービス：Amazon Timestream Write

既存のデータベースの AWS KMS キーを変更します。データベースを更新するときは、使用する新しい AWS KMS キー () のデータベース名と識別子を指定する必要がありますKmsKeyId。同時UpdateDatabaseリクエストがある場合、最初のライターが勝利します。

詳細については、コードサンプルを参照してください。

リクエストの構文

```
{
   "DatabaseName": "string",
   "KmsKeyId": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

DatabaseName

データベースの名前。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 256 です。

必須：はい

KmsKeyId

データベースに保存されているデータの暗号化に使用される新しい AWS KMS キー (KmsKeyId) の識別子。データベースにKmsKeyId現在登録されている がリクエストKmsKeyIdの と同じ場合、更新はありません。

は、次のいずれかKmsKeyIdを使用して指定できます。

- キー ID: 1234abcd-12ab-34cd-56ef-1234567890ab
- キー ARN: arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

- エイリアス名: alias/ExampleAlias
- エイリアス ARN: arn:aws:kms:us-east-1:111122223333:alias/ExampleAlias

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：はい

レスポンスの構文

```
{
   "Database": {
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "KmsKeyId": "string",
      "LastUpdatedTime": number,
      "TableCount": number
   }
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

Database

テーブルの最上位コンテナ。データベースとテーブルは、Amazon Timestream の基本的な管理概念です。データベース内のすべてのテーブルは、同じ AWS KMS キーで暗号化されます。

型: Database オブジェクト

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、Timestream はこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 500

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

オペレーションは存在しないリソースにアクセスしようとしました。リソースが正しく指定されていないか、そのステータスが ではない可能性がありますACTIVE。

HTTP ステータスコード: 400

ServiceQuotaExceededException

このアカウントのリソースのインスタンスクォータを超えました。

HTTP ステータスコード: 400

ThrottlingException

ユーザーによって行われたリクエストが多すぎて、サービスクォータを超えました。リクエストがスロットリングされました。

HTTP ステータスコード: 400

ValidationException

無効なリクエストまたは不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス

- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

UpdateTable
サービス：Amazon Timestream Write

Timestream テーブルのメモリストアとマグネティックストアの保持期間を変更します。保持期間の変更はすぐに有効になることに注意してください。例えば、メモリストアの保持期間が最初に 2 時間に設定され、その後 24 時間に変更された場合、メモリストアは 24 時間のデータを保持できますが、この変更が行われた 22 時間後に 24 時間のデータが入力されます。Timestream は、メモリストアに入力するために磁気ストアからデータを取得しません。

詳細については、コードサンプルを参照してください。

リクエストの構文

```
{
   "DatabaseName": "string",
   "MagneticStoreWriteProperties": {
      "EnableMagneticStoreWrites": boolean,
      "MagneticStoreRejectedDataLocation": {
         "S3Configuration": {
            "BucketName": "string",
            "EncryptionOption": "string",
            "KmsKeyId": "string",
            "ObjectKeyPrefix": "string"
         }
      }
   },
   "RetentionProperties": {
      "MagneticStoreRetentionPeriodInDays": number,
      "MemoryStoreRetentionPeriodInHours": number
   },
   "Schema": {
      "CompositePartitionKey": [
         {
            "EnforcementInRecord": "string",
            "Name": "string",
            "Type": "string"
         }
      ]
   },
   "TableName": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

DatabaseName

Timestream データベースの名前。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 256 です。

必須：はい

MagneticStoreWriteProperties

磁気ストアの書き込みを有効にするときに表に設定するプロパティが含まれます。

型: MagneticStoreWriteProperties オブジェクト

必須：いいえ

RetentionProperties

メモリストアとマグネティックストアの保持期間。

型: RetentionProperties オブジェクト

必須：いいえ

Schema

テーブルのスキーマ。

型: Schema オブジェクト

必須：いいえ

TableName

Timestream テーブルの名前。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 256 です。

必須：はい

レスポンスの構文

```
{
   "Table": {
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "LastUpdatedTime": number,
      "MagneticStoreWriteProperties": {
         "EnableMagneticStoreWrites": boolean,
         "MagneticStoreRejectedDataLocation": {
            "S3Configuration": {
               "BucketName": "string",
               "EncryptionOption": "string",
               "KmsKeyId": "string",
               "ObjectKeyPrefix": "string"
            }
         }
      },
      "RetentionProperties": {
         "MagneticStoreRetentionPeriodInDays": number,
         "MemoryStoreRetentionPeriodInHours": number
      },
      "Schema": {
         "CompositePartitionKey": [
            {
               "EnforcementInRecord": "string",
               "Name": "string",
               "Type": "string"
            }
         ]
      },
      "TableName": "string",
      "TableStatus": "string"
   }
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

Table

更新された Timestream テーブル。

型: Table オブジェクト

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、Timestream はこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 500

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

オペレーションは存在しないリソースにアクセスしようとしました。リソースが正しく指定されていないか、そのステータスが ではない可能性がありますACTIVE。

HTTP ステータスコード: 400

ThrottlingException

ユーザーによって行われたリクエストが多すぎて、サービスクォータを超えました。リクエストがスロットリングされました。

HTTP ステータスコード: 400

ValidationException

無効なリクエストまたは不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

WriteRecords
サービス：Amazon Timestream Write

Timestream に時系列データを書き込むことができます。システムに挿入する単一のデータポイントまたはデータポイントのバッチを指定できます。Timestream は、データベースへの書き込みを呼び出すときに指定したデータポイントのディメンション名とデータ型に基づいて、Timestream テーブルの列名とデータ型を自動的に検出する柔軟なスキーマを提供します。

Timestream は、最終的な整合性リードセマンティクスをサポートします。つまり、データのバッチを Timestream に書き込んだ直後にデータをクエリすると、クエリ結果は最近完了した書き込みオペレーションの結果を反映しない可能性があります。結果には、古いデータが含まれている場合もあります。短時間後にクエリリクエストを繰り返すと、結果は最新のデータを返します。サービスクォータが適用されます。

詳細については、コードサンプルを参照してください。

アップサート

WriteRecords リクエストで Versionパラメータを使用して、データポイントを更新できます。Timestream は、各レコードでバージョン番号を追跡します。 は、リクエスト内のレコードに指定されていない場合、Versionデフォルトで 1 になります。Timestream は、既存のレコードの測定値を、そのレコードのVersion数値が高い書き込みリクエストVersionを受信すると、そのレコードの測定値とともに更新します。測定値が既存のレコードの測定値と同じである更新リクエストを受信すると、Timestream は既存の の値よりも大きい場合でも Versionを更新しますVersion。の値がVersion継続的に増加している限り、データポイントを必要な回数だけ更新できます。

例えば、リクエストVersionに を指定せずに新しいレコードを書き込むとします。Timestream はこのレコードを保存し、 Version を に設定します1。ここで、同じレコードのWriteRecordsリクエストを別のメジャー値で更新しようとしたが、以前のように を指定しないとしますVersion。この場合、更新されたレコードのバージョンがバージョン の既存の値を超えないRejectedRecordsExceptionため、Timestream はこの更新を で拒否します。

ただし、 Versionを に設定して更新リクエストを再送信すると2、Timestream はレコードの値の更新に成功し、 Versionは に設定されます2。次に、同じレコードと同じメジャー値で をVersion設定したWriteRecordsリクエストを送信したとします3。この場合、Timestream は Version にのみ更新されます3。それ以降の更新では、 より大きいバージョン番号を送信する必要があります。そうしないと3、更新リクエストは を受け取りますRejectedRecordsException。

リクエストの構文

```
{
   "CommonAttributes": {
      "Dimensions": [
         {
            "DimensionValueType": "string",
            "Name": "string",
            "Value": "string"
         }
      ],
      "MeasureName": "string",
      "MeasureValue": "string",
      "MeasureValues": [
         {
            "Name": "string",
            "Type": "string",
            "Value": "string"
         }
      ],
      "MeasureValueType": "string",
      "Time": "string",
      "TimeUnit": "string",
      "Version": number
   },
   "DatabaseName": "string",
   "Records": [
      {
         "Dimensions": [
            {
               "DimensionValueType": "string",
               "Name": "string",
               "Value": "string"
            }
         ],
         "MeasureName": "string",
         "MeasureValue": "string",
         "MeasureValues": [
            {
               "Name": "string",
               "Type": "string",
               "Value": "string"
            }
         ],
```

```
            "MeasureValueType": "string",
            "Time": "string",
            "TimeUnit": "string",
            "Version": number
        }
    ],
    "TableName": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次の JSON 形式のデータを受け入れます。

CommonAttributes

リクエスト内のすべてのレコードで共有される一般的なメジャー、ディメンション、時間、バージョン属性を含むレコード。指定された測定属性とディメンション属性は、データが Timestream に書き込まれると、レコードオブジェクトの測定属性とディメンション属性とマージされます。ディメンションが重複しないか、 ValidationExceptionがスローされます。つまり、レコードには一意の名前のディメンションが含まれている必要があります。

型: Record オブジェクト

必須：いいえ

DatabaseName

Timestream データベースの名前。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 256 です。

必須：はい

Records

各時系列データポイントの一意のメジャー、ディメンション、時間、バージョン属性を含むレコードの配列。

型: Record オブジェクトの配列

配列メンバー: 最小数は 1 項目です。最大数は 100 項目です。

必須：はい

TableName

Timestream テーブルの名前。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 256 です。

必須：はい

レスポンスの構文

```
{
   "RecordsIngested": {
      "MagneticStore": number,
      "MemoryStore": number,
      "Total": number
   }
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

RecordsIngested

このリクエストによって取り込まれたレコードに関する情報。

型: RecordsIngested オブジェクト

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、Timestream はこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 500

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

RejectedRecordsException

WriteRecords は、次の場合にこの例外をスローします。

- 同じディメンション、タイムスタンプ、メジャー名を持つ複数のレコードがあるが、次の重複データを含むレコード。
  - 測定値が異なる
  - バージョンがリクエストに存在しないか、新しいレコードのバージョン値が既存の値以下です

  この場合、Timestream がデータを拒否すると、`RejectedRecords`レスポンスの `ExistingVersion` フィールドには現在のレコードのバージョンが表示されます。更新を強制するには、レコードセットのバージョンを より大きい値に設定したリクエストを再送信できます`ExistingVersion`。
- メモリストアの保持期間外のタイムスタンプを含むレコード。
- Timestream で定義された制限を超えるディメンションまたはメジャーを持つレコード。

詳細については、「Amazon Timestream デベロッパーガイド」の「クォータ」を参照してください。

HTTP ステータスコード: 400

ResourceNotFoundException

オペレーションは存在しないリソースにアクセスしようとしました。リソースが正しく指定されていないか、そのステータスが ではない可能性がありますACTIVE。

HTTP ステータスコード: 400

ThrottlingException

ユーザーによって行われたリクエストが多すぎて、サービスクォータを超えました。リクエストがスロットリングされました。

HTTP ステータスコード: 400

ValidationException

無効なリクエストまたは不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 用
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

Amazon Timestream クエリ

Amazon Timestream クエリでは、次のアクションがサポートされています。

- CancelQuery
- CreateScheduledQuery
- DeleteScheduledQuery
- DescribeAccountSettings
- DescribeEndpoints

- DescribeScheduledQuery
- ExecuteScheduledQuery
- ListScheduledQueries
- ListTagsForResource
- PrepareQuery
- Query
- TagResource
- UntagResource
- UpdateAccountSettings
- UpdateScheduledQuery

CancelQuery
サービス：Amazon Timestream Query

発行されたクエリをキャンセルします。キャンセルは、キャンセルリクエストが発行される前にクエリの実行が完了していない場合にのみ提供されます。キャンセルは冪等な操作であるため、後続のキャンセルリクエストは を返しCancellationMessage、クエリがすでにキャンセルされたことを示します。詳細については、コードサンプルを参照してください。

リクエストの構文

```
{
   "QueryId": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

QueryId

キャンセルする必要があるクエリの ID。 QueryIDはクエリ結果の一部として返されます。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

Pattern: [a-zA-Z0-9]+

必須：はい

レスポンスの構文

```
{
   "CancellationMessage": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

CancellationMessage

CancellationMessage は、 で指定されたクエリのCancelQueryリクエストQueryIdが既に発行されたときに返されます。

型: 文字列

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、サービスはこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 400

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ThrottlingException

リクエストのスロットリングにより、リクエストが拒否されました。

HTTP ステータスコード: 400

ValidationException

無効または不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 用
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

CreateScheduledQuery
サービス：Amazon Timestream Query

設定されたスケジュールでユーザーに代わって実行されるスケジュールされたクエリを作成します。Timestream は、ScheduledQueryExecutionRoleArn パラメータの一部として提供される実行ロールを引き受けて、クエリを実行します。NotificationConfiguration パラメータを使用して、スケジュールされたクエリ操作の通知を設定できます。

リクエストの構文

```
{
   "ClientToken": "string",
   "ErrorReportConfiguration": {
      "S3Configuration": {
         "BucketName": "string",
         "EncryptionOption": "string",
         "ObjectKeyPrefix": "string"
      }
   },
   "KmsKeyId": "string",
   "Name": "string",
   "NotificationConfiguration": {
      "SnsConfiguration": {
         "TopicArn": "string"
      }
   },
   "QueryString": "string",
   "ScheduleConfiguration": {
      "ScheduleExpression": "string"
   },
   "ScheduledQueryExecutionRoleArn": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ],
   "TargetConfiguration": {
      "TimestreamConfiguration": {
         "DatabaseName": "string",
         "DimensionMappings": [
            {
               "DimensionValueType": "string",
```

```
                "Name": "string"
            }
        ],
        "MeasureNameColumn": "string",
        "MixedMeasureMappings": [
            {
                "MeasureName": "string",
                "MeasureValueType": "string",
                "MultiMeasureAttributeMappings": [
                    {
                        "MeasureValueType": "string",
                        "SourceColumn": "string",
                        "TargetMultiMeasureAttributeName": "string"
                    }
                ],
                "SourceColumn": "string",
                "TargetMeasureName": "string"
            }
        ],
        "MultiMeasureMappings": {
            "MultiMeasureAttributeMappings": [
                {
                    "MeasureValueType": "string",
                    "SourceColumn": "string",
                    "TargetMultiMeasureAttributeName": "string"
                }
            ],
            "TargetMultiMeasureName": "string"
        },
        "TableName": "string",
        "TimeColumn": "string"
    }
  }
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次の JSON 形式のデータを受け入れます。

ClientToken

を使用すると、 CreateScheduledQuery 冪等性を ClientToken 呼び出すことができます。つまり、同じリクエストを繰り返し行うと、同じ結果になります。複数の同じ CreateScheduledQuery リクエストを行うと、1 つのリクエストを行うのと同じ効果があります。

- CreateScheduledQuery が なしで呼び出された場合`ClientToken`、クエリは`ClientToken`ユーザーに代わって SDKを生成します。
- 8 時間後、同じ `ClientToken` を持つリクエストは新しいリクエストとして処理されます。

型: 文字列

長さの制限: 最小長 32。最大長は 128 です。

必須：いいえ

ErrorReportConfiguration

エラーレポートの構成。エラーレポートは、クエリ結果を書き込むときに問題が発生した場合に生成されます。

型: ErrorReportConfiguration オブジェクト

必須：はい

KmsKeyId

スケジュールされたクエリリソースの保管時の暗号化に使用される Amazon KMSキー。Amazon KMSキーが指定されていない場合、スケジュールされたクエリリソースは Timestream が所有する Amazon KMSキーで暗号化されます。KMS キーを指定するには、キー ID、キー ARN、エイリアス名、またはエイリアス を使用しますARN。エイリアス名を使用する場合は、名前にプレフィックスとして alias/ を付けます。

ErrorReportConfiguration を暗号化タイプ`SSE_KMS`として使用する場合、保管中のエラーレポートの暗号化にも同じ KmsKeyId が使用されます。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：いいえ

Name

スケジュールされたクエリの名前。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

Pattern: `[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

必須：はい

NotificationConfiguration

スケジュールされたクエリの通知構成。Timestream は、クエリの実行が終了したとき、状態が更新されたとき、または削除されたときに通知を送信します。

型: NotificationConfiguration オブジェクト

必須：はい

QueryString

実行するクエリ文字列。パラメータ名は、クエリ文字列 @ 文字とその後に識別子を続けることで指定できます。名前付きパラメータ `@scheduled_runtime` は予約されており、クエリの実行がスケジュールされている時刻を取得するためにクエリ内で使用できます。

ScheduleConfiguration パラメータに従って計算されたタイムスタンプは、クエリ実行ごとに`@scheduled_runtime`パラメータの値になります。例えば、2021-12-01 00:00:00 にスケジュールされたクエリのインスタンスを実行するとします。この場合、クエリを呼び出すと、`@scheduled_runtime` パラメータはタイムスタンプ 2021-12-01 00:00:00 に初期化されます。

型: 文字列

長さの制限：最小長は 1 です。最大長は 262,144 です。

必須：はい

ScheduleConfiguration

クエリのスケジュール設定。

型: ScheduleConfiguration オブジェクト

必須：はい

ScheduledQueryExecutionRoleArn

スケジュールされたクエリの実行時に Timestream ARNが引き受けるIAMロールの 。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：はい

Tags

スケジュールされたクエリにラベルを付けるためのキーと値のペアのリスト。

型: Tag オブジェクトの配列

配列メンバー: 最小数は 0 項目です。最大数は 200 項目です。

必須：いいえ

TargetConfiguration

クエリの結果の書き込みに使用される設定。

型: TargetConfiguration オブジェクト

必須: いいえ

レスポンスの構文

```
{
   "Arn": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

Arn

ARN 作成されたスケジュールされたクエリ。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

ConflictException

キャンセルされたクエリの結果をポーリングできません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、サービスはこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 400

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ServiceQuotaExceededException

サービスクォータを超えました。

HTTP ステータスコード: 400

ThrottlingException

リクエストのスロットリングにより、リクエストが拒否されました。

HTTP ステータスコード: 400

ValidationException

無効または不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ の場合
- AWS SDK Go v2 用
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

DeleteScheduledQuery
サービス：Amazon Timestream Query

スケジュールされた特定のクエリを削除します。これは不可逆的なオペレーションです。

リクエストの構文

```
{
   "ScheduledQueryArn": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

ScheduledQueryArn

スケジュールされたクエリの ARN。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：はい

レスポンス要素

アクションが成功すると、サービスは空のHTTP本文で 200 HTTP レスポンスを送り返します。

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、サービスはこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 400

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

リクエストされたリソースが見つかりませんでした。

HTTP ステータスコード: 400

ThrottlingException

リクエストのスロットリングにより、リクエストが拒否されました。

HTTP ステータスコード: 400

ValidationException

無効または不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用

- AWS SDK Python 用
- AWS SDK Ruby V3 用

DescribeAccountSettings
サービス：Amazon Timestream Query

クエリ料金モデルとTCUs、サービスがクエリワークロードに使用できる設定済みの最大数を含むアカウントの設定について説明します。

ワークロードに使用されるコンピューティングユニットの期間に対してのみ課金されます。

レスポンスの構文

```
{
   "MaxQueryTCU": number,
   "QueryPricingModel": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

MaxQueryTCU

サービスがクエリを処理するために任意の時点で使用する Timestream コンピューティングユニット (TCUs) の最大数。

型: 整数

QueryPricingModel

アカウント内のクエリの料金モデル。

型: 文字列

有効な値: BYTES_SCANNED | COMPUTE_UNITS

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、サービスはこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 400

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ThrottlingException

リクエストのスロットリングにより、リクエストが拒否されました。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

DescribeEndpoints
サービス：Amazon Timestream Query

DescribeEndpoints は、Timestream APIコールを行う利用可能なエンドポイントのリストを返します。これはAPI、書き込みとクエリの両方で使用できます。

Timestream SDKsは、サービスエンドポイントの管理やマッピングなど、サービスのアーキテクチャと透過的に連携するように設計されているため、 でAPIない限り、これを使用することはお勧めしません。

- VPC Timestream でエンドポイント (AWS PrivateLink) を使用している
- アプリケーションが、まだSDKサポートされていないプログラミング言語を使用している
- クライアント側の実装をより適切に制御する必要がある

を使用および実装する方法と時期の詳細については DescribeEndpoints、「エンドポイント検出パターン」を参照してください。

レスポンスの構文

```
{
   "Endpoints": [
      {
         "Address": "string",
         "CachePeriodInMinutes": number
      }
   ]
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

Endpoints

Endpoints オブジェクトは、DescribeEndpointsリクエストが行われたときに返されます。

型: Endpoint オブジェクトの配列

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InternalServerException

内部サーバーエラーのため、サービスはこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 400

ThrottlingException

リクエストのスロットリングにより、リクエストが拒否されました。

HTTP ステータスコード: 400

ValidationException

無効または不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

DescribeScheduledQuery
サービス：Amazon Timestream Query

スケジュールされたクエリに関する詳細情報を提供します。

リクエストの構文

```
{
   "ScheduledQueryArn": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

ScheduledQueryArn

スケジュールされたクエリの ARN。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：はい

レスポンスの構文

```
{
   "ScheduledQuery": {
      "Arn": "string",
      "CreationTime": number,
      "ErrorReportConfiguration": {
         "S3Configuration": {
            "BucketName": "string",
            "EncryptionOption": "string",
            "ObjectKeyPrefix": "string"
         }
      },
      "KmsKeyId": "string",
      "LastRunSummary": {
```

```
         "ErrorReportLocation": {
            "S3ReportLocation": {
               "BucketName": "string",
               "ObjectKey": "string"
            }
         },
         "ExecutionStats": {
            "BytesMetered": number,
            "CumulativeBytesScanned": number,
            "DataWrites": number,
            "ExecutionTimeInMillis": number,
            "QueryResultRows": number,
            "RecordsIngested": number
         },
         "FailureReason": "string",
         "InvocationTime": number,
         "QueryInsightsResponse": {
            "OutputBytes": number,
            "OutputRows": number,
            "QuerySpatialCoverage": {
               "Max": {
                  "PartitionKey": [ "string" ],
                  "TableArn": "string",
                  "Value": number
               }
            },
            "QueryTableCount": number,
            "QueryTemporalRange": {
               "Max": {
                  "TableArn": "string",
                  "Value": number
               }
            }
         },
         "RunStatus": "string",
         "TriggerTime": number
      },
      "Name": "string",
      "NextInvocationTime": number,
      "NotificationConfiguration": {
         "SnsConfiguration": {
            "TopicArn": "string"
         }
      },
```

```
      "PreviousInvocationTime": number,
      "QueryString": "string",
      "RecentlyFailedRuns": [
         {
            "ErrorReportLocation": {
               "S3ReportLocation": {
                  "BucketName": "string",
                  "ObjectKey": "string"
               }
            },
            "ExecutionStats": {
               "BytesMetered": number,
               "CumulativeBytesScanned": number,
               "DataWrites": number,
               "ExecutionTimeInMillis": number,
               "QueryResultRows": number,
               "RecordsIngested": number
            },
            "FailureReason": "string",
            "InvocationTime": number,
            "QueryInsightsResponse": {
               "OutputBytes": number,
               "OutputRows": number,
               "QuerySpatialCoverage": {
                  "Max": {
                     "PartitionKey": [ "string" ],
                     "TableArn": "string",
                     "Value": number
                  }
               },
               "QueryTableCount": number,
               "QueryTemporalRange": {
                  "Max": {
                     "TableArn": "string",
                     "Value": number
                  }
               }
            },
            "RunStatus": "string",
            "TriggerTime": number
         }
      ],
      "ScheduleConfiguration": {
         "ScheduleExpression": "string"
```

```
      },
      "ScheduledQueryExecutionRoleArn": "string",
      "State": "string",
      "TargetConfiguration": {
         "TimestreamConfiguration": {
            "DatabaseName": "string",
            "DimensionMappings": [
               {
                  "DimensionValueType": "string",
                  "Name": "string"
               }
            ],
            "MeasureNameColumn": "string",
            "MixedMeasureMappings": [
               {
                  "MeasureName": "string",
                  "MeasureValueType": "string",
                  "MultiMeasureAttributeMappings": [
                     {
                        "MeasureValueType": "string",
                        "SourceColumn": "string",
                        "TargetMultiMeasureAttributeName": "string"
                     }
                  ],
                  "SourceColumn": "string",
                  "TargetMeasureName": "string"
               }
            ],
            "MultiMeasureMappings": {
               "MultiMeasureAttributeMappings": [
                  {
                     "MeasureValueType": "string",
                     "SourceColumn": "string",
                     "TargetMultiMeasureAttributeName": "string"
                  }
               ],
               "TargetMultiMeasureName": "string"
            },
            "TableName": "string",
            "TimeColumn": "string"
         }
      }
   }
```

```
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

ScheduledQuery

スケジュールされたクエリ。

型: ScheduledQueryDescription オブジェクト

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、サービスはこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 400

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

リクエストされたリソースが見つかりませんでした。

HTTP ステータスコード: 400

ThrottlingException

リクエストのスロットリングにより、リクエストが拒否されました。

HTTP ステータスコード: 400

ValidationException

無効または不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

ExecuteScheduledQuery
サービス：Amazon Timestream Query

これを使用してAPI、スケジュールされたクエリを手動で実行できます。

を有効にするとQueryInsights、Amazon SNS通知の一部として実行したクエリに関連するインサイトとメトリクスAPIも返されます。 QueryInsights はクエリのパフォーマンス調整に役立ちます。の詳細についてはQueryInsights、「クエリインサイトを使用して Amazon Timestream のクエリを最適化する」を参照してください。

リクエストの構文

```
{
   "ClientToken": "string",
   "InvocationTime": number,
   "QueryInsights": {
      "Mode": "string"
   },
   "ScheduledQueryArn": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

ClientToken

使用されていません。

型: 文字列

長さの制限: 最小長 32。最大長は 128 です。

必須：いいえ

InvocationTime

のタイムスタンプUTC。クエリは、このタイムスタンプで呼び出されたかのように実行されます。

型: タイムスタンプ

必須：はい

QueryInsights

を有効にするための設定をカプセル化しますQueryInsights。

を有効にすると、実行したクエリの Amazon SNS通知の一部としてインサイトとメトリクスQueryInsightsが返されます。を使用してQueryInsights、クエリのパフォーマンスとコストを調整できます。

型: ScheduledQueryInsights オブジェクト

必須：いいえ

ScheduledQueryArn

ARN スケジュールされたクエリの 。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：はい

レスポンス要素

アクションが成功すると、サービスは空のHTTP本文で 200 HTTP レスポンスを送り返します。

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、サービスはこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 400

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

リクエストされたリソースが見つかりませんでした。

HTTP ステータスコード: 400

ThrottlingException

リクエストのスロットリングにより、リクエストが拒否されました。

HTTP ステータスコード: 400

ValidationException

無効または不正な形式のリクエスト。

HTTP ステータスコード: 400

例

ENABLED_WITH_RATE_CONTROL モードのスケジュールされたクエリ通知メッセージ

次の例は、 QueryInsightsパラメータの ENABLED_WITH_RATE_CONTROL モードのスケジュールされたクエリ通知メッセージの成功を示しています。

```
"SuccessNotificationMessage": {
    "type": "MANUAL_TRIGGER_SUCCESS",
    "arn": "arn:aws:timestream:<Region>:<Account>:scheduled-query/sq-test-49c6ed55-
c2e7-4cc2-9956-4a0ecea13420-80e05b035236a4c3",
    "scheduledQueryRunSummary": {
        "invocationEpochSecond": 1723710546,
        "triggerTimeMillis": 1723710547490,
        "runStatus": "MANUAL_TRIGGER_SUCCESS",
        "executionStats": {
            "executionTimeInMillis": 17343,
            "dataWrites": 1024,
```

```
            "bytesMetered": 0,
            "cumulativeBytesScanned": 600,
            "recordsIngested": 1,
            "queryResultRows": 1
        },
        "queryInsightsResponse": {
            "querySpatialCoverage": {
                "max": {
                    "value": 1.0,
                    "tableArn": "arn:aws:timestream:<Region>:<Account>:database/BaseDb/
table/BaseTable",
                    "partitionKey": [
                        "measure_name"
                    ]
                }
            },
            "queryTemporalRange": {
                "max": {
                    "value": 2399999999999,
                    "tableArn": "arn:aws:timestream:<Region>:<Account>:database/BaseDb/
table/BaseTable"
                }
            },
            "queryTableCount": 1,
            "outputRows": 1,
            "outputBytes": 59
        }
    }
}
```

DISABLED モードのスケジュールされたクエリ通知メッセージ

次の例は、 QueryInsightsパラメータの DISABLED モードのスケジュールされたクエリ通知メッセージの成功を示しています。

```
"SuccessNotificationMessage": {
    "type": "MANUAL_TRIGGER_SUCCESS",
    "arn": "arn:aws:timestream:<Region>:<Account>:scheduled-query/sq-test-
fa109d9e-6528-4a0d-ac40-482fa05e657f-140faaeecdc5b2a7",
    "scheduledQueryRunSummary": {
        "invocationEpochSecond": 1723711401,
        "triggerTimeMillis": 1723711402144,
        "runStatus": "MANUAL_TRIGGER_SUCCESS",
```

```
        "executionStats": {
            "executionTimeInMillis": 17992,
            "dataWrites": 1024,
            "bytesMetered": 0,
            "cumulativeBytesScanned": 600,
            "recordsIngested": 1,
            "queryResultRows": 1
        }
    }
}
```

ENABLED_WITH_RATE_CONTROL モードの障害通知メッセージ

次の例は、 QueryInsightsパラメータの ENABLED_WITH_RATE_CONTROL モードに対して失敗したスケジュールされたクエリ通知メッセージを示しています。

```
"FailureNotificationMessage": {
    "type": "MANUAL_TRIGGER_FAILURE",
    "arn": "arn:aws:timestream:<Region>:<Account>:scheduled-query/sq-test-
b261670d-790c-4116-9db5-0798071b18b1-b7e27a1d79be226d",
    "scheduledQueryRunSummary": {
        "invocationEpochSecond": 1727915513,
        "triggerTimeMillis": 1727915513894,
        "runStatus": "MANUAL_TRIGGER_FAILURE",
        "executionStats": {
            "executionTimeInMillis": 10777,
            "dataWrites": 0,
            "bytesMetered": 0,
            "cumulativeBytesScanned": 0,
            "recordsIngested": 0,
            "queryResultRows": 4
        },
        "errorReportLocation": {
            "s3ReportLocation": {
                "bucketName": "my-amzn-s3-demo-bucket",
                "objectKey": "4my-organization-f7a3c5d065a1a95e/1727915513/
MANUAL/1727915513894/5e14b3df-b147-49f4-9331-784f749b68ae"
            }
        },
        "failureReason": "Schedule encountered some errors and is incomplete. Please
 take a look at error report for further details"
    }
}
```

DISABLED モードの障害通知メッセージ

次の例は、 QueryInsightsパラメータの DISABLED モードに対して失敗したスケジュールされたクエリ通知メッセージを示しています。

```
"FailureNotificationMessage": {
    "type": "MANUAL_TRIGGER_FAILURE",
    "arn": "arn:aws:timestream:<Region>:<Account>:scheduled-query/sq-test-
b261670d-790c-4116-9db5-0798071b18b1-b7e27a1d79be226d",
    "scheduledQueryRunSummary": {
        "invocationEpochSecond": 1727915194,
        "triggerTimeMillis": 1727915195119,
        "runStatus": "MANUAL_TRIGGER_FAILURE",
        "executionStats": {
            "executionTimeInMillis": 10777,
            "dataWrites": 0,
            "bytesMetered": 0,
            "cumulativeBytesScanned": 0,
            "recordsIngested": 0,
            "queryResultRows": 4
        },
        "errorReportLocation": {
            "s3ReportLocation": {
                "bucketName": "my-amzn-s3-demo-bucket",
                "objectKey": "4my-organization-b7e27a1d79be226d/1727915194/
MANUAL/1727915195119/08dea9f5-9a0a-4e63-a5f7-ded23247bb98"
            }
        },
        "failureReason": "Schedule encountered some errors and is incomplete. Please
 take a look at error report for further details"
    }
}
```

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 用

- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

ListScheduledQueries
サービス：Amazon Timestream Query

発信者の Amazon アカウントとリージョンでスケジュールされたすべてのクエリのリストを取得します。 ListScheduledQueriesは最終的に一貫性があります。

リクエストの構文

```
{
   "MaxResults": number,
   "NextToken": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

MaxResults

出力で返されるアイテムの最大数。使用可能な項目の総数が指定された値を超える場合、出力には NextTokenが指定されます。ページ分割を再開するには、 への後続の呼び出しの引数として NextToken値を指定しますListScheduledQueriesRequest。

型: 整数

有効範囲: 最小値 は 1 です。最大値は 1000 です。

必須：いいえ

NextToken

ページ分割を再開するページ分割トークン。

型: 文字列

必須: いいえ

レスポンスの構文

```
{
```

```
   "NextToken": "string",
   "ScheduledQueries": [
      {
         "Arn": "string",
         "CreationTime": number,
         "ErrorReportConfiguration": {
            "S3Configuration": {
               "BucketName": "string",
               "EncryptionOption": "string",
               "ObjectKeyPrefix": "string"
            }
         },
         "LastRunStatus": "string",
         "Name": "string",
         "NextInvocationTime": number,
         "PreviousInvocationTime": number,
         "State": "string",
         "TargetDestination": {
            "TimestreamDestination": {
               "DatabaseName": "string",
               "TableName": "string"
            }
         }
      }
   ]
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

NextToken

ページ分割を始める場所を指定するトークン。これは、以前に切り捨てられたレスポンス NextToken からの です。

型: 文字列

ScheduledQueries

スケジュールされたクエリのリスト。

型: ScheduledQuery オブジェクトの配列

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、サービスはこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 400

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ThrottlingException

リクエストのスロットリングにより、リクエストが拒否されました。

HTTP ステータスコード: 400

ValidationException

無効または不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用

- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

ListTagsForResource
サービス：Amazon Timestream Query

Timestream クエリリソースのすべてのタグを一覧表示します。

リクエストの構文

```
{
   "MaxResults": number,
   "NextToken": "string",
   "ResourceARN": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

MaxResults

返されるタグの最大数。

型: 整数

有効範囲: 最小値 は 1 です。最大値は 200 です。

必須：いいえ

NextToken

ページ分割を再開するページ分割トークン。

型: 文字列

必須: いいえ

ResourceARN

リストされるタグを持つ Timestream リソース。この値は Amazon リソースネーム () ですARN。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：はい

レスポンスの構文

```
{
   "NextToken": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

NextToken

への後続の呼び出しでページ分割を再開するページ分割トークンListTagsForResourceResponse。

型: 文字列

Tags

Timestream リソースに現在関連付けられているタグ。

型: Tag オブジェクトの配列

配列メンバー: 最小数は 0 項目です。最大数は 200 項目です。

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

リクエストされたリソースが見つかりませんでした。

HTTP ステータスコード: 400

ThrottlingException

リクエストのスロットリングにより、リクエストが拒否されました。

HTTP ステータスコード: 400

ValidationException

無効または不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

PrepareQuery
サービス：Amazon Timestream Query

Timestream が後で実行するために保存するパラメータを含むクエリを送信できる同期オペレーション。Timestream は、 を `ValidateOnly`に設定してこのオペレーションの使用のみをサポートします`true`。

リクエストの構文

```
{
   "QueryString": "string",
   "ValidateOnly": boolean
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

QueryString

準備済みステートメントとして使用する Timestream クエリ文字列。パラメータ名は、クエリ文字列 @ 文字とその後に識別子を続けることで指定できます。

型: 文字列

長さの制限：最小長は 1 です。最大長は 262,144 です。

必須：はい

ValidateOnly

この値を に設定することで`true`、Timestream はクエリ文字列が有効な Timestream クエリであることのみを検証し、後で使用するために準備されたクエリを保存しません。

型: ブール値

必須：いいえ

レスポンスの構文

```
{
   "Columns": [
      {
         "Aliased": boolean,
         "DatabaseName": "string",
         "Name": "string",
         "TableName": "string",
         "Type": {
            "ArrayColumnInfo": {
               "Name": "string",
               "Type": "Type"
            },
            "RowColumnInfo": [
               {
                  "Name": "string",
                  "Type": "Type"
               }
            ],
            "ScalarType": "string",
            "TimeSeriesMeasureValueColumnInfo": {
               "Name": "string",
               "Type": "Type"
            }
         }
      }
   ],
   "Parameters": [
      {
         "Name": "string",
         "Type": {
            "ArrayColumnInfo": {
               "Name": "string",
               "Type": "Type"
            },
            "RowColumnInfo": [
               {
                  "Name": "string",
                  "Type": "Type"
               }
            ],
            "ScalarType": "string",
            "TimeSeriesMeasureValueColumnInfo": {
```

```
                "Name": "string",
                "Type": "Type"
            }
        }
    }
  ],
  "QueryString": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

Columns

送信されたクエリ文字列のSELECT句列のリスト。

型: SelectColumn オブジェクトの配列

Parameters

送信されたクエリ文字列で使用されるパラメータのリスト。

型: ParameterMapping オブジェクトの配列

QueryString

準備するクエリ文字列。

型: 文字列

長さの制限：最小長は 1 です。最大長は 262,144 です。

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、サービスはこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 400

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ThrottlingException

リクエストのスロットリングにより、リクエストが拒否されました。

HTTP ステータスコード: 400

ValidationException

無効または不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ の場合
- AWS SDK Go v2 用
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

Query
サービス：Amazon Timestream Query

Query は、Amazon Timestream データに対してクエリを実行できるようにする同期オペレーションです。

を有効にするとQueryInsights、実行したクエリに関連するインサイトとメトリクスAPIも返されます。 QueryInsights はクエリのパフォーマンス調整に役立ちます。の詳細についてはQueryInsights、「クエリインサイトを使用して Amazon Timestream のクエリを最適化する」を参照してください。

> **Note**
>
> QueryInsights 有効にして実行できるQueryAPIリクエストの最大数は、1 秒あたり 1 クエリです (QPS) 。このクエリレートを超えると、スロットリングが発生する可能性があります。

Query は 60 秒後にタイムアウトします。60 秒のタイムアウトをサポートするSDKには、 のデフォルトのタイムアウトを更新する必要があります。詳細については、コードサンプルを参照してください。

次の場合、クエリリクエストは失敗します。

- 5 分間の冪等性ウィンドウ外で同じクライアントトークンを使用してQueryリクエストを送信する場合。
- 同じクライアントトークンでQueryリクエストを送信し、5 分間の等価性ウィンドウ内で他のパラメータを変更した場合。
- 行のサイズ (クエリメタデータを含む) が 1 MB を超えると、クエリは失敗し、次のエラーメッセージが表示されます。

  ```
  Query aborted as max page response size has been exceeded by the output result row
  ```

- クエリイニシエータのIAMプリンシパルと結果リーダーが同じでない場合、および/またはクエリイニシエータと結果リーダーがクエリリクエストに同じクエリ文字列を持たない場合、クエリはInvalid pagination tokenエラーで失敗します。

リクエストの構文

```
{
   "ClientToken": "string",
   "MaxRows": number,
   "NextToken": "string",
   "QueryInsights": {
      "Mode": "string"
   },
   "QueryString": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次の JSON 形式のデータを受け入れます。

ClientToken

Query リクエストの実行時に指定された最大 64 ASCII文字の一意の大文字と小文字を区別する文字列。を指定するClientTokenと、 Query は冪等な を呼び出します。つまり、同じクエリを繰り返し実行すると、同じ結果が生成されます。つまり、複数の同じQueryリクエストを行うと、1 つのリクエストを行うのと同じ効果があります。クエリClientTokenで を使用する場合は、次の点に注意してください。

- クエリAPIが なしでインスタンス化された場合ClientToken、クエリはClientTokenユーザーに代わって SDKを生成します。
- Query 呼び出しに のみが含まれClientToken、 が含まれていない場合NextToken、 の呼び出しQueryは新しいクエリ実行と見なされます。
- 呼び出しに が含まれている場合NextToken、その特定の呼び出しはクエリ への以前の呼び出しの後続の呼び出しであると想定されAPI、結果セットが返されます。
- 4 時間後、同じ を持つすべてのリクエストClientTokenは新しいリクエストとして扱われます。

型: 文字列

長さの制限: 最小長 32。最大長は 128 です。

必須：いいえ

MaxRows

Query 出力で返される行の合計数。MaxRows 値を指定Queryして を最初に実行すると、クエリの結果セットが 2 つのケースで返されます。

- 結果のサイズが 未満です1MB。
- 結果セットの行数が の値より少なくなりますmaxRows。

それ以外の場合、 の最初の呼び出しは Queryのみを返します。これはNextToken、結果セットを取得するために後続の呼び出しで使用できます。ページ分割を再開するには、後続のコマンドに NextToken 値を指定します。

行サイズが大きい場合 (行に列が多い場合など)、Timestream はレスポンスサイズが 1 MB の制限を超えないように、より少ない行を返すことがあります。が指定されMaxRowsていない場合、Timestream は 1 MB の制限を満たすために必要な数の行を送信します。

型: 整数

有効範囲: 最小値 は 1 です。最大値は 1000 です。

必須 : いいえ

NextToken

一連の結果を返すために使用されるページ分割トークン。を使用して QueryAPIを呼び出すとNextToken、その特定の呼び出しは への以前の呼び出しの後続の呼び出しと見なされQuery、結果セットが返されます。ただし、Query呼び出しに のみが含まれている場合ClientToken、 の呼び出しQueryは新しいクエリ実行と見なされます。

クエリ NextToken で を使用する場合は、次の点に注意してください。

- ページ分割トークンは、最大 5 Query 回の呼び出し、または最大 1 時間のいずれか早い方まで使用できます。
- 同じ を使用するとNextToken、同じレコードセットが返されます。結果セットをページ分割し続けるには、最新の を使用する必要がありますnextToken。
- Query 呼び出しが 2 つのNextToken値 TokenAと を返すとしますTokenB。TokenB が後続Queryの呼び出しで使用されている場合、 TokenAは無効になり、再利用することはできません。
- ページ分割の開始後にクエリから以前の結果セットをリクエストするには、クエリ を再呼び出す必要がありますAPI。

- 最新の を使用して、 nullが返されるまでページ分割NextTokenし、その時点で新しい NextToken を使用する必要があります。
- クエリイニシエータのIAMプリンシパルと結果リーダーが同じでない場合、および/またはクエリイニシエータと結果リーダーがクエリリクエストに同じクエリ文字列を持たない場合、クエリはInvalid pagination token工ラーで失敗します。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：いいえ

QueryInsights

を有効にする設定をカプセル化しますQueryInsights。

を有効にすると、実行したクエリのクエリ結果に加えて、インサイトとメトリクスQueryInsightsが返されます。QueryInsights を使用してクエリパフォーマンスを調整できます。

型: QueryInsights オブジェクト

必須：いいえ

QueryString

Timestream によって実行されるクエリ。

型: 文字列

長さの制限：最小長は 1 です。最大長は 262,144 です。

必須：はい

レスポンスの構文

```
{
   "ColumnInfo": [
      {
         "Name": "string",
         "Type": {
            "ArrayColumnInfo": "ColumnInfo",
            "RowColumnInfo": [
               "ColumnInfo"
```

```
            ],
            "ScalarType": "string",
            "TimeSeriesMeasureValueColumnInfo": "ColumnInfo"
         }
      }
   ],
   "NextToken": "string",
   "QueryId": "string",
   "QueryInsightsResponse": {
      "OutputBytes": number,
      "OutputRows": number,
      "QuerySpatialCoverage": {
         "Max": {
            "PartitionKey": [ "string" ],
            "TableArn": "string",
            "Value": number
         }
      },
      "QueryTableCount": number,
      "QueryTemporalRange": {
         "Max": {
            "TableArn": "string",
            "Value": number
         }
      },
      "UnloadPartitionCount": number,
      "UnloadWrittenBytes": number,
      "UnloadWrittenRows": number
   },
   "QueryStatus": {
      "CumulativeBytesMetered": number,
      "CumulativeBytesScanned": number,
      "ProgressPercentage": number
   },
   "Rows": [
      {
         "Data": [
            {
               "ArrayValue": [
                  "Datum"
               ],
               "NullValue": boolean,
               "RowValue": "Row",
               "ScalarValue": "string",
```

```
                "TimeSeriesValue": [
                   {
                      "Time": "string",
                      "Value": "Datum"
                   }
                ]
             }
          ]
       }
    ]
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

ColumnInfo

返された結果セットの列データ型。

型: ColumnInfo オブジェクトの配列

NextToken

Query 呼び出しで再度使用して次の結果セットを取得できるページ分割トークン。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

QueryId

指定されたクエリの一意の ID。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

パターン: [a-zA-Z0-9]+

QueryInsightsResponse

実行したクエリに関連するインサイトとメトリクスQueryInsightsを含むカプセル化。

型: QueryInsightsResponse オブジェクト

QueryStatus

スキャンされた進行状況やバイト数など、クエリのステータスに関する情報。

型: QueryStatus オブジェクト

Rows

クエリによって返される行の結果セット。

型: Row オブジェクトの配列

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

ConflictException

キャンセルされたクエリの結果をポーリングできません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、サービスはこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 400

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

QueryExecutionException

Timestream はクエリを正常に実行できませんでした。

HTTP ステータスコード: 400

ThrottlingException

リクエストのスロットリングにより、リクエストが拒否されました。

HTTP ステータスコード: 400

ValidationException

無効または不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

TagResource
サービス：Amazon Timestream Query

一連のタグを Timestream リソースに関連付けます。その後、これらのユーザー定義タグをアクティブ化して、コスト配分の追跡のために請求とコスト管理コンソールに表示されるようにすることができます。

リクエストの構文

```
{
   "ResourceARN": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

ResourceARN

タグを追加する Timestream リソースを識別します。この値は Amazon リソースネーム () です ARN。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：はい

Tags

Timestream リソースに割り当てるタグ。

型: Tag オブジェクトの配列

配列メンバー: 最小数は 0 項目です。最大数は 200 項目です。

必須：はい

レスポンス要素

アクションが成功すると、サービスは空のHTTP本文で 200 HTTP レスポンスを送り返します。

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

リクエストされたリソースが見つかりませんでした。

HTTP ステータスコード: 400

ServiceQuotaExceededException

サービスクォータを超えました。

HTTP ステータスコード: 400

ThrottlingException

リクエストのスロットリングにより、リクエストが拒否されました。

HTTP ステータスコード: 400

ValidationException

無効または不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

UntagResource
サービス：Amazon Timestream Query

Timestream クエリリソースからタグの関連付けを削除します。

リクエストの構文

```
{
   "ResourceARN": "string",
   "TagKeys": [ "string" ]
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次のJSON形式のデータを受け入れます。

ResourceARN

タグを削除する Timestream リソース。この値は Amazon リソースネーム () ですARN。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：はい

TagKeys

タグキーのリスト。このリストのメンバーであるキーを持つリソースの既存のタグは、Timestream リソースから削除されます。

型: 文字列の配列

配列メンバー：最小数は 0 項目です。最大数は 200 項目です。

長さの制限：最小長は 1 です。最大長は 128 です。

必須：はい

レスポンス要素

アクションが成功すると、サービスは空のHTTP本文で 200 HTTP レスポンスを送り返します。

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

リクエストされたリソースが見つかりませんでした。

HTTP ステータスコード: 400

ThrottlingException

リクエストのスロットリングにより、リクエストが拒否されました。

HTTP ステータスコード: 400

ValidationException

無効または不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3

- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

UpdateAccountSettings
サービス：Amazon Timestream Query

クエリの料金TCUs設定に使用するようにアカウントを移行し、設定した最大クエリコンピューティングユニットを変更します。の値を目的の設定MaxQueryTCUに減らすと、新しい値が有効になるまでに最大 24 時間かかることがあります。

> **Note**
>
> クエリ料金TCUsに使用するようにアカウントを移行した後は、クエリ料金でスキャンされたバイトの使用に移行することはできません。

リクエストの構文

```
{
   "MaxQueryTCU": number,
   "QueryPricingModel": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次の JSON 形式のデータを受け入れます。

MaxQueryTCU

サービスがクエリを処理するために任意の時点で使用するコンピューティングユニットの最大数。クエリを実行するには、最小容量を 4 に設定する必要がありますTCU。の最大数は、4、8、16、32 などの 4 倍TCU数で設定できます。

でサポートされる最大値MaxQueryTCUは 1000 です。このソフト制限の引き上げをリクエストするには、 AWS サポートにお問い合わせください。のデフォルトクォータの詳細についてはmaxQueryTCU、「デフォルトクォータ」を参照してください。

タイプ: 整数

必須：いいえ

QueryPricingModel

アカウント内のクエリの料金モデル。

> **Note**
>
> QueryPricingModel パラメータは複数の Timestream オペレーションで使用されますが、UpdateAccountSettingsAPIオペレーションは 以外の値を認識しませんCOMPUTE_UNITS。

型: 文字列

有効な値: BYTES_SCANNED | COMPUTE_UNITS

必須：いいえ

レスポンスの構文

```
{
   "MaxQueryTCU": number,
   "QueryPricingModel": "string"
}
```

レスポンス要素

アクションが成功すると、サービスは 200 HTTP レスポンスを送り返します。

次のデータは、 サービスによって JSON 形式で返されます。

MaxQueryTCU

サービスがクエリを処理するために任意の時点で使用する、設定されたコンピューティングユニットの最大数。

型: 整数

QueryPricingModel

アカウントの料金モデル。

型: 文字列

有効な値: BYTES_SCANNED | COMPUTE_UNITS

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、サービスはこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 400

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ThrottlingException

リクエストのスロットリングにより、リクエストが拒否されました。

HTTP ステータスコード: 400

ValidationException

無効または不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス
- AWS SDK の 。NET
- AWS SDK C++ 用

- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

UpdateScheduledQuery
サービス：Amazon Timestream Query

スケジュールされたクエリを更新します。

リクエストの構文

```
{
   "ScheduledQueryArn": "string",
   "State": "string"
}
```

リクエストパラメータ

すべてのアクションに共通のパラメータの詳細については、「共通パラメータ」を参照してください。

リクエストは、次の JSON 形式のデータを受け入れます。

ScheduledQueryArn

ARN のスケジューラされたクエリ。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：はい

State

スケジュールされたクエリの状態。

型: 文字列

有効な値: ENABLED | DISABLED

必須：はい

レスポンス要素

アクションが成功すると、サービスは空のHTTP本文で 200 HTTP レスポンスを送り返します。

エラー

すべてのアクションに共通のエラーについては、「共通エラー」を参照してください。

AccessDeniedException

このアクションを実行する権限がありません。

HTTP ステータスコード: 400

InternalServerException

内部サーバーエラーのため、サービスはこのリクエストを完全に処理できませんでした。

HTTP ステータスコード: 400

InvalidEndpointException

リクエストされたエンドポイントが無効です。

HTTP ステータスコード: 400

ResourceNotFoundException

リクエストされたリソースが見つかりませんでした。

HTTP ステータスコード: 400

ThrottlingException

リクエストのスロットリングにより、リクエストが拒否されました。

HTTP ステータスコード: 400

ValidationException

無効または不正な形式のリクエスト。

HTTP ステータスコード: 400

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS コマンドラインインターフェイス

- AWS SDK の 。NET
- AWS SDK C++ 用
- AWS SDK Go v2 の場合
- AWS SDK Java V2 用
- AWS SDK for JavaScript V3
- AWS SDK PHP V3 用
- AWS SDK Python 用
- AWS SDK Ruby V3 用

データ型

Amazon Timestream Write では、次のデータ型がサポートされています。

- BatchLoadProgressReport
- BatchLoadTask
- BatchLoadTaskDescription
- CsvConfiguration
- Database
- DataModel
- DataModelConfiguration
- DataModelS3Configuration
- DataSourceConfiguration
- DataSourceS3Configuration
- Dimension
- DimensionMapping
- Endpoint
- MagneticStoreRejectedDataLocation
- MagneticStoreWriteProperties
- MeasureValue
- MixedMeasureMapping
- MultiMeasureAttributeMapping
- MultiMeasureMappings

- PartitionKey
- Record
- RecordsIngested
- RejectedRecord
- ReportConfiguration
- ReportS3Configuration
- RetentionProperties
- S3Configuration
- Schema
- Table
- Tag

Amazon Timestream クエリでは、次のデータ型がサポートされています。

- ColumnInfo
- Datum
- DimensionMapping
- Endpoint
- ErrorReportConfiguration
- ErrorReportLocation
- ExecutionStats
- MixedMeasureMapping
- MultiMeasureAttributeMapping
- MultiMeasureMappings
- NotificationConfiguration
- ParameterMapping
- QueryInsights
- QueryInsightsResponse
- QuerySpatialCoverage
- QuerySpatialCoverageMax
- QueryStatus

- QueryTemporalRange
- QueryTemporalRangeMax
- Row
- S3Configuration
- S3ReportLocation
- ScheduleConfiguration
- ScheduledQuery
- ScheduledQueryDescription
- ScheduledQueryInsights
- ScheduledQueryInsightsResponse
- ScheduledQueryRunSummary
- SelectColumn
- SnsConfiguration
- Tag
- TargetConfiguration
- TargetDestination
- TimeSeriesDataPoint
- TimestreamConfiguration
- TimestreamDestination
- Type

Amazon Timestream 書き込み

Amazon Timestream Write では、次のデータ型がサポートされています。

- BatchLoadProgressReport
- BatchLoadTask
- BatchLoadTaskDescription
- CsvConfiguration
- Database
- DataModel
- DataModelConfiguration

- DataModelS3Configuration
- DataSourceConfiguration
- DataSourceS3Configuration
- Dimension
- DimensionMapping
- Endpoint
- MagneticStoreRejectedDataLocation
- MagneticStoreWriteProperties
- MeasureValue
- MixedMeasureMapping
- MultiMeasureAttributeMapping
- MultiMeasureMappings
- PartitionKey
- Record
- RecordsIngested
- RejectedRecord
- ReportConfiguration
- ReportS3Configuration
- RetentionProperties
- S3Configuration
- Schema
- Table
- Tag

BatchLoadProgressReport
サービス：Amazon Timestream Write

バッチロードタスクの進行状況に関する詳細。

内容

BytesMetered

型: Long

必須：いいえ

FileFailures

型: Long

必須：いいえ

ParseFailures

型: Long

必須：いいえ

RecordIngestionFailures

型: Long

必須：いいえ

RecordsIngested

型: Long

必須：いいえ

RecordsProcessed

型: Long

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

BatchLoadTask
サービス：Amazon Timestream Write

バッチロードタスクに関する詳細。

内容

CreationTime

Timestream バッチロードタスクが作成された時刻。

型: タイムスタンプ

必須：いいえ

DatabaseName

バッチロードタスクがデータをロードするデータベースのデータベース名。

型: 文字列

必須: いいえ

LastUpdatedTime

Timestream バッチロードタスクが最後に更新された時刻。

型: タイムスタンプ

必須：いいえ

ResumableUntil

型: タイムスタンプ

必須：いいえ

TableName

バッチロードタスクがデータをロードするテーブルのテーブル名。

型: 文字列

必須: いいえ

TaskId

バッチロードタスクの ID。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 32 です。

パターン: `[A-Z0-9]+`

必須: いいえ

TaskStatus

バッチロードタスクのステータス。

型: 文字列

有効な値: `CREATED | IN_PROGRESS | FAILED | SUCCEEDED | PROGRESS_STOPPED | PENDING_RESUME`

必須：いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

BatchLoadTaskDescription
サービス：Amazon Timestream Write

バッチロードタスクに関する詳細。

内容

CreationTime

Timestream バッチロードタスクが作成された時刻。

型: タイムスタンプ

必須：いいえ

DataModelConfiguration

バッチロードタスクのデータモデル設定。これには、バッチロードタスクのデータモデルが保存される場所に関する詳細が含まれます。

型: DataModelConfiguration オブジェクト

必須：いいえ

DataSourceConfiguration

バッチロードタスクのデータソースに関する設定の詳細。

型: DataSourceConfiguration オブジェクト

必須: いいえ

ErrorMessage

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：いいえ

LastUpdatedTime

Timestream バッチロードタスクが最後に更新された時刻。

型: タイムスタンプ

必須：いいえ

ProgressReport

型: BatchLoadProgressReport オブジェクト

必須：いいえ

RecordVersion

型: Long

必須：いいえ

ReportConfiguration

バッチロードタスクの設定をレポートします。これには、エラーレポートの保存先に関する詳細が含まれます。

型: ReportConfiguration オブジェクト

必須：いいえ

ResumableUntil

型: タイムスタンプ

必須: いいえ

TargetDatabaseName

型: 文字列

必須: いいえ

TargetTableName

型: 文字列

必須: いいえ

TaskId

バッチロードタスクの ID。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 32 です。

パターン: `[A-Z0-9]+`

必須: いいえ

TaskStatus

バッチロードタスクのステータス。

型: 文字列

有効な値: `CREATED | IN_PROGRESS | FAILED | SUCCEEDED | PROGRESS_STOPPED | PENDING_RESUME`

必須：いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

CsvConfiguration
サービス：Amazon Timestream Write

列区切り文字をカンマにし、レコード区切り文字を改行文字とする区切りデータ形式。

内容

ColumnSeparator

列区切り文字は、カンマ (',')、パイプ ('|)、セミコロン (';')、タブ ('/t')、または空白 (' ') のいずれかです。

型: 文字列

長さの制限：1 の固定長。

必須：いいえ

EscapeChar

エスケープ文字は、次のいずれかになります。

型: 文字列

長さの制限：1 の固定長。

必須：いいえ

NullValue

空白 (' ') にすることができます。

型: 文字列

長さの制限：最小長は 1 です。最大長は 256 です。

必須：いいえ

QuoteChar

一重引用符 (') または二重引用符 (「) を使用できます。

型: 文字列

長さの制限：1 の固定長。

必須：いいえ

TrimWhiteSpace

先頭と末尾の空白をトリミングするように指定します。

型: ブール値

必須：いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

Database
サービス：Amazon Timestream Write

テーブルの最上位コンテナ。データベースとテーブルは、Amazon Timestream の基本的な管理概念です。データベース内のすべてのテーブルは、同じ AWS KMS キーで暗号化されます。

内容

Arn

このデータベースを一意に識別する Amazon リソースネーム。

型: 文字列

必須: いいえ

CreationTime

データベースが作成された時刻。Unix エポック時間から計算されます。

型: タイムスタンプ

必須：いいえ

DatabaseName

Timestream データベースの名前。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 256 です。

必須：いいえ

KmsKeyId

データベースに保存されているデータの暗号化に使用される AWS KMS キーの識別子。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：いいえ

LastUpdatedTime

このデータベースが最後に更新された時刻。

型: タイムスタンプ

必須：いいえ

TableCount

Timestream データベース内で見つかったテーブルの合計数。

型: Long

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

DataModel
サービス：Amazon Timestream Write

バッチロードタスクのデータモデル。

内容

DimensionMappings

ディメンションのソースからターゲットへのマッピング。

型: DimensionMapping オブジェクトの配列

配列メンバー：最小数は 1 項目です。

必須: はい

MeasureNameColumn

型: 文字列

長さの制限：最小長は 1 です。最大長は 256 です。

必須：いいえ

MixedMeasureMappings

メジャーのソースからターゲットへのマッピング。

型: MixedMeasureMapping オブジェクトの配列

配列メンバー：最小数は 1 項目です。

必須：いいえ

MultiMeasureMappings

マルチメジャーレコードのソースからターゲットへのマッピング。

型: MultiMeasureMappings オブジェクト

必須：いいえ

TimeColumn

時刻にマッピングされるソース列。

型: 文字列

長さの制限：最小長は 1 です。最大長は 256 です。

必須：いいえ

TimeUnit

タイムスタンプユニットの粒度。時間値が秒、ミリ秒、ナノ秒、またはその他のサポートされている値であるかどうかを示します。デフォルトは MILLISECONDS です。

型: 文字列

有効な値: MILLISECONDS | SECONDS | MICROSECONDS | NANOSECONDS

必須：いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

DataModelConfiguration
サービス：Amazon Timestream Write

内容

DataModel

型: DataModel オブジェクト

必須：いいえ

DataModelS3Configuration

型: DataModelS3Configuration オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

DataModelS3Configuration
サービス：Amazon Timestream Write

内容

BucketName

型: 文字列

長さの制限: 最小長は 3 です。最大長は 63 です。

パターン: `[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]`

必須: いいえ

ObjectKey

型: 文字列

長さの制限: 最小長は 1 です。最大長は 1,024 です。

パターン: `[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

DataSourceConfiguration
サービス：Amazon Timestream Write

データソースの設定の詳細を定義します。

内容

DataFormat

これは現在 ですCSV。

型: 文字列

有効な値: CSV

必須：はい

DataSourceS3Configuration

ロードするデータを含むファイルの S3 の場所の設定。

型: DataSourceS3Configuration オブジェクト

必須：はい

CsvConfiguration

列区切り文字をカンマにし、レコード区切り文字を改行文字とする区切りデータ形式。

型: CsvConfiguration オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

DataSourceS3Configuration
サービス：Amazon Timestream Write

内容

BucketName

S3 バケットカスタマーのバケット名。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 63 です。

Pattern: `[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]`

必須：はい

ObjectKeyPrefix

型: 文字列

長さの制限: 最小長は 1 です。最大長は 1,024 です。

パターン: `[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

Dimension
サービス：Amazon Timestream Write

時系列のメタデータ属性を表します。例えば、EC2インスタンスの名前とアベイラビリティーゾーン、または風力タービンの製造元の名前はディメンションです。

内容

Name

ディメンションは、時系列のメタデータ属性を表します。例えば、EC2インスタンスの名前とアベイラビリティーゾーン、または風力タービンの製造元の名前はディメンションです。

ディメンション名の制約については、「制約の命名」を参照してください。

型: 文字列

長さの制限：最小長は 1 です。最大長は 60 です。

必須：はい

Value

ディメンションの値。

型: 文字列

必須: はい

DimensionValueType

時系列データポイントのディメンションのデータ型。

型: 文字列

有効な値: VARCHAR

必須：いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

DimensionMapping
サービス：Amazon Timestream Write

内容

DestinationColumn

型: 文字列

長さの制限: 最小長は 1 です。

必須: いいえ

SourceColumn

型: 文字列

長さの制限: 最小長は 1 です。

必須：いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

Endpoint
サービス：Amazon Timestream Write

API 呼び出し先となる利用可能なエンドポイントとそのエンドポイントTTLの を表します。

内容

Address

エンドポイントアドレス。

型: 文字列

必須: はい

CachePeriodInMinutes

エンドポイントTTLの を分単位で指定します。

タイプ: Long

必須：はい

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

MagneticStoreRejectedDataLocation
サービス：Amazon Timestream Write

磁気ストアの書き込み中に拒否されたレコードのエラーレポートを非同期に書き込む場所。

内容

S3Configuration

磁気ストアの書き込み中に拒否されたレコードのエラーレポートを非同期に書き込む S3 ロケーションの設定。

型: S3Configuration オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

MagneticStoreWriteProperties
サービス：Amazon Timestream Write

マグネティックストアの書き込みを設定するためのテーブル上のプロパティセット。

内容

EnableMagneticStoreWrites

マグネティックストアの書き込みを有効にするフラグ。

型: ブール値

必須：はい

MagneticStoreRejectedDataLocation

マグネティックストアの書き込み中に拒否されたレコードのエラーレポートを非同期に書き込む場所。

型: MagneticStoreRejectedDataLocation オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

MeasureValue
サービス：Amazon Timestream Write

時系列のデータ属性を表します。例えば、EC2インスタンスまたは風力タービンRPMの のCPU使用率は measures. に名前と値の両方 MeasureValue があります。

MeasureValue はタイプ でのみ許可されますMULTI。MULTI タイプを使用すると、同じ時系列に関連付けられた複数のデータ属性を 1 つのレコードに渡すことができます。

内容

Name

の名前 MeasureValue。

MeasureValue 名前の制約については、「Amazon Timestream デベロッパーガイド」の「命名制約」を参照してください。

型: 文字列

長さの制限：最小長は 1 です。

必須：はい

Type

MeasureValue 時系列データポイントの のデータ型が含まれます。

型: 文字列

有効な値: `DOUBLE | BIGINT | VARCHAR | BOOLEAN | TIMESTAMP | MULTI`

必須：はい

Value

の値 MeasureValue。詳細については、「 データ型」を参照してください。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：はい

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

MixedMeasureMapping
サービス：Amazon Timestream Write

内容

MeasureValueType

型: 文字列

有効な値: DOUBLE | BIGINT | VARCHAR | BOOLEAN | TIMESTAMP | MULTI

必須: はい

MeasureName

型: 文字列

長さの制限: 最小長は 1 です。

必須：いいえ

MultiMeasureAttributeMappings

型: MultiMeasureAttributeMapping オブジェクトの配列

配列メンバー：最小数は 1 項目です。

必須: いいえ

SourceColumn

型: 文字列

長さの制限: 最小長は 1 です。

必須: いいえ

TargetMeasureName

型: 文字列

長さの制限: 最小長は 1 です。

必須：いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

MultiMeasureAttributeMapping
サービス：Amazon Timestream Write

内容

SourceColumn

型: 文字列

長さの制限：最小長は 1 です。

必須: はい

MeasureValueType

型: 文字列

有効な値: `DOUBLE | BIGINT | BOOLEAN | VARCHAR | TIMESTAMP`

必須: いいえ

TargetMultiMeasureAttributeName

型: 文字列

長さの制限: 最小長は 1 です。

必須：いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

MultiMeasureMappings
サービス：Amazon Timestream Write

内容

MultiMeasureAttributeMappings

型: MultiMeasureAttributeMapping オブジェクトの配列

配列メンバー：最小数は 1 項目です。

必須: はい

TargetMultiMeasureName

型: 文字列

長さの制限: 最小長は 1 です。

必須：いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

PartitionKey
サービス：Amazon Timestream Write

テーブル内のデータのパーティショニングに使用される属性。ディメンションキーは、ディメンション名で指定されたディメンションの値をパーティションキーとして使用してデータをパーティション化し、メジャーキーはメジャー名 ('measure_name' 列の値) を使用してデータをパーティション化します。

内容

Type

パーティションキーのタイプ。オプションは DIMENSION (ディメンションキー) と MEASURE (メジャーキー) です。

型: 文字列

有効な値: `DIMENSION | MEASURE`

必須：はい

EnforcementInRecord

取り込まれたレコードのディメンションキーの指定に対する適用レベル。オプションは REQUIRED (ディメンションキーを指定する必要があります) と OPTIONAL (ディメンションキーを指定する必要はありません) です。

型: 文字列

有効な値: `REQUIRED | OPTIONAL`

必須：いいえ

Name

ディメンションキーに使用される属性の名前。

型: 文字列

長さの制限: 最小長は 1 です。

必須：いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

Record
サービス：Amazon Timestream Write

Timestream に書き込まれる時系列データポイントを表します。各レコードにはディメンションの配列が含まれています。属性は、インスタンス名やEC2インスタンスのアベイラビリティーゾーンなど、時系列データポイントのメタデータ属性を表します。レコードには、収集されるメジャーの名前であるメジャー名 (EC2インスタンスのCPU使用率など) も含まれます。さらに、レコードには測定値と、測定値のデータ型である値型が含まれます。また、レコードには、メジャーが収集された時刻のタイムスタンプと、タイムスタンプの粒度を表すタイムスタンプ単位が含まれます。

レコードには、データポイントの更新に使用できる 64 ビットの `Version` フィールドlongがあります。同じディメンション、タイムスタンプ、メジャー名を持つ重複レコードの書き込みは、書き込みリクエスト内のレコードの`Version`属性が既存のレコードの属性よりも高い場合にのみ成功します。Timestream は、 `Version`フィールドのないレコード1のデフォルトは `Version`/ です。

内容

Dimensions

時系列データポイントのディメンションのリストが含まれます。

型: Dimension オブジェクトの配列

配列メンバー: 最大 128 項目。

必須：いいえ

MeasureName

Measure は、時系列のデータ属性を表します。例えば、EC2インスタンスまたは風力タービンRPMの のCPU使用率は測定値です。

型: 文字列

長さの制限：最小長は 1 です。最大長は 256 です。

必須：いいえ

MeasureValue

時系列データポイントの測定値が含まれます。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：いいえ

MeasureValues

時系列データポイント MeasureValue のリストが含まれます。

これはタイプ でのみ許可されますMULTI。スカラー値の場合は、レコードの MeasureValue 属性を直接使用します。

型: MeasureValue オブジェクトの配列

必須：いいえ

MeasureValueType

時系列データポイントの測定値のデータ型が含まれます。デフォルトのタイプは ですDOUBLE。詳細については、「 データ型」を参照してください。

型: 文字列

有効な値: DOUBLE | BIGINT | VARCHAR | BOOLEAN | TIMESTAMP | MULTI

必須：いいえ

Time

データポイントの測定値が収集された時刻が含まれます。時間値に単位を足して、エポックからの経過時間を指定します。例えば、時間値が 12345で、単位が の場合ms、エポックから12345 ms経過しています。

型: 文字列

長さの制限：最小長は 1 です。最大長は 256 です。

必須：いいえ

TimeUnit

タイムスタンプユニットの粒度。時間値が秒、ミリ秒、ナノ秒、またはその他のサポートされている値であるかどうかを示します。デフォルトは MILLISECONDS です。

型: 文字列

有効な値: MILLISECONDS | SECONDS | MICROSECONDS | NANOSECONDS

必須：いいえ

Version

レコードの更新に使用される 64 ビット属性。バージョン番号が大きい重複データの書き込みリクエストは、既存の測定値とバージョンを更新します。測定値が同じ場合、 `Version` は引き続き更新されます。デフォルト値は `1` です。

> **Note**
>
> `Version` は `1` 以上である必要があります。そうしないと、`ValidationException`エラーが発生します。

型: Long

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

RecordsIngested
サービス：Amazon Timestream Write

このリクエストによって取り込まれたレコードに関する情報。

内容

MagneticStore

マグネティックストアに取り込まれたレコードの数。

タイプ: 整数

必須：いいえ

MemoryStore

メモリストアに取り込まれたレコードの数。

タイプ: 整数

必須：いいえ

Total

正常に取り込まれたレコードの合計数。

タイプ: 整数

必須：いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

RejectedRecord
サービス：Amazon Timestream Write

時系列データをシステムに再挿入する前に解決する必要があるデータ検証の問題が原因で Timestream に正常に挿入されなかったレコードを表します。

内容

ExistingVersion

レコードの既存のバージョン。この値は、書き込みリクエストのバージョンよりも高いバージョンを持つ同じレコードが存在するシナリオで入力されます。

型: Long

必須：いいえ

Reason

レコードが Timestream に正常に挿入されなかった理由。考えられる障害の原因は次のとおりです。

- 同じディメンション、タイムスタンプ、およびメジャー名を持つ複数のレコードがあるが、以下の重複データを含むレコード。
  - 測定値が異なる
  - リクエストにバージョンが存在しないか、新しいレコードのバージョン値が既存の値以下です

  Timestream がこのケースのデータを拒否した場合、RejectedRecordsレスポンスの ExistingVersion フィールドには現在のレコードのバージョンが表示されます。更新を強制するには、レコードセットのバージョンを より大きい値に設定したリクエストを再送信できますExistingVersion。
- メモリストアの保持期間外のタイムスタンプを含むレコード。

> **Note**
>
> 保持ウィンドウが更新されると、新しいウィンドウ内にデータをすぐに取り込もうとすると、RejectedRecords例外が表示されます。RejectedRecords 例外を回避するには、新しいデータを取り込む新しいウィンドウの期間まで待ちます。詳細については、「Timestream の設定に関するベストプラクティス」と「Timestream でのストレージの仕組みの説明」を参照してください。

- Timestream で定義された制限を超えるディメンションまたはメジャーを持つレコード。

  詳細については、Timestream 開発者ガイドの「アクセス管理」を参照してください。

  型: 文字列

  必須: いいえ

RecordIndex

の入力リクエスト内のレコードのインデックス WriteRecords。インデックスは 0 で始まります。

タイプ: 整数

必須：いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

ReportConfiguration
サービス：Amazon Timestream Write

バッチロードタスクの設定をレポートします。これには、エラーレポートの保存先に関する詳細が含まれます。

内容

ReportS3Configuration

バッチロードのエラーレポートとイベントを書き込む S3 の場所の設定。

型: ReportS3Configuration オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

ReportS3Configuration
サービス：Amazon Timestream Write

内容

BucketName

型: 文字列

長さの制限: 最小長は 3 です。最大長は 63 です。

Pattern: `[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]`

必須：はい

EncryptionOption

型: 文字列

有効な値: `SSE_S3 | SSE_KMS`

必須: いいえ

KmsKeyId

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須: いいえ

ObjectKeyPrefix

型: 文字列

長さの制限：最小長は 1 です。最大長は 928 です。

パターン: `[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

RetentionProperties
サービス：Amazon Timestream Write

保存プロパティには、時系列データをマグネティックストアとメモリストアに保存する必要がある期間が含まれます。

内容

MagneticStoreRetentionPeriodInDays

データをマグネティックストアに保存しなければならない期間。

型: 長整数

有効範囲: 最小値は 1 です。最大値は 73000 です。

必須：はい

MemoryStoreRetentionPeriodInHours

データをメモリーストアに保存しなければならない期間。

型: 長整数

有効範囲: 最小値は 1 です。最大値は 8766 です。

必須：はい

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

S3Configuration
サービス：Amazon Timestream Write

S3 の場所を指定する設定。

内容

BucketName

S3 バケットカスタマーのバケット名。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 63 です。

パターン: `[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]`

必須: いいえ

EncryptionOption

S3 ロケーションカスタマーの暗号化オプション。オプションは、S3 マネージドキーまたは マネージド AWS キーを使用した S3 サーバー側の暗号化です。

型: 文字列

有効な値: `SSE_S3 | SSE_KMS`

必須：いいえ

KmsKeyId

AWS マネージド AWS KMS キーで暗号化するときのカスタマー S3 ロケーションのキー ID。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：いいえ

ObjectKeyPrefix

カスタマー S3 ロケーションのオブジェクトキープレビュー。

型: 文字列

長さの制限：最小長は 1 です。最大長は 928 です。

パターン: `[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

Schema
サービス：Amazon Timestream Write

スキーマは、テーブルの予想されるデータモデルを指定します。

内容

CompositePartitionKey

テーブルデータのパーティション化に使用される属性を定義するパーティションキーの空でないリスト。リストの順序によってパーティション階層が決まります。各パーティションキーの名前とタイプ、およびパーティションキーの順序は、テーブルの作成後に変更することはできません。ただし、各パーティションキーの強制レベルは変更できます。

型: PartitionKey オブジェクトの配列

配列メンバー：最小数は 1 項目です。

必須：いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

Table
サービス：Amazon Timestream Write

Timestream のデータベーステーブルを表します。テーブルには、1 つ以上の関連する時系列が含まれます。テーブルのメモリストアとマグネティックストアの保持期間を変更できます。

内容

Arn

このテーブルを一意に識別する Amazon リソースネーム。

型: 文字列

必須: いいえ

CreationTime

Timestream テーブルが作成された時刻。

型: タイムスタンプ

必須：いいえ

DatabaseName

このテーブルを含む Timestream データベースの名前。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 256 です。

必須：いいえ

LastUpdatedTime

Timestream テーブルが最後に更新された時刻。

型: タイムスタンプ

必須：いいえ

MagneticStoreWriteProperties

磁気ストアの書き込みを有効にするときに表に設定するプロパティが含まれます。

型: MagneticStoreWriteProperties オブジェクト

必須：いいえ

RetentionProperties

メモリストアと磁気ストアの保持期間。

型: RetentionProperties オブジェクト

必須：いいえ

Schema

テーブルのスキーマ。

型: Schema オブジェクト

必須：いいえ

TableName

Timestream テーブルの名前。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 256 です。

必須：いいえ

TableStatus

テーブルの現在の状態：

- DELETING - テーブルは削除中です。
- ACTIVE - テーブルの使用準備が整いました。

型: 文字列

有効な値: ACTIVE | DELETING | RESTORING

必須：いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

Tag
サービス：Amazon Timestream Write

タグは、目的、所有者、環境などによって Timestream データベースand/or table. Each tag consists of a key and an optional value, both of which you define. With tags, you can categorize databases and/orテーブルに割り当てるラベルです。

内容

Key

タグのキー。タグキーでは、大文字と小文字が区別されます。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

必須：はい

Value

タグの値。タグ値は大文字と小文字が区別され、null にすることができます。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 256 です。

必須：はい

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

Amazon Timestream クエリ

Amazon Timestream クエリでは、次のデータ型がサポートされています。

- ColumnInfo
- Datum
- DimensionMapping
- Endpoint
- ErrorReportConfiguration
- ErrorReportLocation
- ExecutionStats
- MixedMeasureMapping
- MultiMeasureAttributeMapping
- MultiMeasureMappings
- NotificationConfiguration
- ParameterMapping
- QueryInsights
- QueryInsightsResponse
- QuerySpatialCoverage
- QuerySpatialCoverageMax
- QueryStatus
- QueryTemporalRange
- QueryTemporalRangeMax
- Row
- S3Configuration
- S3ReportLocation
- ScheduleConfiguration
- ScheduledQuery
- ScheduledQueryDescription
- ScheduledQueryInsights
- ScheduledQueryInsightsResponse
- ScheduledQueryRunSummary
- SelectColumn
- SnsConfiguration

- Tag
- TargetConfiguration
- TargetDestination
- TimeSeriesDataPoint
- TimestreamConfiguration
- TimestreamDestination
- Type

ColumnInfo
サービス：Amazon Timestream Query

列名、データ型、その他の属性などのクエリ結果のメタデータが含まれます。

内容

Type

結果セット列のデータ型。データ型は、スカラーでも複合でもかまいません。スカラーデータ型は、整数、文字列、二重、ブール値などです。複雑なデータ型は、配列、行などの型です。

型: Type オブジェクト

必須：はい

Name

結果セット列の名前。結果セットの名前は、配列を除くすべてのデータ型の列で使用できます。

型: 文字列

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

Datum
サービス：Amazon Timestream Query

データムは、クエリ結果内の単一のデータポイントを表します。

内容

ArrayValue

データポイントが配列であるかどうかを示します。

型: Datum オブジェクトの配列

必須：いいえ

NullValue

データポイントが null かどうかを示します。

型: ブール値

必須：いいえ

RowValue

データポイントが行かどうかを示します。

型: Row オブジェクト

必須：いいえ

ScalarValue

データポイントが整数、文字列、二重、ブール値などのスカラー値であるかどうかを示します。

型: 文字列

必須: いいえ

TimeSeriesValue

データポイントが時系列データ型であるかどうかを示します。

型: TimeSeriesDataPoint オブジェクトの配列

必須：いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

DimensionMapping
サービス：Amazon Timestream Query

このタイプは、クエリ結果の列を宛先テーブルのディメンションにマッピングするために使用されます。

内容

DimensionValueType

ディメンションのタイプ。

型: 文字列

有効な値: VARCHAR

必須：はい

Name

クエリ結果の列名。

型: 文字列

必須: はい

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

Endpoint
サービス：Amazon Timestream Query

API 呼び出し先となる利用可能なエンドポイントとそのエンドポイントTTLの を表します。

内容

Address

エンドポイントアドレス。

型: 文字列

必須: はい

CachePeriodInMinutes

エンドポイントTTLの を分単位で指定します。

タイプ: Long

必須：はい

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

ErrorReportConfiguration
サービス：Amazon Timestream Query

エラーレポートに必要な構成。

内容

S3Configuration

エラーレポートの S3 構成。

型: S3Configuration オブジェクト

必須：はい

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

ErrorReportLocation
サービス：Amazon Timestream Query

これには、1 回のスケジュールされたクエリ呼び出しのエラーレポートの場所が含まれます。

内容

S3ReportLocation

エラーレポートが書き込まれる S3 の場所。

型: S3ReportLocation オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

ExecutionStats
サービス：Amazon Timestream Query

1 回のスケジュールされたクエリ実行の統計。

内容

BytesMetered

スケジュールされたクエリ実行を 1 回計測したバイト数。

型: Long

必須：いいえ

CumulativeBytesScanned

1 回のスケジュールされたクエリ実行に対してスキャンされたバイト数。

型: Long

必須：いいえ

DataWrites

1 回のスケジュールされたクエリ実行で取り込まれたレコードを計測したデータ書き込み。

型: Long

必須：いいえ

ExecutionTimeInMillis

スケジュールされたクエリ実行の完了に必要なミリ秒単位で測定される合計時間。

型: Long

必須：いいえ

QueryResultRows

送信先データソースに取り込む前にクエリを実行してから出力に存在する行の数。

型: Long

必須：いいえ

RecordsIngested

1 回のスケジュールされたクエリ実行に対して取り込まれたレコードの数。

型: Long

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

MixedMeasureMapping
サービス：Amazon Timestream Query

MixedMeasureMappings は、派生テーブル内の狭いメジャーと複数のメジャーの混合にデータを取り込むために使用できるマッピングです。

内容

MeasureValueType

から読み取る値のタイプsourceColumn。マッピングが の場合MULTI、 を使用します MeasureValueTypeMULTI。

型: 文字列

有効な値: `BIGINT | BOOLEAN | DOUBLE | VARCHAR | MULTI`

必須：はい

MeasureName

結果行の measure_name の値を参照します。 MeasureNameColumn が指定されている場合、このフィールドは必須です。

型: 文字列

必須: いいえ

MultiMeasureAttributeMappings

measureValueType が の場合は必須ですMULTI。MULTI 値メジャーの属性マッピング。

型: MultiMeasureAttributeMapping オブジェクトの配列

配列メンバー：最小数は 1 項目です。

必須：いいえ

SourceColumn

このフィールドは、結果をマテリアライズするためにメジャー値を読み取るソース列を参照します。

型: 文字列

必須: いいえ

TargetMeasureName

使用するターゲットメジャー名。指定しない場合、ターゲットメジャー名はデフォルトでは measure-name になります。 sourceColumn それ以外の場合は measure-name になります。

型: 文字列

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

MultiMeasureAttributeMapping
サービス：Amazon Timestream Query

MULTI 値メジャーの属性マッピング。

内容

MeasureValueType

ソース列から読み取られる属性のタイプ。

型: 文字列

有効な値: `BIGINT | BOOLEAN | DOUBLE | VARCHAR | TIMESTAMP`

必須：はい

SourceColumn

属性値が読み取られるソース列。

型: 文字列

必須: はい

TargetMultiMeasureAttributeName

派生テーブルの属性名に使用するカスタム名。指定しない場合は、ソース列名が使用されます。

型: 文字列

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

MultiMeasureMappings
サービス：Amazon Timestream Query

MixedMeasureMappings または の 1 つだけ MultiMeasureMappings を指定します。MultiMeasureMappings は、派生テーブルでデータをマルチメジャーとして取り込むために使用できます。

内容

MultiMeasureAttributeMappings

必須。クエリ結果をマッピングして複数メジャー属性のデータを取り込むために使用される属性マッピング。

型: MultiMeasureAttributeMapping オブジェクトの配列

配列メンバー：最小数は 1 項目です。

必須：はい

TargetMultiMeasureName

派生テーブル内のターゲット複数メジャー名の名前。この入力は、 measureNameColumn が指定されていない場合に必要です。 MeasureNameColumn を指定すると、その列の値がマルチメジャー名として使用されます。

型: 文字列

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

NotificationConfiguration
サービス：Amazon Timestream Query

スケジュールされたクエリの通知構成。Timestream は、スケジュールされたクエリが作成されたとき、状態が更新されたとき、または削除されたときに通知を送信します。

内容

SnsConfiguration

SNS 設定の詳細。

型: SnsConfiguration オブジェクト

必須：はい

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

ParameterMapping
サービス：Amazon Timestream Query

名前付きパラメータのマッピング。

内容

Name

パラメータ名。

型: 文字列

必須: はい

Type

クエリ結果セット内の列のデータ型が含まれます。データ型は、スカラーでも複雑でもかまいません。サポートされているスカラーデータ型は、整数、ブール値、文字列、二重、タイムスタンプ、日付、時刻、間隔です。サポートされている複雑なデータ型は、配列、行、時系列です。

型: Type オブジェクト

必須：はい

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

QueryInsights
サービス：Amazon Timestream Query

QueryInsights は、クエリの最適化、コストの削減、パフォーマンスの向上に役立つパフォーマンス調整機能です。を使用するとQueryInsights、クエリのプルーニング効率を評価し、クエリのパフォーマンスを向上させるための改善が必要な領域を特定できます。を使用するとQueryInsights、時間的および空間的なプルーニングの観点からクエリの有効性を分析し、パフォーマンスを向上させる機会を特定することもできます。具体的には、クエリが時間ベースのインデックス作成戦略とパーティションキーベースのインデックス作成戦略を使用してデータ取得を最適化する程度を評価できます。クエリのパフォーマンスを最適化するには、クエリの実行を管理する時間パラメータと空間パラメータの両方を微調整することが重要です。

によって提供される主要なメトリクスQueryInsightsは QuerySpatialCoverageと ですQueryTemporalRange。 QuerySpatialCoverageは、クエリがスキャンする空間軸の量を示し、値が低いほど効率的です。 QueryTemporalRangeは、スキャンされる時間範囲を示し、範囲が狭いほどパフォーマンスが高くなります。

の利点 QueryInsights

を使用する主な利点は次のとおりですQueryInsights。

- 非効率的なクエリの特定 – クエリによってアクセスされるテーブルの時間ベースおよび属性ベースのプルーニングに関する情報QueryInsightsを提供します。この情報は、最適ではないアクセス先のテーブルを特定するのに役立ちます。
- データモデルとパーティショニングの最適化 – QueryInsights情報を使用して、データモデルとパーティショニング戦略にアクセスして微調整できます。
- クエリの調整 – インデックスをより効果的に使用する機会QueryInsightsを強調します。

> **Note**
>
> QueryInsights 有効にして実行できるQueryAPIリクエストの最大数は、1 秒あたり 1 クエリです (QPS) 。このクエリレートを超えると、スロットリングが発生する可能性があります。

内容

Mode

を有効にするには、次のモードを提供しますQueryInsights。

- ENABLED_WITH_RATE_CONTROL - 処理中のクエリQueryInsightsを有効にします。このモードにはレート制御メカニズムも含まれており、QueryInsightsこの機能は 1 秒あたり 1 クエリに制限されます (QPS) 。
- DISABLED – を無効にしますQueryInsights。

型: 文字列

有効な値: ENABLED_WITH_RATE_CONTROL | DISABLED

必須：はい

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

QueryInsightsResponse
サービス：Amazon Timestream Query

実行したクエリに関連するさまざまなインサイトとメトリクスを提供します。

内容

OutputBytes

クエリ結果セットのサイズをバイト単位で示します。このデータを使用して、クエリ調整演習の一環として結果セットが変更されたかどうかを検証できます。

型: Long

必須：いいえ

OutputRows

クエリ結果セットの一部として返される行の合計数を示します。このデータを使用して、クエリ調整演習の一環として結果セットの行数が変更されたかどうかを検証できます。

型: Long

必須：いいえ

QuerySpatialCoverage

最適でない (最大) 空間プルーニングを持つテーブルなど、クエリの空間カバレッジに関するインサイトを提供します。この情報は、空間剪定を強化するために、パーティショニング戦略を改善すべき分野を特定するのに役立ちます。

型: QuerySpatialCoverage オブジェクト

必須：いいえ

QueryTableCount

クエリ内のテーブルの数を示します。

型: Long

必須：いいえ

QueryTemporalRange

最大 (最大) 時間範囲のテーブルなど、クエリの時間範囲に関するインサイトを提供します。以下は、時間ベースのプルーニングを最適化するための潜在的なオプションの一部です。

- 欠落している時間予測を追加します。
- 時間述語の前後で関数を削除します。
- すべてのサブクエリにタイム述語を追加します。

型: QueryTemporalRange オブジェクト

必須：いいえ

UnloadPartitionCount

`Unload` オペレーションによって作成されたパーティションを示します。

型: Long

必須：いいえ

UnloadWrittenBytes

`Unload` オペレーションによって書き込まれたバイト単位でサイズを示します。

型: Long

必須：いいえ

UnloadWrittenRows

`Unload` クエリによって書き込まれた行を示します。

型: Long

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

QuerySpatialCoverage
サービス：Amazon Timestream Query

最適でない (最大) 空間プルーニングを持つテーブルなど、クエリの空間カバレッジに関するインサイトを提供します。この情報は、空間剪定を強化するために、パーティショニング戦略を改善すべき分野を特定するのに役立ちます。

例えば、 `QuerySpatialCoverage`情報を使用して以下を実行できます。

- measure_name を追加するか、カスタマー定義のパーティションキー (CDPK) 述語を使用します。
- 前述のアクションを既に実行している場合は、それらの周りの関数や などの句を削除しますLIKE。

内容

Max

実行されたクエリの空間カバレッジと、最も非効率な空間プルーニングを持つテーブルに関するインサイトを提供します。

- `Value` – 空間カバレッジの最大比率。
- `TableArn` – 空間プルーニングが最適ではないテーブルの Amazon リソースネーム (ARN) 。
- `PartitionKey` – パーティション分割に使用されるパーティションキー。デフォルトmeasure_nameまたは にすることができますCDPK。

型: QuerySpatialCoverageMax オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

QuerySpatialCoverageMax
サービス：Amazon Timestream Query

クエリによってスキャンされた最も最適な空間範囲をテーブルにインサイトを提供します。

内容

PartitionKey

パーティショニングに使用されるパーティションキー。デフォルトmeasure_nameまたはユーザー定義のパーティションキー になります。

型: 文字列の配列

必須：いいえ

TableArn

空間プルーニングが最も最適ではないテーブルの Amazon リソースネーム (ARN) 。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：いいえ

Value

空間カバレッジの最大比率。

型: 倍精度浮動小数点数

必須：いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

QueryStatus
サービス：Amazon Timestream Query

スキャンされた進行状況やバイト数など、クエリのステータスに関する情報。

内容

CumulativeBytesMetered

クエリによってスキャンされたデータの量。請求されるバイト数。これは累積合計であり、クエリが開始されてから請求されるデータの合計量を表します。料金は 1 回のみ適用され、クエリの実行が完了したとき、またはクエリがキャンセルされたときに適用されます。

型: Long

必須：いいえ

CumulativeBytesScanned

クエリによってスキャンされたデータのバイト数。これは累積合計であり、クエリが開始されてからスキャンされたバイトの合計量を表します。

型: Long

必須：いいえ

ProgressPercentage

パーセンテージで表されるクエリの進行状況。

型: 倍精度浮動小数点数

必須：いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

QueryTemporalRange
サービス：Amazon Timestream Query

最大 (最大) 時間範囲のテーブルなど、クエリの時間範囲に関するインサイトを提供します。

内容

Max

時間軸で最もパフォーマンスが最適でないテーブルに関するインサイトを提供する次のプロパティをカプセル化します。

- `Value` – クエリの開始から終了までのナノ秒単位の最大期間。
- `TableArn` – 最大時間範囲でクエリされるテーブルの Amazon リソースネーム (ARN) 。

型: QueryTemporalRangeMax オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

QueryTemporalRangeMax
サービス：Amazon Timestream Query

クエリによってスキャンされた最も最適な時間的剪定を使用して、テーブルに関するインサイトを提供します。

内容

TableArn

最大時間範囲でクエリされるテーブルの Amazon リソースネーム (ARN) 。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：いいえ

Value

クエリの開始から終了までのナノ秒単位の最大期間。

型: Long

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

Row
サービス：Amazon Timestream Query

クエリ結果の 1 行を表します。

内容

Data

結果セットの 1 行のデータポイントのリスト。

型: Datum オブジェクトの配列

必須：はい

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

S3Configuration
サービス：Amazon Timestream Query

クエリの実行に起因するエラーレポートの S3 ロケーションの詳細。

内容

BucketName

エラーレポートが作成される S3 バケットの名前。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 63 です。

Pattern: `[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]`

必須：はい

EncryptionOption

エラーレポートの保管時の暗号化オプション。暗号化オプションが指定されていない場合、Timestream はデフォルトで SSE_S3 を選択します。

型: 文字列

有効な値: `SSE_S3 | SSE_KMS`

必須：いいえ

ObjectKeyPrefix

エラーレポートキーのプレフィックス。Timestream は、デフォルトで、エラーレポートのパスに次のプレフィックスを追加します。

型: 文字列

長さの制限：最小長は 1 です。最大長は 896 です。

パターン: `[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

S3ReportLocation
サービス：Amazon Timestream Query

スケジュールされたクエリ実行の S3 レポートの場所。

内容

BucketName

S3 バケット名です。

型: 文字列

長さの制限: 最小長は 3 です。最大長は 63 です。

パターン: `[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]`

必須: いいえ

ObjectKey

S3 キー。

型: 文字列

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

ScheduleConfiguration
サービス：Amazon Timestream Query

クエリのスケジュールの構成。

内容

ScheduleExpression

スケジュールされたクエリの実行をトリガーするタイミングを示す式。これは Cron 式でも間隔の式でも構いません。

型: 文字列

長さの制限：最小長は 1 です。最大長は 256 です。

必須：はい

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

ScheduledQuery
サービス：Amazon Timestream Query

スケジュールされたクエリ

内容

Arn

Amazon リソース名。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：はい

Name

スケジュールされたクエリの名前。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

Pattern: `[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

必須：はい

State

スケジュールされたクエリの状態。

型: 文字列

有効な値: `ENABLED | DISABLED`

必須：はい

CreationTime

スケジュールされたクエリの作成時刻。

型: タイムスタンプ

必須：いいえ

ErrorReportConfiguration

スケジュールされたクエリエラーレポートの設定。

型: ErrorReportConfiguration オブジェクト

必須：いいえ

LastRunStatus

最後にスケジュールされたクエリ実行のステータス。

型: 文字列

有効な値: `AUTO_TRIGGER_SUCCESS | AUTO_TRIGGER_FAILURE | MANUAL_TRIGGER_SUCCESS | MANUAL_TRIGGER_FAILURE`

必須：いいえ

NextInvocationTime

スケジュールされたクエリが次回実行されるとき。

型: タイムスタンプ

必須：いいえ

PreviousInvocationTime

スケジュールされたクエリが最後に実行された時刻。

型: タイムスタンプ

必須：いいえ

TargetDestination

最終的なスケジュールされたクエリ結果が書き込まれるターゲットデータソース。

型: TargetDestination オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

ScheduledQueryDescription
サービス：Amazon Timestream Query

スケジュールされたクエリを記述する構造。

内容

Arn

スケジュールされたクエリ ARN。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：はい

Name

スケジュールされたクエリの名前。

型: 文字列

長さの制限：最小長は 1 です。最大長は 64 文字です。

Pattern: `[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

必須：はい

NotificationConfiguration

通知設定。

型: NotificationConfiguration オブジェクト

必須：はい

QueryString

実行するクエリ。

型: 文字列

長さの制限：最小長は 1 です。最大長は 262,144 です。

必須：はい

ScheduleConfiguration

スケジュールの構成。

型: ScheduleConfiguration オブジェクト

必須：はい

State

スケジュールされたクエリの状態。

型: 文字列

有効な値: `ENABLED | DISABLED`

必須：はい

CreationTime

スケジュールされたクエリの作成時刻。

型: タイムスタンプ

必須：いいえ

ErrorReportConfiguration

スケジュールされたクエリのエラー報告設定。

型: ErrorReportConfiguration オブジェクト

必須：いいえ

KmsKeyId

スケジュールされたクエリリソースの暗号化に使用されるカスタマー提供のKMSキー。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：いいえ

LastRunSummary

最後にスケジュールされたクエリ実行のランタイムの概要。

型: ScheduledQueryRunSummary オブジェクト

必須：いいえ

NextInvocationTime

次回にスケジュールされたクエリの実行がスケジュールされます。

型: タイムスタンプ

必須：いいえ

PreviousInvocationTime

クエリが最後に実行された時刻。

型: タイムスタンプ

必須：いいえ

RecentlyFailedRuns

過去 5 回の失敗したスケジュールされたクエリ実行のランタイムの概要。

型: ScheduledQueryRunSummary オブジェクトの配列

必須：いいえ

ScheduledQueryExecutionRoleArn

IAM Timestream がスケジュールクエリの実行に使用する ロール。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：いいえ

TargetConfiguration

スケジュールされたクエリターゲットストア構成。

型: TargetConfiguration オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

ScheduledQueryInsights
サービス：Amazon Timestream Query

QueryInsights で を有効にするための設定をカプセル化しますExecuteScheduledQueryRequest。

内容

Mode

を有効にするには、次のモードを提供しますScheduledQueryInsights。

- ENABLED_WITH_RATE_CONTROL – 処理中のクエリScheduledQueryInsightsに対して有効にします。このモードにはレート制御メカニズムも含まれており、QueryInsightsこの機能は 1 秒あたり 1 クエリに制限されます (QPS) 。
- DISABLED – を無効にしますScheduledQueryInsights。

型: 文字列

有効な値: ENABLED_WITH_RATE_CONTROL | DISABLED

必須：はい

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

ScheduledQueryInsightsResponse
サービス：Amazon Timestream Query

実行ExecuteScheduledQueryRequestされた に関連するさまざまなインサイトとメトリクスを提供します。

内容

OutputBytes

クエリ結果セットのサイズをバイト単位で示します。このデータを使用して、クエリ調整演習の一環として結果セットが変更されたかどうかを検証できます。

型: Long

必須：いいえ

OutputRows

クエリ結果セットの一部として返される行の合計数を示します。このデータを使用して、クエリ調整演習の一環として結果セットの行数が変更されたかどうかを検証できます。

型: Long

必須：いいえ

QuerySpatialCoverage

最適でない (最大) 空間プルーニングを持つテーブルなど、クエリの空間カバレッジに関するインサイトを提供します。この情報は、空間剪定を強化するために、パーティショニング戦略を改善すべき分野を特定するのに役立ちます。

型: QuerySpatialCoverage オブジェクト

必須：いいえ

QueryTableCount

クエリ内のテーブルの数を示します。

型: Long

必須：いいえ

QueryTemporalRange

最大 (最大) 時間範囲のテーブルなど、クエリの時間範囲に関するインサイトを提供します。以下は、時間ベースのプルーニングを最適化するための潜在的なオプションの一部です。

- 欠落している時間予測を追加します。
- 時間述語の前後で関数を削除します。
- すべてのサブクエリにタイム述語を追加します。

型: QueryTemporalRange オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

ScheduledQueryRunSummary
サービス：Amazon Timestream Query

スケジュールされたクエリの実行の概要

内容

ErrorReportLocation

エラーレポートの S3 の場所。

型: ErrorReportLocation オブジェクト

必須：いいえ

ExecutionStats

スケジュールされた実行のランタイム統計。

型: ExecutionStats オブジェクト

必須：いいえ

FailureReason

障害が発生した場合のスケジュールされたクエリのエラーメッセージ。エラーレポートを確認して、より詳細なエラー理由を取得する必要がある場合があります。

型: 文字列

必須: いいえ

InvocationTime

InvocationTime この実行の場合。これは、クエリの実行がスケジュールされている時刻です。パラメータ`@scheduled_runtime`は、クエリで値を取得するために使用できます。

型: タイムスタンプ

必須：いいえ

QueryInsightsResponse

スケジュールされたクエリの実行概要に関連するさまざまなインサイトとメトリクスを提供します。

型: ScheduledQueryInsightsResponse オブジェクト

必須：いいえ

RunStatus

スケジュールされたクエリ実行のステータス。

型: 文字列

有効な値: `AUTO_TRIGGER_SUCCESS | AUTO_TRIGGER_FAILURE | MANUAL_TRIGGER_SUCCESS | MANUAL_TRIGGER_FAILURE`

必須：いいえ

TriggerTime

クエリが実行された実際の時刻。

型: タイムスタンプ

必須：いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

SelectColumn
サービス：Amazon Timestream Query

クエリによって返される列の詳細。

内容

Aliased

列名がクエリによってエイリアスされている場合、True。それ以外の場合は False。

型: ブール値

必須：いいえ

DatabaseName

この列を持つデータベース。

型: 文字列

必須: いいえ

Name

列の名前。

型: 文字列

必須: いいえ

TableName

この列を持つデータベース内のテーブル。

型: 文字列

必須: いいえ

Type

クエリ結果セット内の列のデータ型が含まれます。データ型は、スカラーでも複雑でもかまいません。サポートされているスカラーデータ型は、整数、ブール値、文字列、二重、タイムスタンプ、日付、時刻、間隔です。サポートされている複雑なデータ型は、配列、行、時系列です。

型: Type オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

SnsConfiguration
サービス：Amazon Timestream Query

通知の送信SNSに必要な の詳細。

内容

TopicArn

SNS スケジュールされたクエリステータス通知ARNが送信される トピック。

型: 文字列

長さの制限：最小長は 1 です。最大長は 2,048 です。

必須：はい

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

Tag
サービス：Amazon Timestream Query

タグは、目的、所有者、環境などによって Timestream データベースand/or table. Each tag consists of a key and an optional value, both of which you define. Tags enable you to categorize databases and/orテーブルに割り当てるラベルです。

内容

Key

タグのキー。タグキーでは、大文字と小文字が区別されます。

型: 文字列

長さの制限: 最小長は 1 です。最大長は 128 です。

必須：はい

Value

タグの値。タグ値は大文字と小文字が区別され、null にすることができます。

型: 文字列

長さの制限: 最小長は 0 です。最大長は 256 です。

必須：はい

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

TargetConfiguration
サービス：Amazon Timestream Query

クエリの出力を書き込むために使用される構成。

内容

TimestreamConfiguration

Timestream データベースとテーブルにデータを書き込むために必要な構成。

型: TimestreamConfiguration オブジェクト

必須：はい

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

TargetDestination
サービス：Amazon Timestream Query

ターゲットデータソースのデータを書き込む送信先の詳細。現在サポートされているデータソースは Timestream です。

内容

TimestreamDestination

Timestream データソースの結果の送信先の詳細をクエリします。

型: TimestreamDestination オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

TimeSeriesDataPoint
サービス：Amazon Timestream Query

時系列データ型は、時間の経過に伴うメジャーの値を表します。時系列は、タイムスタンプと測定値の行の配列で、行は時間の昇順にソートされます。 TimeSeriesDataPoint は、時系列内の単一のデータポイントです。これは、時系列の (時間、測定値) のタプルを表します。

内容

Time

測定値が収集されたタイムスタンプ。

型: 文字列

必須: はい

Value

データポイントの測定値。

型: Datum オブジェクト

必須：はい

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

TimestreamConfiguration
サービス：Amazon Timestream Query

Timestream データベースとテーブルにデータを書き込むための構成。この構成により、ユーザーはクエリ結果の選択した列を送信先テーブルの列にマッピングできます。

内容

DatabaseName

クエリ結果が書き込まれる Timestream データベースの名前。

型: 文字列

必須: はい

DimensionMappings

これにより、クエリ結果の列を宛先テーブルのディメンションにマッピングできます。

型: DimensionMapping オブジェクトの配列

必須：はい

TableName

クエリ結果が書き込まれる Timestream テーブルの名前。テーブルは、Timestream の構成で指定されるのと同じデータベース内にある必要があります。

型: 文字列

必須: はい

TimeColumn

宛先テーブルの時間列として使用する必要があるクエリ結果の列。この列タイプは である必要がありますTIMESTAMP。

型: 文字列

必須: はい

MeasureNameColumn

メジャー列の名前。

型: 文字列

必須: いいえ

MixedMeasureMappings

メジャーを複数メジャーレコードにマッピングする方法を指定します。

型: MixedMeasureMapping オブジェクトの配列

配列メンバー：最小数は 1 項目です。

必須：いいえ

MultiMeasureMappings

複数メジャーマッピング。

型: MultiMeasureMappings オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

TimestreamDestination
サービス：Amazon Timestream Query

スケジュールされたクエリの送信先。

内容

DatabaseName

Timestream データベース名。

型: 文字列

必須: いいえ

TableName

Timestream テーブル名。

型: 文字列

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

Type
サービス：Amazon Timestream Query

クエリ結果セット内の列のデータ型が含まれます。データ型は、スカラーでも複雑でもかまいません。サポートされているスカラーデータ型は、整数、ブール値、文字列、二重、タイムスタンプ、日付、時刻、間隔です。サポートされている複雑なデータ型は、配列、行、時系列です。

内容

ArrayColumnInfo

列が配列であるかどうかを示します。

型: ColumnInfo オブジェクト

必須：いいえ

RowColumnInfo

列が行かどうかを示します。

型: ColumnInfo オブジェクトの配列

必須：いいえ

ScalarType

列が文字列、整数、ブール値、二重、タイムスタンプ、日付、時刻のいずれであるかを示します。詳細については、「サポートされているデータ型」を参照してください。

型: 文字列

有効な値: `VARCHAR | BOOLEAN | BIGINT | DOUBLE | TIMESTAMP | DATE | TIME | INTERVAL_DAY_TO_SECOND | INTERVAL_YEAR_TO_MONTH | UNKNOWN | INTEGER`

必須：いいえ

TimeSeriesMeasureValueColumnInfo

列が時系列データ型であるかどうかを示します。

型: ColumnInfo オブジェクト

必須: いいえ

以下の資料も参照してください。

言語固有の のいずれかAPIでこれを使用する方法の詳細については AWS SDKs、以下を参照してください。

- AWS SDK C++ 用
- AWS SDK Java V2 用
- AWS SDK Ruby V3 用

共通エラー

このセクションでは、すべての AWS サービスのAPIアクションに共通するエラーを一覧表示します。このサービスのAPIアクションに固有のエラーについては、そのAPIアクションのトピックを参照してください。

AccessDeniedException

このアクションを実行する十分なアクセス権限がありません。

HTTP ステータスコード: 400

IncompleteSignature

リクエスト署名が AWS 標準に準拠していません。

HTTP ステータスコード: 400

InternalFailure

リクエストの処理が、不明なエラー、例外、または障害により実行できませんでした。

HTTP ステータスコード: 500

InvalidAction

リクエストされたアクション、またはオペレーションは無効です。アクションが正しく入力されていることを確認してください。

HTTP ステータスコード: 400

InvalidClientTokenId

指定された X.509 証明書または AWS アクセスキー ID は、レコードに存在しません。

HTTP ステータスコード: 403

NotAuthorized

このアクションを実行するためのアクセス許可がありません。

HTTP ステータスコード: 400

OptInRequired

AWS アクセスキー ID には、サービスのサブスクリプションが必要です。

HTTP ステータスコード: 403

RequestExpired

リクエストが、リクエストの日付スタンプから 15 分以上経過してからサービスに到達したか、リクエストの有効期限から 15 分以上経過してからサービスに到達したか (署名付き など URLs) 、リクエストの日付スタンプが 15 分以上経過してからサービスに到達しました。

HTTP ステータスコード: 400

ServiceUnavailable

サーバーの一時的な障害により、リクエストは失敗しました。

HTTP ステータスコード: 503

ThrottlingException

リクエストのスロットリングにより、リクエストが拒否されました。

HTTP ステータスコード: 400

ValidationError

入力が AWS サービスで指定された制約を満たさない。

HTTP ステータスコード: 400

共通パラメータ

次のリストには、すべてのアクションが署名バージョン 4 リクエストにクエリ文字列で署名するために使用するパラメータを示します。アクション固有のパラメータは、アクションのトピックに示されています。署名バージョン 4 の詳細については、IAM「 ユーザーガイド」の「リクエストの署名 AWS API」を参照してください。

Action

実行するアクション。

型: 文字列

必須: はい

Version

リクエストが書き込まれるAPIバージョン。 形式で表されます YYYY-MM-DD。

タイプ: 文字列

必須: はい

X-Amz-Algorithm

リクエストの署名を作成するのに使用したハッシュアルゴリズム。

条件: 認証情報をHTTP認証ヘッダーではなくクエリ文字列に含めるときに、このパラメータを指定します。

タイプ: 文字列

有効な値: `AWS4-HMAC-SHA256`

必須: 条件による

X-Amz-Credential

認証情報スコープの値で、アクセスキー、日付、対象とするリージョン、リクエストしているサービス、および終了文字列 ("aws4_request") を含む文字列です。値は access_key /YYYYMMDD/region /service /aws4_request の形式で表されます。

詳細については、IAM「 ユーザーガイド」の「署名付き AWS APIリクエストの作成」を参照してください。

条件: 認証情報をHTTP認証ヘッダーではなくクエリ文字列に含めるときに、このパラメータを指定します。

タイプ: 文字列

必須: 条件による

X-Amz-Date

署名を作成するときに使用する日付です。形式は 8601 ISO の基本形式 (YYYYMMDD'T'HHMMSS'Z') である必要があります。例えば、次の日時は有効な X-Amz-Date値です: 20120325T120000Z。

条件： X-Amz-Date はすべてのリクエストでオプションです。リクエストの署名に使用される日付を上書きするために使用できます。Date ヘッダーが 8601 ISO 基本形式で指定されている場合、 X-Amz-Date は必要ありません。を使用すると、常に Date X-Amz-Dateヘッダーの値を上書きします。詳細については、IAM「 ユーザーガイド」の AWS API「リクエスト署名の要素」を参照してください。

タイプ: 文字列

必須: 条件による

X-Amz-Security-Token

AWS Security Token Service () への呼び出しを通じて取得した一時的なセキュリティトークン AWS STS。からの一時的なセキュリティ認証情報をサポートするサービスのリストについては AWS STS、 IAM ユーザーガイドのAWS のサービス 「 で動作する IAM」を参照してください。

条件: から一時的なセキュリティ認証情報を使用している場合は AWS STS、セキュリティトークンを含める必要があります。

タイプ: 文字列

必須: 条件による

X-Amz-Signature

署名する文字列と派生署名キーから計算された 16 進符号化署名を指定します。

条件: 認証情報をHTTP認証ヘッダーではなくクエリ文字列に含めるときに、このパラメータを指定します。

タイプ: 文字列

必須: 条件による

X-Amz-SignedHeaders

正規リクエストの一部として含まれていたすべてのHTTPヘッダーを指定します。署名付きヘッダーの指定の詳細については、IAM「 ユーザーガイド」の「署名付き AWS APIリクエストの作成」を参照してください。

条件: 認証情報をHTTP認証ヘッダーではなくクエリ文字列に含めるときに、このパラメータを指定します。

タイプ: 文字列

必須: 条件による

ドキュメント履歴

変更	説明	日付
ドキュメントのみの更新	Quotas トピックを更新して、デフォルトのクォータとシステム制限を分離しました。	2024 年 10 月 22 日
Amazon Timestream がクエリインサイトをサポートするようになりました	Timestream には、クエリの最適化、パフォーマンスの向上、コスト削減に役立つクエリインサイト機能のサポートが含まれるようになりました。	2024 年 10 月 22 日
Amazon Timestream for InfluxDB が既存のポリシーに更新されました。	Amazon Timestream for InfluxDB が、ルートテーブルを記述するための `ec2:DescribeRouteTables` アクションを既存の`AmazonTimestreamInfluxDBFullAccess` 管理ポリシーに追加しました	2024 年 10 月 8 日
`AmazonTimestreamReadOnlyAccess` – 既存のポリシーの更新	の Timestream LiveAnalytics は、 AWS アカウント 設定を記述するための`DescribeAccountSettings` アクセス許可を `AmazonTimestreamReadOnlyAcc`	2024 年 6 月 3 日

	ess マネージドポリシーに追加しました。	
Amazon Timestream for が Timestream コンピューティングユニット (TCUs) をサポートする LiveAnalytics ようになりました	Amazon Timestream for LiveAnalytics には、クエリのニーズに割り当てられたコンピューティング容量を測定する Timestream コンピューティングユニット (TCUs) のサポートが含まれています。	2024 年 4 月 29 日
新しいポリシーが追加されました	Amazon Timestream for InfluxDB に 2 つの新しいポリシーが追加されました。1 つは、サービスがアカウント内のネットワークインターフェイスとセキュリティグループを管理できるようにするポリシーです。詳細については、「」を参照してくださいAmazonTimestreamInfluxDBServiceRolePolicy。Amazon Timestream InfluxDB インスタンスを作成、更新、削除、一覧表示し、パラメータグループを作成、一覧表示するための完全な管理アクセスを提供するもう 1 つの です。詳細については、「」を参照してくださいAmazonTimestreamInfluxDBFullAccess。	2024 年 3 月 14 日
Amazon Timestream for InfluxDB が一般公開されました。	このドキュメントでは、Amazon Timestream for InfluxDB の初期リリースについて説明します。	2024 年 3 月 14 日

Amazon Timestream for LiveAnalytics Query イベントは、 で利用できます。 AWS CloudTrail	Amazon Timestream for は、クエリAPIデータイベントをに発行する LiveAnalytics ようになりました AWS CloudTrail。お客様は、 AWS アカウントで行われたすべてのクエリAPIリクエストを監査し、リクエストを行ったIAMユーザー/ロール、リクエストが行われた日時、クエリされたデータベースとテーブル、リクエストのクエリ ID などの情報を確認できます。	2023 年 9 月 12 日
の Amazon Timestream LiveAnalytics UNLOAD	Amazon Timestream for LiveAnalytics では、 がクエリ結果を S3 UNLOADにエクスポートできるようになりました。	2023 年 5 月 12 日
既存のポリシー LiveAnalytics を更新するための Amazon Timestream。	マネージドポリシーに追加されたバッチロードアクセス許可。	2023 年 2 月 24 日
LiveAnalytics バッチロード用の Amazon Timestream。	Amazon Timestream for がバッチロード機能をサポートする LiveAnalytics ようになりました。	2023 年 2 月 24 日
Amazon Timestream for がをサポートする LiveAnalytics ようになりました AWS Backup。	Amazon Timestream for がをサポートする LiveAnalytics ようになりました AWS Backup。	2022 年 12 月 14 日

AWS マネージドポリシー LiveAnalytics の更新のための Amazon Timestream	既存の AWS マネージドポリシーの更新など LiveAnalytics、 の マネージドポリシーと Amazon Timestream に関する新しい情報。	2021 年 11 月 29 日
の Amazon Timestream がスケジュールされたクエリ LiveAnalytics をサポート	Amazon Timestream for では、スケジュールに基づいて、ユーザーに代わってクエリの実行がサポートされる LiveAnalytics ようになりました。	2021 年 11 月 29 日
Amazon Timestream for は、マグネティックストア LiveAnalytics をサポートしています。	Amazon Timestream for では、テーブルの書き込みにマグネティックストレージの使用がサポートされる LiveAnalytics ようになりました。	2021 年 11 月 29 日
LiveAnalytics マルチメジャーレコードの Amazon Timestream。	Amazon Timestream for は、時系列データを保存するためのよりコンパクトな形式をサポートする LiveAnalytics ようになりました。	2021 年 11 月 29 日
AWSマネージドポリシー LiveAnalytics の更新のための Amazon Timestream	既存の AWS マネージドポリシーの更新など LiveAnalytics、 の マネージドポリシーと Amazon Timestream に関する新しい情報。	2021 年 5 月 24 日
Amazon Timestream for LiveAnalytics が欧州 (フランクフルト) リージョンで利用可能になりました。	Amazon Timestream for LiveAnalytics が欧州 (フランクフルト) リージョン () で一般公開されましたeu-central-1 。	2021 年 4 月 23 日

Amazon Timestream for LiveAnalytics がVPCエンドポイント () をサポートするようになりましたAWS PrivateLink。	Amazon Timestream for では、VPCエンドポイント () の使用がサポートされる LiveAnalytics ようになりましたAWS PrivateLink。	2021 年 3 月 23 日
Amazon Timestream がクロステーブルクエリをサポートするようになりました。	の Amazon Timestream を使用して LiveAnalytics 、クロステーブルクエリを実行できます。	2021 年 2 月 10 日
Amazon Timestream for は、拡張クエリ実行統計をサポートする LiveAnalytics ようになりました。	Amazon Timestream for は、スキャンされたデータ量などの拡張クエリ実行統計をサポートする LiveAnalytics ようになりました。	2021 年 2 月 10 日
Amazon Timestream for は、高度な時系列関数をサポートする LiveAnalytics ようになりました。	LiveAnalytics の Amazon Timestream を使用して、派生、積分、相関などの高度な時系列関数を使用してSQLクエリを実行できます。	2021 年 2 月 10 日
の Amazon Timestream LiveAnalytics が HIPAA、ISO 、および PCI に準拠するようになりました。	HIPAA、、および PCI準拠のインフラストラクチャを必要とするワークロード LiveAnalytics に対して、Amazon Timestream ISOを使用できるようになりました。	2021 年 1 月 27 日

Amazon Timestream for がオープンソースのテレグラフと grafana をサポートする LiveAnalytics ようになりました。	メトリクスの収集と報告にオープンソースのプラグイン駆動型サーバーエージェントである Telegraf と、データベースのオープンソース分析とモニタリングプラットフォームである Grafana を、用の Amazon Timestream で使用できるようになりました LiveAnalytics。	2020 年 11 月 25 日
Amazon Timestream for LiveAnalytics が一般公開されました。	このドキュメントでは、 用の Amazon Timestream の初期リリースについて説明します LiveAnalytics。	2020 年 9 月 30 日

InfluxDB の Timestream とは

Amazon Timestream for InfluxDB は、アプリケーションデベロッパーや DevOps チームがオープンソース を使用してリアルタイムの時系列アプリケーション AWS 用に InfluxDB データベースを簡単に で実行できるようにするマネージド時系列データベースエンジンですAPIs。Amazon Timestream for InfluxDB を使用すると、1 桁ミリ秒のクエリ応答時間でクエリに応答できる時系列ワークロードを簡単にセットアップ、運用、スケーリングできます。

Amazon Timestream for InfluxDB では、2.x ブランチで使い慣れた InfluxDB のオープンソースバージョンの機能にアクセスできます。つまり、既存の InfluxDB オープンソースデータベースで現在既に使用しているコード、アプリケーション、ツールは、Amazon Timestream for InfluxDB とシームレスに連携する必要があります。Amazon Timestream for InfluxDB は、データベースを自動的にバックアップし、データベースソフトウェアを最新バージョンで最新の状態に保つことができます。さらに、Amazon Timestream for InfluxDB を使用すると、レプリケーションを簡単に使用してデータベースの可用性を高め、データの耐久性を向上させることができます。すべての AWS サービスと同様に、前払いの投資は必要ありません。また、使用するリソースに対してのみ料金が発生します。

DB インスタンス

DB インスタンスはクラウドで実行される独立したデータベース環境です。これは、Amazon Timestream for InfluxDB の基本的な構成要素です。DB インスタンスには、ユーザーが作成した複数のデータベース (InfluxDb 2.x データベースの場合は組織とバケット) を含めることができ、スタンドアロンのセルフマネージド InfluxDB インスタンスへのアクセスに使用するのと同じクライアントツールとアプリケーションを使用してアクセスできます。DB インスタンスは、コマンドラインツール、Amazon Timestream InfluxDB APIオペレーション、または を使用して AWS 簡単に作成および変更できます AWS Management Console。

> **Note**
>
> Amazon Timestream for InfluxDB は、Influx APIオペレーションと Influx UI を使用したデータベースへのアクセスをサポートしています。Amazon Timestream for InfluxDB では、ホストへの直接アクセスは許可されません。

InfluxDB インスタンスには、最大 40 個の Amazon Timestream を使用できます。

各 DB インスタンスには DB インスタンス名があります。この顧客提供の名前は、Amazon Timestream for InfluxDB APIおよび AWS CLI コマンドを操作するときに DB インスタンスを一意に

識別します。DB インスタンス名は、 AWS リージョン内のその顧客に対して一意である必要があります。

DB インスタンス名は、InfluxDB の Timestream によってインスタンスに割り当てられたDNSホスト名の一部を形成します。例えば、DB インスタンス名として influxdb1 を指定すると、Timestream はインスタンスにDNSエンドポイントを自動的に割り当てます。エンドポイントの例は です。ここで`influxdb1-3ksj4dla5nfjhi.us-east-1.timestream-influxdb.amazonaws.com`、`influxdb1` はインスタンス名です。

エンドポイント の例では`influxdb1-3ksj4dla5nfjhi.us-east-1.timestream-influxdb.amazonaws.com`、文字列`3ksj4dla5nfjhi`は によって生成された一意のアカウント識別子です AWS。この例`3ksj4dla5nfjhi`の識別子は、特定のリージョンで指定されたアカウントでは変更されません。したがって、このアカウントによって作成されたすべての DB インスタンスは、同じ固定識別子を共有します。固定識別子の以下の特徴を考えてみましょう。

- 現在、InfluxDB の Timestream は DB インスタンスの名前変更をサポートしていません。
- 同じ DB インスタンス識別子を持つ DB インスタンスを削除して再作成した場合、エンドポイントは同じになります。
- 同じアカウントを使用して別のリージョンに DB インスタンスを作成した場合、内部で生成される識別子は異なります。これは、`influxdb2.4a3j5du5ks7md2.us-west-1.timestream-influxdb.amazonaws.com` にあるようにリージョンが異なるためです。

各 DB インスタンスは、InfluxDB データベースエンジンの Timestream を 1 つだけサポートします。

DB インスタンスを作成する場合、InfluxDB には組織名を指定する必要があります。DB インスタンスは、複数の組織と、各組織に関連付けられた複数のバケットをホストできます。

Amazon Timestream for InfluxDB では、作成プロセスの一環として DB インスタンスのマスターユーザーアカウントとパスワードを作成できます。このマスターユーザーには、組織、バケットを作成し、データに対して読み取り、書き込み、削除、アップサート操作を実行するアクセス許可があります。また、InfluxUI にアクセスして、オペレータートークンを取得することもできます。最初の口グイン。そこから、すべてのアクセストークンを管理できます。DB インスタンスの作成時にマスターユーザーパスワードを設定する必要がありますが、Influx 、Influx API、CLIまたは InfluxUI を使用していつでも変更できます。

DB インスタンスクラス

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

DB インスタンスクラスは、DB インスタンスクラスタイプとサイズの両方で構成されます。例えば、 db.influxは、 InfluxDb ワークロードの実行に関連する高性能メモリ要件に適したメモリ最適化 DB インスタンスクラスタイプです。db.influx インスタンスクラスタイプ内で、db.influx.2xlarge は DB インスタンスクラスです。このクラスのサイズは 2xlarge です。

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

DB インスタンスクラスタイプ

Amazon Timestream for InfluxDB は、InfluxDB ユースケースに最適化された次のユースケースの DB インスタンスクラスをサポートしています。

- **db.influx**— これらのインスタンスクラスは、オープンソースの InfluxDB データベースでメモリを大量に消費するワークロードを実行するのに最適です。

DB インスタンスクラスのハードウェア仕様

次の用語では、DB インスタンスクラスのハードウェア仕様について説明します。

- vCPU

  仮想中央処理ユニットの数 (CPUs) 。仮想 CPUは、DB インスタンスクラスの比較に使用できる容量の単位です。

- メモリ (GiB)

  DB インスタンスに割り当てられた はRAM、ギビバイト単位です。多くの場合、メモリと v の比率は一貫していますCPU。例えば、r7g インスタンスクラスと同様のメモリ対 vCPU の比率を持つ db.influx EC2 インスタンスクラスを考えてみましょう。

- 流入最適化

DB インスタンスは、最適化された設定スタックを使用し、I/O 用に専用の容量を追加で提供します。このように最適化することで、I/O と、インスタンスからのその他のトラフィックとの間の競合を最小に抑え、最高のパフォーマンスを実現します。

- ネットワーク帯域幅

  他の DB インスタンスクラスとの相対的なネットワーク速度。次の表に、Amazon Timestream for InfluxDB インスタンスクラスに関するハードウェアの詳細を示します。

インスタンスクラス	vCPU	メモリ (GiB)	ストレージタイプ	ネットワーク帯域幅 (Gbps)
db.influx.medium	1	8	IOPS 含まれる流入	10
db.influx.large	2	16	IOPS 含まれる流入	10
db.influx.xlarge	4	32	IOPS 含まれる流入	10
db.influx.2xlarge	8	64	IOPS 含まれる流入	10
db.influx.4xlarge	16	128	IOPS 含まれる流入	10
db.influx.8xlarge	32	256	IOPS 含まれる流入	12
db.influx.12xlarge	48	384	IOPS 含まれる流入	20
db.influx.16xlarge	64	512	IOPS 含まれる流入	25

InfluxDB インスタンスストレージ

Amazon Timestream for InfluxDB の DB インスタンスは、データベースとログストレージに Influx IOPSに含まれるボリュームを使用します。

場合によっては、データベースワークロードがIOPSプロビジョニングした の 100% を達成できないことがあります。詳細については、「ストレージのパフォーマンスに影響する要因」を参照してください。Timestream for InfluxDB ストレージ料金の詳細については、「Amazon Timestream 料金」を参照してください。

Amazon Timestream for InfluxDB ストレージタイプ

Amazon Timestream for InfluxDB は、Influx IOPSを含む 1 つのストレージタイプをサポートします。最大 16 テビバイト (TiB) のストレージを持つ InfluxDB インスタンスの Timestream を作成できます。

使用可能なストレージタイプの簡単な説明を次に示します。

- Influx IO に含まれるストレージ: ストレージパフォーマンスは、1 秒あたりの I/O オペレーション (IOPS) と、ストレージボリュームが読み書きを実行する速度 (ストレージスループット) の組み合わせです。Influx IOPS Included ストレージボリュームでは、Amazon Timestream for InfluxDB は、さまざまなタイプのワークロードに必要な最適なIOPSスループットで事前設定された 3 つのストレージ層を提供します。

InfluxDB インスタンスのサイズ

InfluxDB インスタンスの Timestream の最適な設定は、取り込みレート、バッチサイズ、時系列カーディナリティ、同時クエリ、クエリタイプなど、多くの要因によって異なります。サイジングに関する推奨事項を提供するために、次の特徴を持つ代表的なワークロードに焦点を当てています。

- データは、データセンターからシステム、、メモリCPU、ディスク、IO などを収集する Telegraf エージェントフリートによって収集および書き込みされます。

  各書き込みリクエストには 5,000 行が含まれます。
- システムで実行されるクエリのタイプは、「中程度の複雑さ」クエリに分類されます。このカテゴリのクエリには、次の特徴があります。
  - 複数の関数と 1 つまたは 2 つの正規表現を持つ

- また、句ごとにグループ化したり、複数週の時間範囲をサンプリングしたりすることもできます。
- 通常、実行には数百ミリ秒から数千ミリ秒かかります。
- CPU は、主にクエリのパフォーマンスを優先します。

インスタンスクラス	ストレージタイプ	書き込み (1 秒あたりの行数)	読み取り (1 秒あたりのクエリ数)
db.influx.large	Influx IO に含まれる 3K	~ 50,000	<10
db.influx.2xlarge	Influx IO に含まれる 3K	~ 150,000	<25
db.influx.4xlarge	Influx IO に含まれる 3K	~ 200,000	~ 25
db.influx.4xlarge	Influx IO を含む 12K	~ 250,000	~ 35
db.influx.8xlarge	Influx IO を含む 12K	~ 500,000	~ 50
db.influx.12xlarge	Influx IO を含む 12K	<750,000	<100

AWS リージョンとアベイラビリティーゾーン

Amazon クラウドコンピューティングリソースは、世界各地の多くの場所でホストされています。これらの口ケーションは、 AWS リージョンとアベイラビリティーゾーンで構成されます。各 AWS リージョンは個別の地理的エリアです。各 AWS リージョンには、アベイラビリティーゾーンと呼ばれる複数の分離された場所があります。

> **Note**
>
> AWS リージョンのアベイラビリティーゾーンの検索については、「Amazon ユーザーガイド」の「リージョンとゾーン」を参照してください。 EC2

Amazon Timestream for InfluxDB を使用すると、DB インスタンスやデータを複数の場所に配置できます。

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

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

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

AWS リージョンの可用性

Amazon Timestream for InfluxDB が現在利用可能な AWS リージョンと各リージョンのエンドポイントの詳細については、「Amazon Timestream エンドポイントとクォータ」を参照してください。

AWS リージョン設計

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

リソースを表示すると、指定した AWS リージョンに関連付けられているリソースのみが表示されます。これは、 AWS リージョンが互いに分離されており、 AWS リージョン間でリソースを自動的にレプリケートしないためです。

AWS アベイラビリティーゾーン

DB インスタンスを作成すると、Amazon Timestream for InfluxDB はサブネット設定に基づいてランダムにインスタンスを選択します。アベイラビリティーゾーンは、 AWS リージョンコードとそれに続く文字識別子 (例:) で表されますus-east-1a。

アカウントで有効になっている指定されたリージョン内のアベイラビリティーゾーンを記述するには、次のように describe-availability-zones Amazon EC2 コマンドを使用します。

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

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

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

マルチ AZ DB デプロイ では、プライマリ DB インスタンスとセカンダリ DB インスタンスのアベイラビリティーゾーンを選択することはできません。Amazon Timesteram for InfluxDB は、ランダムにそれらを選択します。マルチ AZ デプロイの詳細については、「」を参照してくださいマルチ AZ デプロイの設定と管理。

Amazon Timestream for InfluxDB の DB インスタンス請求

Amazon Timestream for InfluxDB インスタンスには、次のコンポーネントに基づいて請求されます。

- DB インスタンス時間 (1 時間あたり) — DB インスタンスの DB インスタンスクラスに基づきます。例えば、db.influx.large です。料金は 1 時間単位で表示されますが、請求の計算方法には秒単位が適用され、時間は 10 進形式で表示されます。Amazon Timestream for InfluxDB の使用には、最低 10 分で 1 秒単位で課金されます。詳細については、DB インスタンスクラス「DB インスタンスクラス」を参照してください。
- ストレージ (1 か月あたりの GiB あたり) — DB インスタンスにプロビジョニングしたストレージ容量。詳細については、「InfluxDB インスタンスストレージ」を参照してください。
- データ転送 (GB あたり) — DB インスタンスとの間でインターネットやその他の AWS リージョンとの間で送受信されるデータ転送。

Amazon Timestream for InfluxDB の料金情報については、「Amazon Timestream for InfluxDB の料金」ページを参照してください。

InfluxDB 用の Amazon Timestream のセットアップ

Amazon Timestream for InfluxDB を初めて使用する前に、次のタスクを完了してください。

AWS アカウントを既にお持ちの場合は、Amazon Timestream for InfluxDB の要件を確認し、 IAM と VPC InfluxDB の Timestream の開始方法Getting Starting with Amazon Timestream for InfluxDB のデフォルトを使用することをお勧めします。

AWS アカウントにサインアップする

AWS アカウントがない場合は、次の手順を実行してアカウントを作成します。

アカウントにサインアップ AWS するには

- AWS サインインページに移動します。
- 新しいアカウントを作成するを選択し、指示に従います。

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

AWS アカウントにサインアップすると、 AWS アカウントルートユーザーが作成されます。ルートユーザーは、アカウント内のすべての AWS サービスとリソースにアクセスできます。セキュリティ

のベストプラクティスとして、管理ユーザーに管理アクセスを割り当て、ルートユーザーアクセスが必要なタスクを実行する場合にのみ、ルートユーザーを使用してください。

AWS は、サインアッププロセスが完了した後に確認 E メールを送信します。/ に移動し、マイアカウント を選択して、いつでも現在のアカウントアクティビティを表示https://aws.amazon.comし、アカウントを管理できます。

ユーザー管理

管理ユーザーを作成する

管理ユーザーの作成

AWS アカウントにサインアップしたら、日常的なタスクにルートユーザーを使用しないように管理ユーザーを作成します。

AWS アカウントのルートユーザーを保護する

ルートユーザーを選択し、アカウントの E メールアドレスを入力して、 AWS アカウント所有者として AWS マネジメントコンソールにサインインします。次のページでパスワードを入力します。ルートユーザーを使用してサインインする方法については、「サインインユーザーガイド」の「ルートユーザーとしてAWS サインインする」を参照してください。

ルートユーザーの多要素認証 (MFA) を有効にします。手順については、「 ユーザーガイド」の AWS 「アカウントルートユーザー (コンソール) の仮想MFAデバイスの有効化」を参照してください。 IAM

プログラムによるアクセスの許可

ユーザーが の AWS 外部とやり取りする場合は、プログラムによるアクセスが必要です AWS Management Console。プログラマチックアクセス権を付与する方法は、 AWSにアクセスしているユーザーのタイプによって異なります。

プログラムによるアクセスをユーザーに付与するには、次のいずれかのオプションを選択します。

プログラマチックアクセス権を必要とするユーザー	目的	方法
ワークフォース ID (IAMアイデンティティセンターで管理されるユーザー)	一時的な認証情報を使用して AWS CLI、 AWS SDKs、または へのプログラムによるリ	使用するインターフェイスの手順に従います。* については

プログラマチックアクセス権を必要とするユーザー	目的	方法
	クエストに署名します AWS APIs。	AWS CLI、「」を参照してください。 Identity Center AWS CLIを使用する AWS IAMように を設定する の AWS コマンドラインインターフェイスユーザーガイド .* AWS SDKs、ツール、および については AWS APIs、「」を参照してください。 IAM Identity Center 認証 の AWS SDKs および Tools リファレンスガイド 。
IAM	一時的な認証情報を使用して、 AWS CLI、、SDKsおよび へのプログラムによるリクエストに署名しますAPIs。	IAM 「 ユーザーガイド」の「AWS リソースでの一時的な認証情報の使用」の手順に従います。

プログラマチックアクセス権を必要とするユーザー	目的	方法
IAM	(推奨されません) 長期認証情報を使用して、CLI、、SDKs および へのプログラムによるリクエストに署名します AWS APIs。	使用するインターフェイスの手順に従います。 については AWS CLI、「」を参照してください。 IAM ユーザー認証情報を使用した認証 の AWS コマンドラインインターフェイスユーザーガイド . および ツールについては AWS SDKs、「」を参照してください。 長期認証情報を使用した認証 の AWS SDKs および ツールリファレンスガイド 。 については AWS APIs、「」を参照してください。 IAMユーザーのアクセスキーの管理 の IAM ユーザーガイド 。

要件の確認

Amazon Timestream for Influx の基本的な構成要素は DB インスタンスです。DB インスタンスで、バケットを作成します。DB インスタンスは、エンドポイントというネットワークアドレスを提供します。アプリケーションでは、このエンドポイントを使用して DB インスタンスに接続します。また、ブラウザから同じエンドポイントを使用して InfluxUI にアクセスします。DB インスタンスを作

成するときは、ストレージ、メモリ、データベースエンジンとバージョン、ネットワーク設定、セキュリティなどの詳細を指定します。DB インスタンスへのネットワークアクセスは、セキュリティグループを通じて制御します。

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

- リソース要件 — アプリケーションまたはサービスのメモリとプロセッサの要件は何ですか？ これらの設定を使用すると、使用する DB インスタンスクラスを判断するのに役立ちます。DB インスタンスクラスの仕様については、「DB インスタンスクラス」を参照してください。
- VPC およびセキュリティグループ — DB インスタンスは仮想プライベートクラウド (VPC) にある可能性が最も高いです。DB インスタンスに接続するには、セキュリティグループルールを設定する必要があります。これらのルールは、使用するタイプとVPC使用方法に応じて異なります。例えば、デフォルトVPCまたはユーザー定義の を使用できますVPC。

  次のリストは、各VPCオプションのルールを示しています。

  - デフォルト VPC — AWS アカウントが現在の AWS リージョンVPCにデフォルトがある場合、DB インスタンスをサポートするようにVPC設定されています。DB インスタンスの作成VPC時にデフォルトを指定する場合は、アプリケーションまたはサービスから Amazon Timestream for InfluxDB DB インスタンスへの接続を許可するVPCセキュリティグループを作成してください。VPC コンソールまたは AWS CLIでセキュリティグループオプションを使用して、VPCセキュリティグループを作成します。詳細については、「ステップ 3: VPC セキュリティグループを作成する」を参照してください。
- ユーザー定義 VPC — DB インスタンスの作成VPC時にユーザー定義を指定する場合は、次の点に注意してください。
  - アプリケーションまたはサービスから Amazon Timestream for InfluxDB DB インスタンスへの接続を許可するVPCセキュリティグループを必ず作成してください。VPC コンソールまたは AWS CLIでセキュリティグループオプションを使用して、VPCセキュリティグループを作成します。詳細については、「ステップ 3: VPC セキュリティグループを作成する」を参照してください。
  - は、それぞれが個別のアベイラビリティーゾーンに少なくとも 2 つのサブネットを持つなど、DB インスタンスをホストするために特定の要件を満たしているVPC必要があります。詳細については、「InfluxDB 用 Amazon VPCVPCsおよび Amazon Timestream」を参照してください。
- 高可用性 — フェイルオーバーサポートが必要ですか？ Amazon Timestream for InfluxDB では、マルチ AZ デプロイによって、フェイルオーバーサポート用のプライマリ DB インスタンスとセカンダリスタンバイ DB インスタンスが別のアベイラビリティーゾーンに作成されます。本番稼働用

のワークロードには、高可用性を維持するためにマルチ AZ 配置をお勧めします。開発およびテストの目的では、マルチ AZ 配置以外のデプロイを使用できます。詳細については、「マルチAZ DB インスタンスのデプロイ」を参照してください。

- IAM ポリシー — AWS アカウントには、Amazon Timestream for InfluxDB オペレーションの実行に必要なアクセス許可を付与するポリシーがありますか？ IAM 認証情報 AWS を使用して に接続する場合、IAMアカウントには、InfluxDB コントロールプレーンの Amazon Timestream オペレーションを実行するために必要なアクセス許可を付与するIAMポリシーが必要です。詳細については、「Amazon Timestream for InfluxDB の ID とアクセスの管理」を参照してください。
- オープンポート — データベースはどの TCP/IP ポートをリッスンしていますか？ 一部の企業のファイアウォールでは、データベースエンジン用のデフォルトポートへの接続がブロックされる場合があります。InfluxDB の Timestream のデフォルトは 8086 です。
- AWS リージョン — データベースをどの AWS リージョンにしたいですか？ アプリケーションやウェブサービスの近くにデータベースを配置すると、ネットワークレイテンシーを低減できます。詳細については、「AWS リージョンとアベイラビリティーゾーン」を参照してください。
- DB ディスクサブシステム — ストレージ要件は何ですか？ Amazon Timestream for InfluxDB には、Influx IOPS組み込みストレージタイプ用の 3 つの設定が用意されています。
  - Influx Io 込み 3k IOPS (SSD)
  - Influx Io を含む 12k IOPS (SSD)
  - Influx Io を含む 25k IOPS (SSD)

  Amazon Timestream for InfluxDB ストレージの詳細については、「Amazon Timestream for InfluxDB DB インスタンスストレージ」を参照してください。セキュリティグループと DB インスタンスの作成に必要な情報を把握したら、次のステップに進みます。

セキュリティグループVPCを作成して、 内の DB インスタンスへのアクセスを提供する

VPC セキュリティグループは、 内の DB インスタンスへのアクセスを提供しますVPC。セキュリティグループは、関連付けられた DB インスタンスのファイアウォールとして動作し、インバウンドトラフィックとアウトバウンドトラフィックの両方を DB インスタンスレベルで制御します。DB インスタンスはデフォルトでファイアウォールによって作成され、DB インスタンスを保護するデフォルトのセキュリティグループとなります。

DB インスタンスに接続する前に、接続を可能にするルールをセキュリティグループに追加する必要があります。ネットワークと設定に関する情報を使用して、DB インスタンスへのアクセスを許可するルールを作成します。

例えば、 の DB インスタンスのデータベースにアクセスするアプリケーションがあるとします VPC。この場合、アプリケーションがデータベースへのアクセスに使用するポート範囲と IP アドレスを指定するカスタムTCPルールを追加する必要があります。Amazon EC2インスタンスにアプリケーションがある場合は、Amazon EC2インスタンス用に設定したセキュリティグループを使用できます。

VPC アクセス用のセキュリティグループの作成

VPC セキュリティグループを作成するには、 にサインイン AWS Management Console し、 を選択しますVPC。

> **Note**
>
> Amazon Timesteam for InfluxDB VPCコンソールではなく、コンソールにあることを確認してください。

- の右上隅で AWS Management Console、VPCセキュリティグループと DB インスタンスを作成するAWSリージョンを選択します。その AWS リージョンの Amazon VPCリソースのリストには、少なくとも 1 つのサブネットVPCと複数のサブネットが表示されます。そうでない場合は、その AWS リージョンVPCにデフォルトはありません。
- ナビゲーションペインで、[Security Groups] を選択します。
- [セキュリティグループの作成] を選択します。
- セキュリティグループページの基本的な詳細セクションに、セキュリティグループ名と説明 を入力します。でVPC、DB インスタンスVPCを作成する を選択します。
- [Inbound rules] (インバウンドルール) で、[Add rule] (ルールを追加) を選択します。
  - タイプ で、カスタム TCPを選択します。
  - ソース では、セキュリティグループ名を選択するか、DB インスタンスにアクセスする IP アドレス範囲 (CIDR 値) を入力します。[マイ IP] を選択すると、ブラウザで検出された IP アドレスから DB インスタンスにアクセスできます。

Source では、セキュリティグループ名を選択するか、DB インスタンスにアクセスする IP アドレス範囲 (CIDR 値) を入力します。[マイ IP] を選択すると、ブラウザで検出された IP アドレスから DB インスタンスにアクセスできます。

- (オプション) [Outbound rules] (アウトバウンドルール) で、アウトバウンドトラフィックのルールを追加します。デフォルトではすべてのアウトバウンドトラフィックが許可されます。
- [セキュリティグループの作成] を選択します。

このVPCセキュリティグループは、作成時に DB インスタンスのセキュリティグループとして使用できます。

> **Note**
>
> デフォルトの VPCを使用する場合、 のすべてのサブネットにまたがるデフォルトのVPC サブネットグループが作成されます。DB インスタンスを作成するときは、デフォルトの eiifccntf VPCを選択し、DB サブネットグループのデフォルトを選択できます。

セットアップに必要なステップが完了したら、ユーザーの要件とセキュリティグループを利用して、DB インスタンスを作成できます。これを行うには、「DB インスタンスの作成」の手順に従います。

InfluxDB の Timestream の開始方法

次の例では、Amazon Timestream for InfluxDB Service を使用して DB インスタンスを作成して接続する方法を説明します。

> **Note**
>
> DB インスタンスを作成したり、DB インスタンスに接続したりする前に、必ず InfluxDB 用の Amazon Timestream のセットアップ のタスクを完了してください。

トピック

- InfluxDB インスタンスの Timestream を作成して接続する
- InfluxDB インスタンスの新しいオペレータトークンの作成

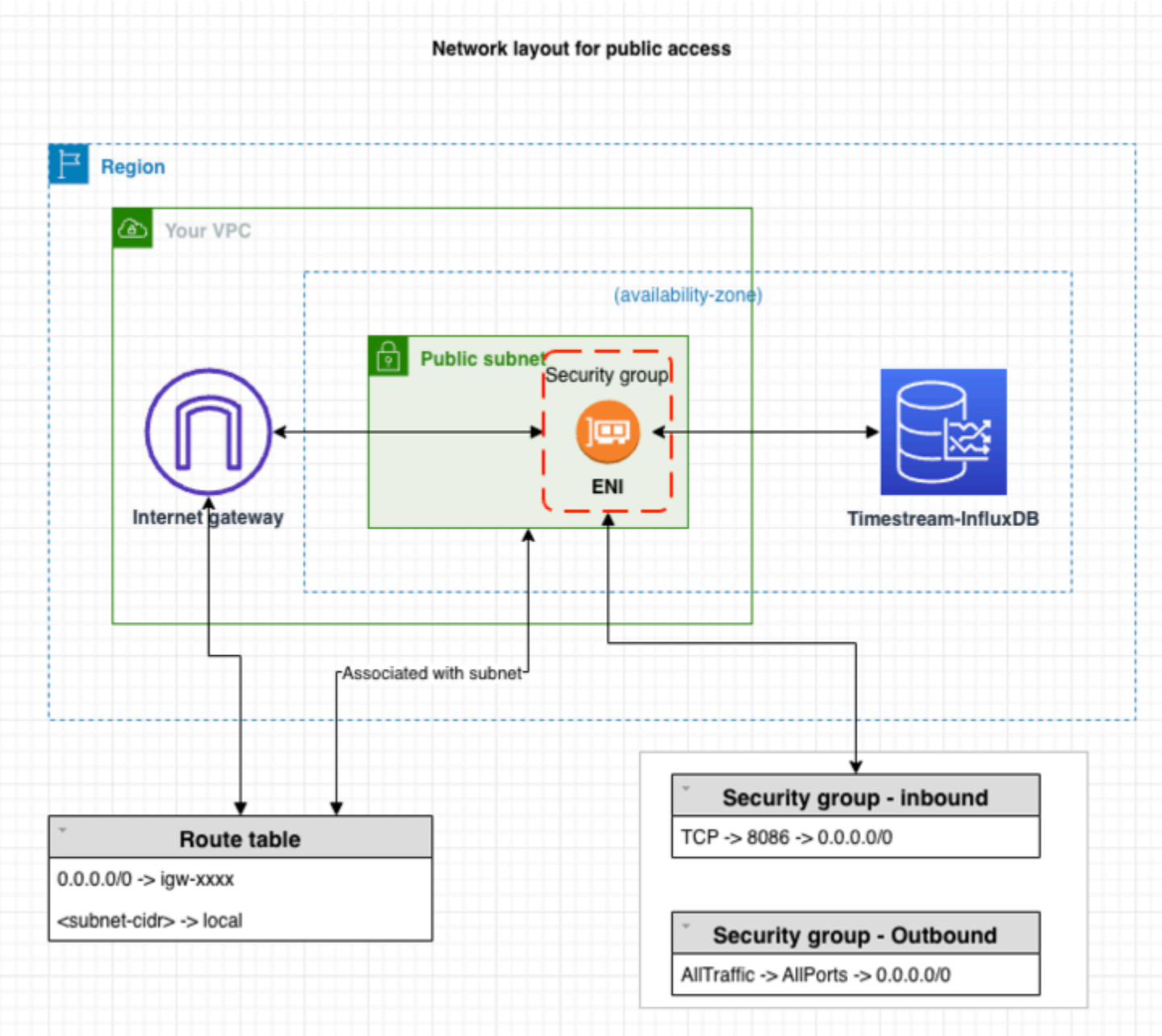
InfluxDB インスタンスの Timestream を作成して接続する

このチュートリアルでは、Amazon EC2インスタンスと Amazon Timestream for InfluxDB DB インスタンスを作成します。このチュートリアルでは、Telegraf クライアントを使用してインスタンスから DB EC2インスタンスにデータを書き込む方法を示します。ベストプラクティスとして、このチュートリアルでは仮想プライベートクラウド () にプライベート DB インスタンスを作成しますVPC。ほとんどの場合、EC2インスタンスVPCなど同じ 内の他のリソースは DB インスタンスにアクセスできますが、 外のリソースVPCはアクセスできません。

チュートリアルを完了すると、 の各アベイラビリティーゾーンにパブリックサブネットとプライベートサブネットがありますVPC。1 つのアベイラビリティーゾーンでは、EC2インスタンスはパブリックサブネットにあり、DB インスタンスはプライベートサブネットにあります。

> **Note**
>
> AWS アカウントの作成には料金はかかりません。ただし、このチュートリアルを完了すると、使用する AWS リソースのコストが発生する可能性があります。これらのリソースが不要になった場合は、チュートリアルの完了後に削除できます。

次の図は、アクセシビリティがパブリックの場合の設定を示しています。



Warning

アクセスに 0.0.0.0/0 を使用することはお勧めしません。HTTPすべての IP アドレスが 経由でパブリック InfluxDB インスタンスにアクセスできるようにするためですHTTP。このアプローチは、テスト環境で短時間でも受け入れられません。for HTTP WebUI または API アクセスを使用して InfluxDB インスタンスにアクセスするための特定の IP アドレスまたはアドレス範囲のみを許可します。

このチュートリアルでは、 で InfluxDB を実行する DB インスタンスを作成します AWS Management Console。DB インスタンスのサイズと DB インスタンス識別子のみに焦点を当てます。他の設定オプションにはデフォルト設定を使用します。この例で作成された DB インスタンスはプライベートになります。

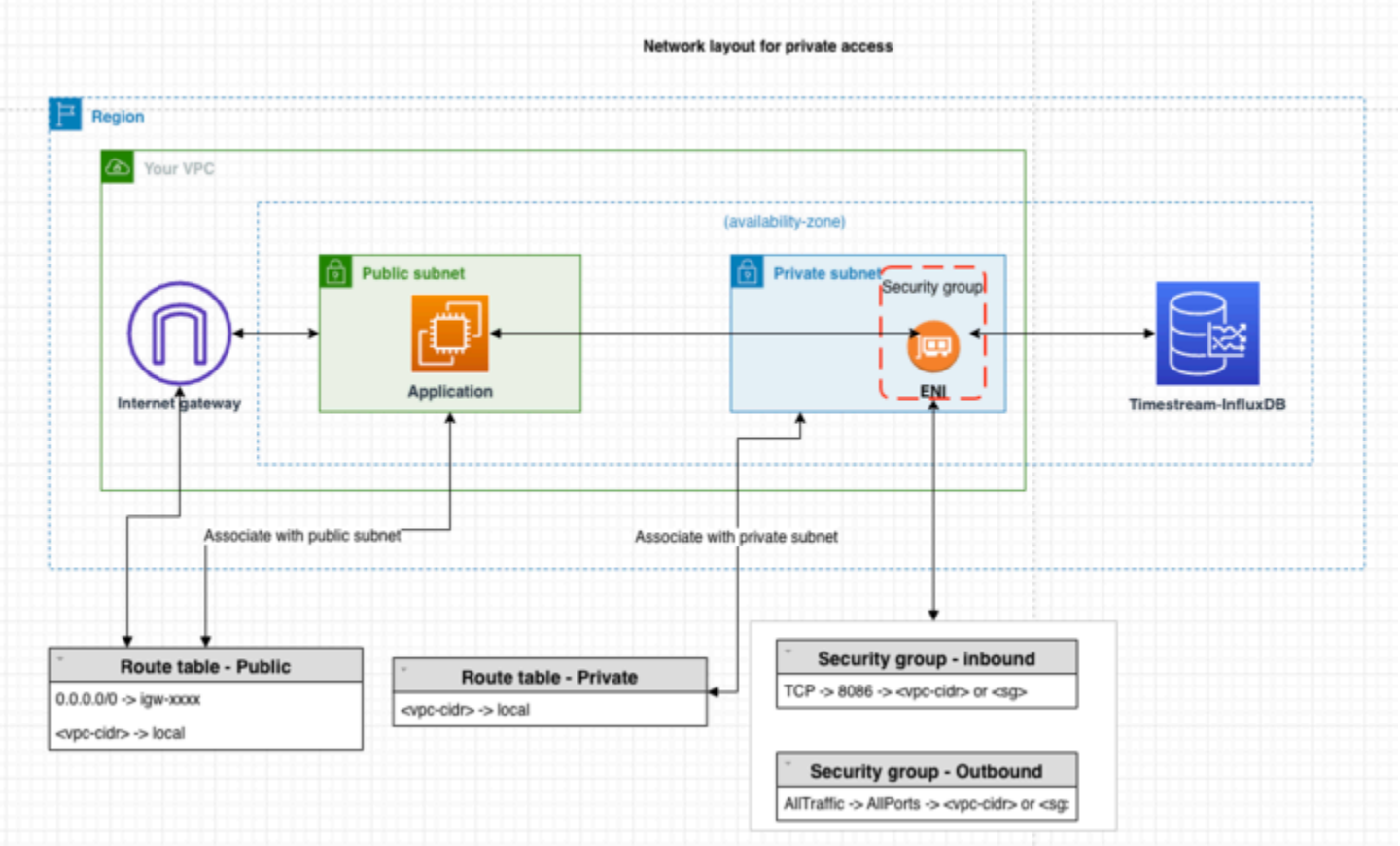
設定できるその他の設定には、可用性、セキュリティ、ログ記録などがあります。パブリック DB インスタンスを作成するには、Connectivity 設定セクションでインスタンスを「パブリックにアクセス可能」にすることを選択する必要があります。DB インスタンスの作成については、「」を参照してくださいDB インスタンスの作成。

インスタンスにパブリックにアクセスできない場合は、以下を実行します。

- トラフィックをトンネリングできるインスタンスVPCの にホストを作成します。
- インスタンスへの ssh トンネリングを設定します。詳細については、AWS 「Systems Manager を使用した Amazon EC2インスタンスポート転送」を参照してください。
- 証明書を機能させるには、クライアントマシンの`/etc/hosts`ファイルに次の行を追加します。`127.0.0.1`これはインスタンスのDNSアドレスです。
- https://<DNS>:8086 など、完全修飾ドメイン名を使用してインスタンスに接続します。

> **Note**
>
> Localhost は証明書 の一部ではないため、証明書を検証できませんSAN。

次の図は、アクセシビリティがプライベートの場合の設定を示しています。



前提条件

開始する前に、以下のセクションのステップを完了してください。

- AWS アカウントにサインアップします。
- 管理者ユーザーを作成します。

ステップ 1: Amazon EC2インスタンスを作成する

データベースへの接続に使用する Amazon EC2インスタンスを作成します。

1. にサインイン AWS Management Console し、 で Amazon EC2コンソールを開きますhttps://console.aws.amazon.com/ec2/。
2. の右上隅で AWS Management Console、EC2インスタンスを作成する AWS リージョンを選択します。
3. EC2 ダッシュボード を選択し、インスタンスを起動 を選択します。
4. インスタンスの起動ページが開いたら、インスタンスの起動ページで次の設定を選択します。

a. 名前 と タグで、名前 に ec2-database-connect と入力します。

b. Application and OS Images (Amazon Machine Image) で Amazon Linux を選択し、Amazon Linux 2023 を選択しますAMI。他の選択肢は、デフォルトの選択のままにします。

c. [Instance type] (インスタンスタイプ) で [t2.micro] を選択します。

d. [Key pair (login)] (キーペア (ログイン)) で、[Key pair name] (キーペア名) を選択して、既存のキーペアを使用します。Amazon EC2インスタンスの新しいキーペアを作成するには、新しいキーペアの作成を選択し、キーペアの作成ウィンドウを使用して作成します。新しいキーペアの作成の詳細については、「Linux インスタンス用 Amazon ユーザーガイド」の「キーペアの作成」を参照してください。 EC2

e. ネットワーク設定でSSHトラフィックを許可するには、EC2インスタンスSSHへの接続のソースを選択します。表示された IP アドレスがSSH接続に正しい場合は、My IP を選択できます。それ以外の場合は、Secure Shell () VPCを使用して、 のEC2インスタンスへの接続に使用する IP アドレスを決定できますSSH。パブリック IP アドレスを確認するには、別のブラウザウィンドウまたはタブで、 で サービスを使用できます https://checkip.amazonaws.com。IP アドレスの例は 192.0.2.1/32 です。多くの場合、インターネットサービスプロバイダー (ISP) を介して接続するか、静的 IP アドレスなしでファイアウォールの背後から接続できます。その場合、クライアントコンピュータが使用する IP アドレスの範囲を確認してください。

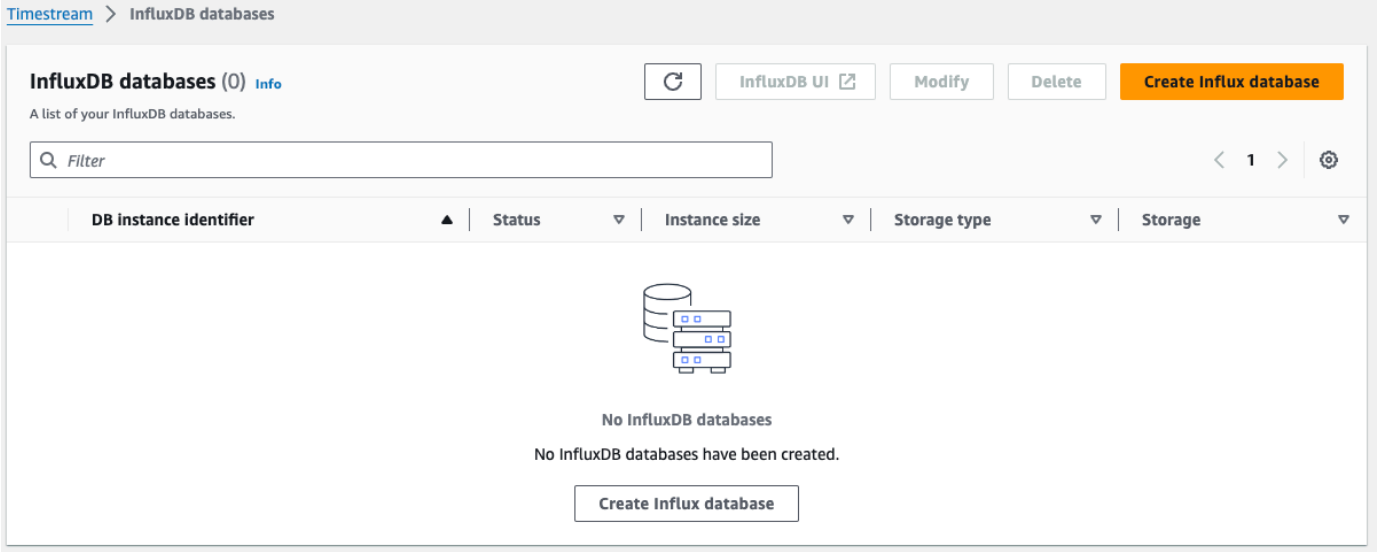
> **Warning**
>
> アクセスに 0.0.0.0/0 を使用することはお勧めしません。SSHすべての IP アドレスが を使用してパブリックEC2インスタンスにアクセスできるようにするためですSSH。このアプローチは、テスト環境で短時間でも受け入れられません。 を使用してEC2インスタンスにアクセスするには、特定の IP アドレスまたはアドレス範囲のみを承認しますSSH。

ステップ 2: InfluxDB DB インスタンスを作成する

Amazon Timestream for InfluxDB の基本的な構成要素は DB インスタンスです。この環境では、InfluxDB データベースを実行します。

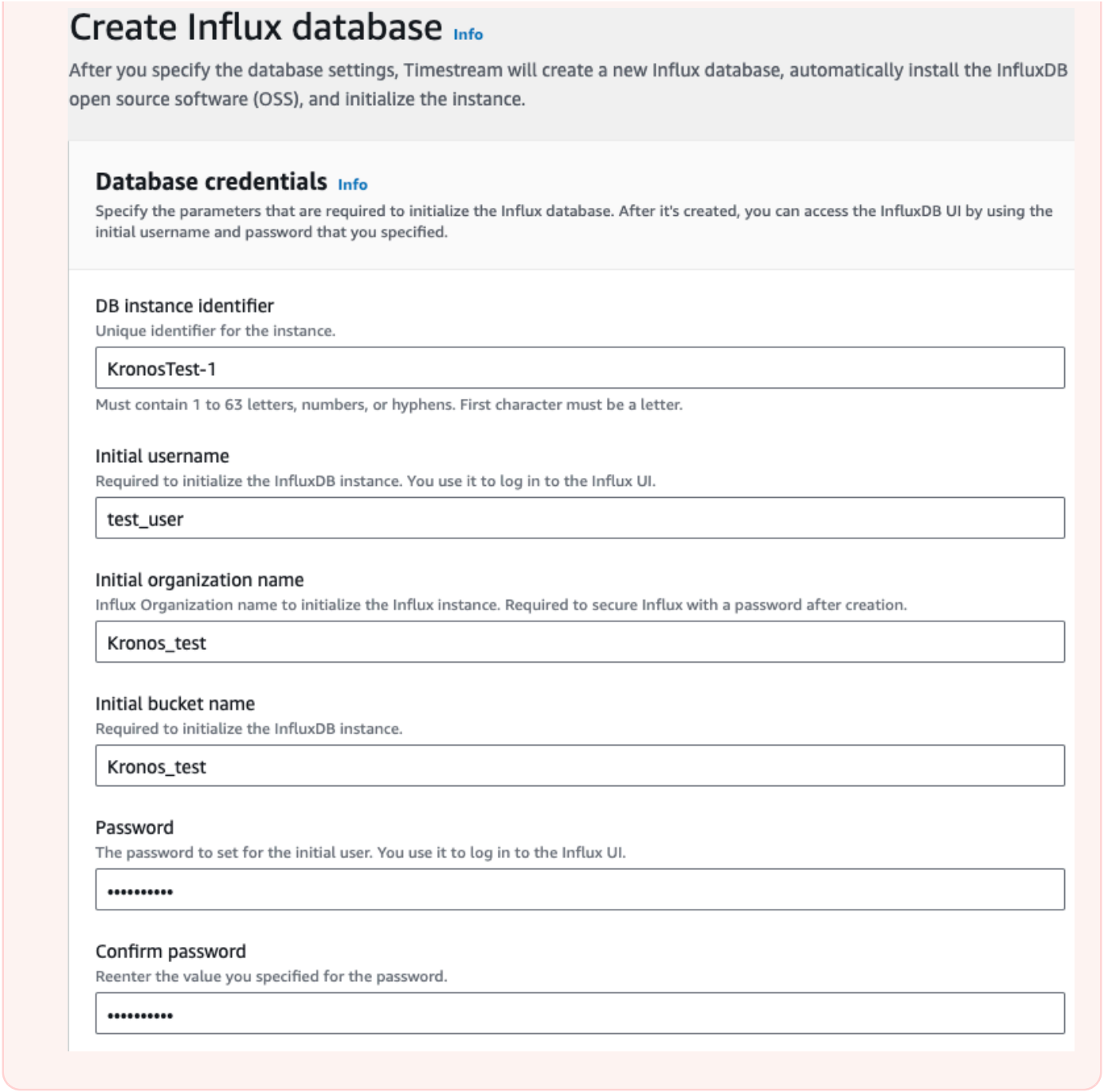
この例では、db.influx.large DB インスタンスクラスを使用して InfluxDB データベースエンジンを実行する DB インスタンスを作成します。

1. にサインイン AWS Management Console し、Amazon Timestream for InfluxDB コンソールをで開きますhttps://console.aws.amazon.com/timestream/。
2. Amazon Timestream for InfluxDB コンソールの右上隅で、DB インスタンスを作成する AWS リージョンを選択します。
3. ナビゲーションペインで、InfluxDB データベース を選択します。
4. Influx データベースの作成 を選択します。

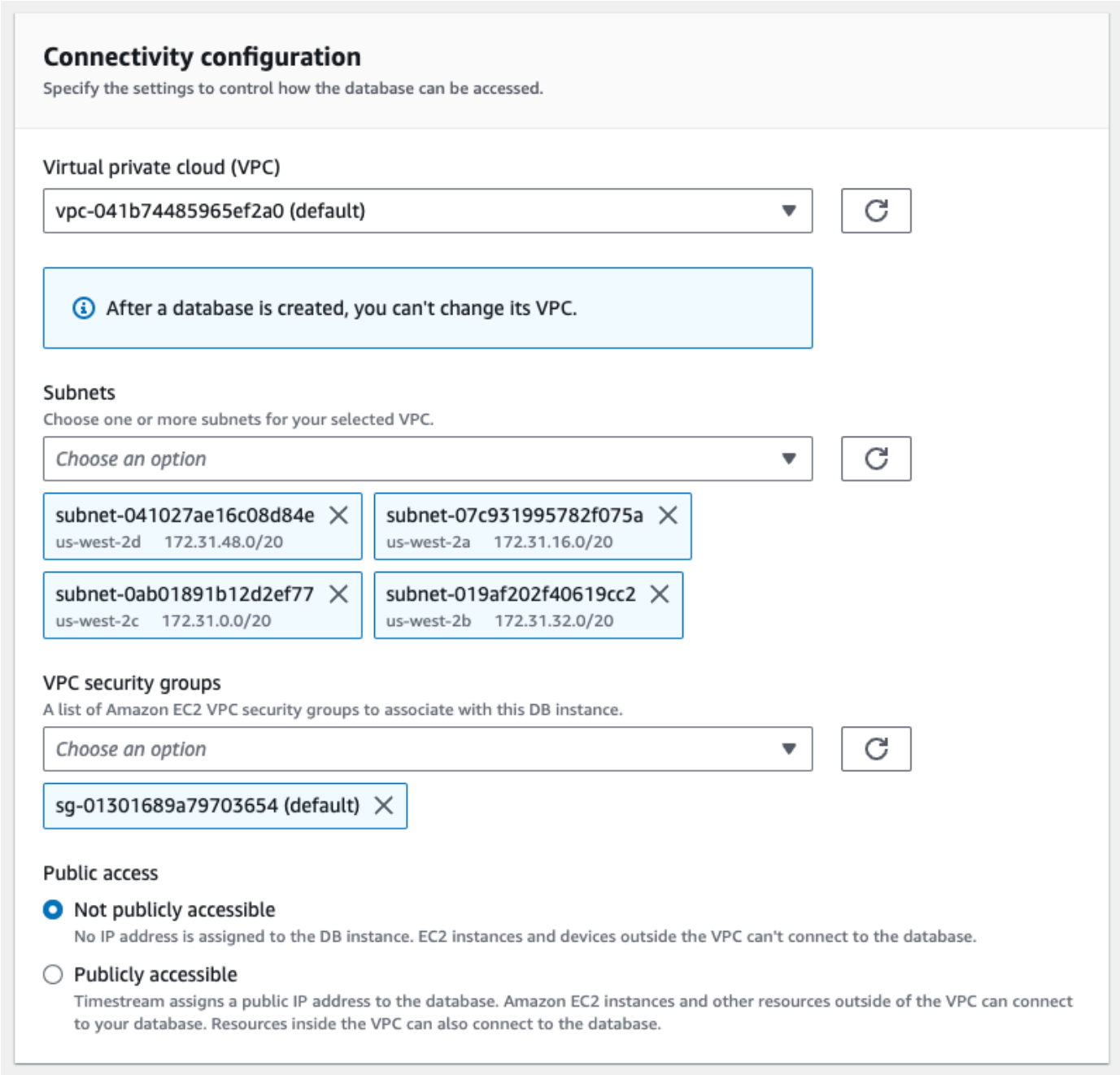


5. DB インスタンス識別子 の場合は、 KronosTest-1 と入力します。
6. InfluxDB の基本設定パラメータを指定します: ユーザー名 、組織 、バケット名 およびパスワード 。

> **Important**
>
> ユーザーパスワードを再度表示することはできません。パスワードなしでインスタンスにアクセスしてオペレータトークンを取得することはできません。記録していない場合は、変更する必要がある場合があります。「InfluxDB インスタンスの新しいオペレータトークンの作成」を参照してください。
> DB インスタンスが利用可能になった後にユーザーパスワードを変更する必要がある場合は、DB インスタンスを変更して変更することができます。DB インスタンスの変更の詳細については、「DB インスタンスの更新」を参照してください。



7. DB インスタンスクラス の場合は、db.influx.large を選択します。
8. DB ストレージクラス では、influx IOPS Included 3K を選択します。
9. ログを設定します。詳細については、「Timestream InfluxDB インスタンスで InfluxDB ログを表示するセットアップ」を参照してください。
10. 接続設定セクションで、InfluxDB インスタンスが新しく作成したEC2インスタンスと同じサブネットにあることを確認します。



11. Influx データベースの作成 を選択します。
12. データベースリストで、新しい InfluxDB インスタンスの名前を選択して詳細を表示します。DB インスタンスのステータスは、使用準備が整うまで Creating です。

ステータスが利用可能な に変わったら、DB インスタンスに接続できます。DB インスタンスクラスとストレージの合計によっては、新しいインスタンスを使用できるようになるまで最長 20 分かかることがあります。

> **Important**
>
> 現時点では、既存のインスタンスのコンピューティング (インスタンスタイプ) およびストレージ (ストレージタイプ) 設定を変更することはできません。

ステップ 3: Telegraf データを InfluxDB インスタンスに送信する

Telegraf エージェントを使用して、InfluxDB DB インスタンスへのテレメトリデータの送信を開始できるようになりました。この例では、パフォーマンスメトリクスを InfluxDB DB インスタンスに送信するように Telegraf エージェントをインストールして設定します。

1. DB インスタンスのエンドポイント (DNS名前) とポート番号を検索します。

   a. AWS マネジメントコンソールにサインインし、 で Amazon Timestream コンソールを開きますhttps://console.aws.amazon.com/timestream/。

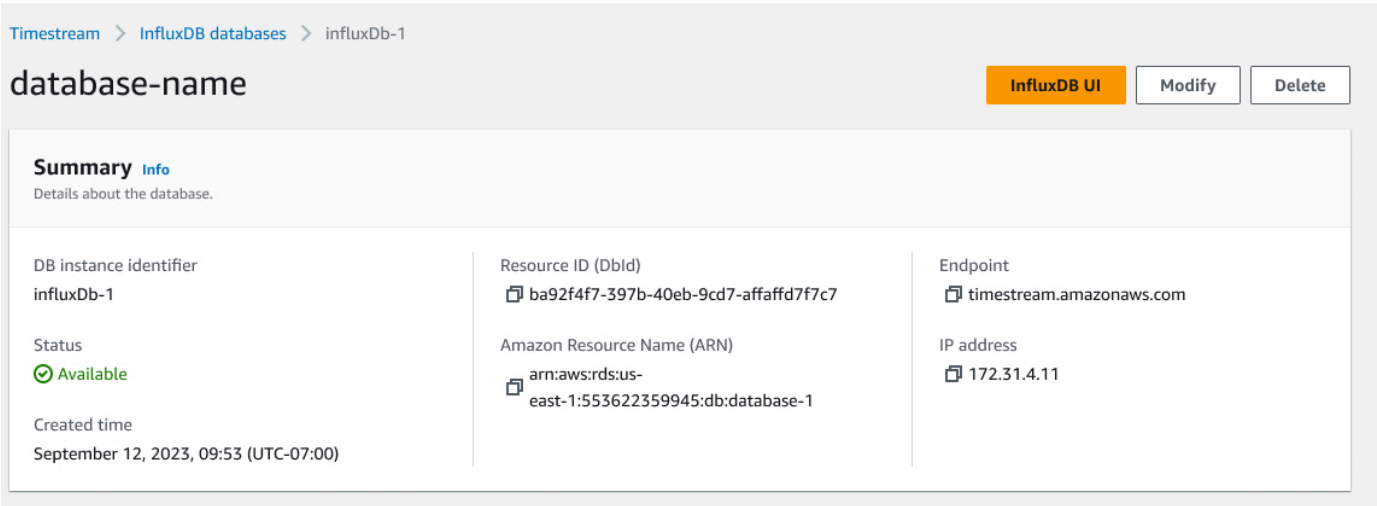
   b. Amazon Timestream コンソールの右上隅で、DB インスタンスの AWS リージョンを選択します。

   c. ナビゲーションペインで、InfluxDB データベース を選択します。

   d. InfluxDB DB インスタンス名を選択して詳細を表示します。

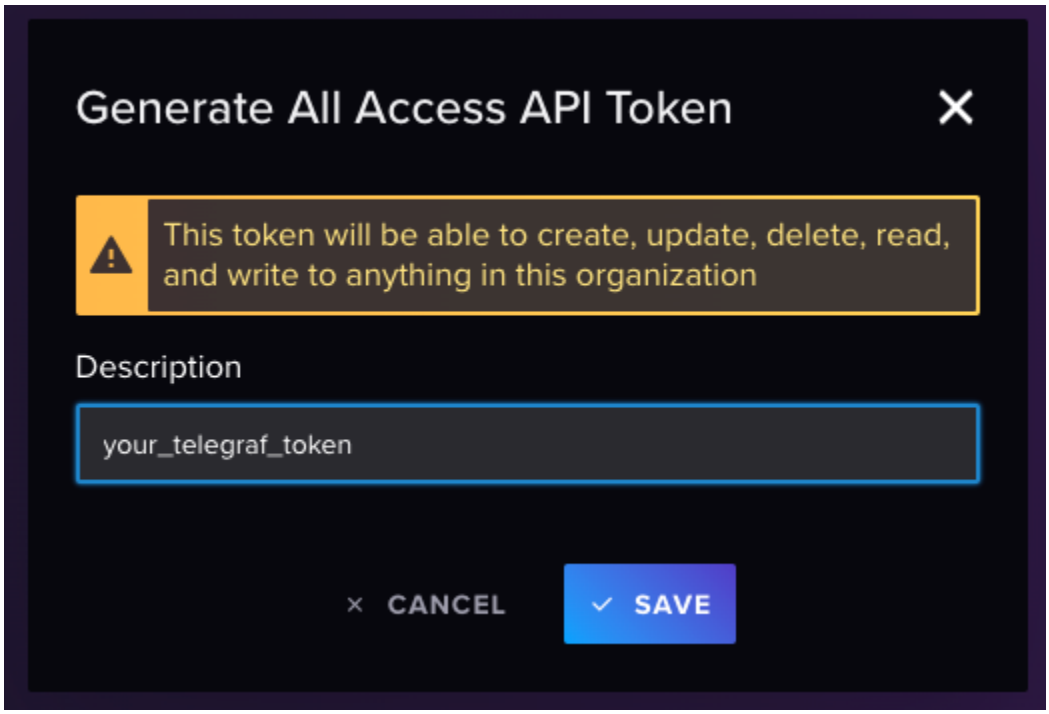
   e. 概要セクションで、エンドポイントをコピーします。また、ポート番号を書き留めます。DB インスタンスに接続するには、エンドポイントとポート番号の両方が必要です (InfluxDB のデフォルトのポート番号は 8086)。

2. 次に、InfluxDB UI を選択します。



3. これにより、ログインプロンプトが表示される新しいブラウザウィンドウが開きます。InfluxDB Db インスタンスを作成するために以前に使用した認証情報を入力します。
4. ナビゲーションペインで、矢印 をクリックし、APIトークン を選択します。
5. このテストでは、All Access Token を生成します。

> **Note**
>
> 本番稼働シナリオでは、Telegraf の特定のニーズに合わせて構築された、必要なバケットへの特定のアクセスを持つトークンを作成することをお勧めします。



6. トークンが画面に表示されます。

> **Important**
>
> トークンは再度表示できないため、必ずコピーして保存してください。

7. 「Linux EC2インスタンス用 Amazon ユーザーガイド」の「Linux インスタンスに接続する」の手順に従って、先ほど作成したインスタンスに接続します。 EC2

を使用してEC2インスタンスに接続することをお勧めしますSSH。SSH クライアントユーティリティが Windows、Linux、または Mac にインストールされている場合は、次のコマンド形式を使用してインスタンスに接続できます。

```
ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
```

例えば、 ec2-database-connect-key-pair.pemが Linux /dir1に
に保存され、EC2インスタンスIPv4DNSのパブリックが であるとしま
すec2-12-345-678-90.compute-1.amazonaws.com。SSH コマンドは次のようになります。

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

8. インスタンスにインストールされた最新バージョンの Telegraf を取得します。これを行うには、次のコマンドを使用します。

```
cat <<EOF | sudo tee /etc/yum.repos.d/influxdata.repo
[influxdata]
name = InfluxData Repository - Stable
baseurl = https://repos.influxdata.com/stable/\$basearch/main
enabled = 1
gpgcheck = 1
gpgkey = https://repos.influxdata.com/influxdata-archive_compat.key
EOF

sudo yum install telegraf
```

9. Telegraf インスタンスを設定します。

> **Note**
>
> telegraf.conf が存在しないか、timestreamセクションが含まれている場合は、以下を使用してセクションを生成できます。
>
> ```
> telegraf –section-filter agent:inputs:outputs –input-filter cpu:mem –output-
> filter timestream config > telegraf.conf
> ```

a. 通常 にある設定ファイルを編集します/etc/telegraf。

```
sudo nano /etc/telegraf/telegraf.conf
```

b. CPU、、MEMおよび の基本入力を設定しますDISK。

```
[[inputs.cpu]]
  percpu = true
  totalcpu = true
  collect_cpu_time = false
  report_active = false

[[inputs.mem]]

[[inputs.disk]]
  ignore_fs = ["tmpfs", "devtmpfs", "devfs"]
```

c. InfluxDB DB インスタンスにデータを送信し、変更を保存するように出力プラグインを設定します。

```
[[outputs.influxdb_v2]]
   urls = ["https://us-west-2-1.aws.cloud2.influxdata.com"]
   token = "<your_telegraf_token"
   organization = "your_org"
   bucket = "your_bucket"
   timeout = "5s"
```

d. Timestream ターゲットを設定します。

```
# Configuration for sending metrics to Amazon Timestream.
[[outputs.timestream]]

  ## Amazon Region and credentials
  region = "us-east-1"
  access_key = "<AWS key here>"
  secret_key = "<AWS secret key here>"
  database_name = "<timestream database name>" # needs to exist

  ## Specifies if the plugin should describe t start.
  describe_database_on_start = false
  mapping_mode = "multi-table" # allows multible tables for each input metrics
```

```
create_table_if_not_exists = true
create_table_magnetic_store_retention_period_in_days = 365
create_table_memory_store_retention_period_in_hours = 24

use_multi_measure_records = true # Important to use multi-measure records
measure_name_for_multi_measure_records = "telegraf_measure"
max_write_go_routines = 25
```

10. Telegraf サービスを有効にして開始します。

```
$ sudo systemctl enable telegraf
$ sudo systemctl start telegraf
```

ステップ 4: Amazon EC2インスタンスと InfluxDB DB インスタンスを削除する

InfluxUI で InfluxDB DB インスタンスを使用して Telegraf 生成データを探索したら、 EC2と InfluxDB DB インスタンスの両方を削除して、それらの料金が発生しないようにします。 InfluxUI

EC2インスタンスを削除するには：

1. にサインイン AWS Management Console し、 で Amazon EC2コンソールを開きますhttps://console.aws.amazon.com/ec2/。
2. ナビゲーションペインで、[インスタンス] を選択します。
3. EC2 インスタンスを選択し、インスタンスの状態 を選択し、インスタンスの終了 を選択します。
4. 確認を求めるメッセージが表示されたら、[Terminate (終了)] を選択します。

EC2 インスタンスの削除の詳細については、Amazon EC2ユーザーガイドの「インスタンスの終了」を参照してください。

最終 DB スナップショットのない DB インスタンスを削除するには：

1. にサインイン AWS Management Console し、Amazon Timestream for InfluxDB コンソールをで開きますhttps://console.aws.amazon.com/timestream/。
2. ナビゲーションペインで、InfluxDB データベース を選択します。
3. 削除する DB インスタンスを選択します。
4. [アクション] で、[削除] を選択します。

5. 確認を完了し、[削除] を選択します。

(オプション) Amazon Managed Grafana を使用して DB インスタンスに接続する

Amazon Managed Grafana を使用してダッシュボードを作成し、Amazon Timestream for InfluxDB を使用してEC2インスタンスのパフォーマンスをモニタリングできます。Amazon Managed Grafana は Grafana のフルマネージドサービスです。これは、メトリクス、ログ、トレースのクエリ、可視化、アラートを可能にする一般的なオープンソース分析プラットフォームです。

InfluxDB インスタンスの新しいオペレータトークンの作成

新しい InfluxDB インスタンスのオペレータトークンを取得する必要がある場合は、次の手順を実行します。

1. オペレータトークンを変更するには、Influx を使用することをお勧めしますCLI。手順については、「Influx をインストールして使用するCLI」を参照してください。
2. 演算子を作成できるように、 CLI を使用する`--username-password`ように を設定します。

```
influx config create --config-name CONFIG_NAME1  --host-url "https://
yourinstanceid.eu-central-1.timestream-influxdb.amazonaws.com:8086" --org [YOURORG]
  --username-password [YOURUSERNAME] --active
```

3. 新しいオペレータトークンを作成します。このステップを確認するためにパスワードの入力を求められます。

```
influx auth create --org [YOURORG] --operator
```

> **Important**
>
> 新しいオペレータトークンが作成されたら、現在古いトークンを使用しているクライアントを更新する必要があります。

セルフマネージド InfluxDB から InfluxDB の Timestream InfluxDB へのデータの移行

Influx 移行スクリプトは、OSSインスタンスが によって管理されているかどうかにかかわらず、InfluxDB インスタンス間でデータを移行する Python スクリプト AWS です。

InfluxDB は時系列データベースです。InfluxDB にはポイント が含まれており、これには多数のキーと値のペアとタイムスタンプが含まれています。ポイントがキーと値のペアでグループ化されると、それらはシリーズを形成します。シリーズは、測定 と呼ばれる文字列識別子によってグループ化されます。InfluxDB は、オペレーションのモニタリング、IOTデータ、分析によく使用されます。バケットは、InfluxDB 内のデータを保存するためのコンテナの一種です。 AWSマネージド InfluxDB は、 AWS エコシステム内の InfluxDB です。InfluxDB は、データにアクセスしてデータベースを変更APIするための InfluxDB v2 を提供します。InfluxDB v2 APIは、Influx 移行スクリプトがデータの移行に使用するものです。

- Influx 移行スクリプトは、バケットとそのメタデータを移行したり、すべての組織からすべてのバケットを移行したり、完全な移行を実行したりできます。これにより、送信先インスタンスのすべてのデータが置き換えられます。
- スクリプトは、スクリプトを実行するすべてのシステムでソースインスタンスからデータをローカルにバックアップし、データを宛先インスタンスに復元します。データは、移行ごとに 1 つずつ、code>influxdb-backup-<timestamp></timestamp> ディレクトリに保存されます。
- このスクリプトには、移行時にローカルストレージの使用を制限するための S3 バケットのマウントや、移行時に使用する組織の選択など、さまざまなオプションと設定が用意されています。

トピック

準備

InfluxDB のデータ移行は、InfluxDB CLI機能と InfluxDB v2 を利用する Python InfluxDB スクリプトを使用して行われますAPI。移行スクリプトを実行するには、次の環境設定が必要です。

- サポートされているバージョン： InfluxDB と Influx の最小バージョン 2.3 CLIがサポートされています。

- トークン環境変数
  - ソース InfluxDB インスタンスのトークンINFLUX_SRC_TOKENを含む環境変数を作成します。
  - 送信先 InfluxDB インスタンスのトークンINFLUX_DEST_TOKENを含む環境変数を作成します。
- Python 3
  - インストールを確認します。 `python3 --version`
  - インストールされていない場合は、Python ウェブサイトから をインストールします。最小バージョン 3.7 が必要です。Windows では、デフォルトの Python 3 エイリアスは単に Python です。
  - Python モジュールリクエストが必要です。次の方法でインストールします。 `shell python3 -m pip install requests`
  - TThe Python モジュール influxdb_client が必要です。次の方法でインストールします。 `shell python3 -m pip install influxdb_client`
- InfluxDB CLI
  - インストールを確認します。 `influx version`
  - インストールされていない場合は、InfluxDB ドキュメントの「インストールガイド」に従ってください。

    $ に流入を追加しますPATH。
- S3 マウントツール (オプション)

  S3 マウントを使用すると、すべてのバックアップファイルはユーザー定義の S3 バケットに保存されます。S3 マウントは、実行中のマシンのスペースを節約したり、バックアップファイルを共有する必要がある場合に役立ちます。S3 マウントを使用しない場合、 `--s3-bucket`オプションを省略することで、スクリプトが実行されたディレクトリと同じ`influxdb-backup-<millisecond timestamp>`ディレクトリにバックアップファイルを保存するためのローカルディレクトリが作成されます。

  Linux の場合: mountpoint-s3。

  Windows の場合: rclone (以前の rclone 設定が必要です) 。
- ディスク容量
  - 移行プロセスでは、提供されるプログラム引数に応じて、バックアップファイルのセットを保存するための一意のディレクトリが自動的に作成され、これらのバックアップディレクトリが S3 またはローカルファイルシステムに保持されます。

- データベースのバックアップに十分なディスク容量があることを確認し、 --s3-bucketオプションを省略し、バックアップと復元にローカルストレージを使用する場合は、既存の InfluxDB データベースのサイズを 2 倍にすることをお勧めします。
- Windows のドライブプロパティを確認して、 df -h (UNIX/Linux)または でスペースを確認します。

- 直接接続

  移行スクリプトを実行しているシステムと送信元および送信先システムの間に直接ネットワーク接続が存在することを確認します。 influx ping --host <host>は直接接続を検証する方法の 1 つです。

スクリプトの使用方法

スクリプトを実行する簡単な例は、 コマンドです。

```
python3 influx_migration.py --src-host <source host> --src-bucket <source bucket> --
dest-host <destination host>
```

1 つのバケットを移行する 。

すべてのオプションは、 を実行して表示できます。

```
python3 influx_migration.py -h
```

使用方法

```
shell   influx_migration.py [-h] [--src-bucket SRC_BUCKET] [--dest-bucket DEST_BUCKET]
 [--src-host SRC_HOST] --dest-host DEST_HOST [--full] [--confirm-full] [--src-org
 SRC_ORG] [--dest-org DEST_ORG] [--csv] [--retry-restore-dir RETRY_RESTORE_DIR] [--dir-
name DIR_NAME] [--log-level LOG_LEVEL] [--skip-verify] [--s3-bucket S3_BUCKET]
```

オプション

- -confirm-full (オプション): を --full なしで使用すると、送信先データベース内のすべてのトークン、ユーザー、バケット、ダッシュボード、およびその他のキーバリューデータが、送信元データベース内のすべてのトークン、ユーザー、バケット、ダッシュボード、およびその他のキーバリューデータ--csvに置き換えられます。 --full では、バケット組織を含むすべてのバケット

およびバケットメタデータ--csvのみが移行されます。このオプション (--confirm-full) は完全な移行を確認し、ユーザー入力なしで続行します。このオプションが指定されておらず、 --fullが指定されていて、提供--csvされていない場合、スクリプトは実行を一時停止し、ユーザーの確認を待ちます。これは重要なアクションです。注意して続行してください。デフォルトは false です。

- -csv (オプション): バックアップと復元に csv ファイルを使用するかどうか。--full も渡されると、システムバケット、ユーザー、トークン、ダッシュボードではなく、すべての組織内のすべてのユーザー定義バケットが移行されます。既存のソース組織ではなく、送信先サーバー内のすべてのバケットに単一の組織が必要な場合は、 を使用します--dest-org。
- -dest-bucket DEST_BUCKET (オプション): 送信先サーバー内の InfluxDB バケットの名前は、既存のバケットであってはなりません。指定None--src-bucketしない場合、デフォルト値は --src-bucketまたは になります。
- -dest-host DEST_HOST: 送信先サーバーのホスト。例: http://localhost:8086.
- -dest-org DEST_ORG (オプション): 宛先サーバーでバケットを復元する組織の名前。これを省略すると、ソースサーバーから移行されたすべてのバケットは元の組織を保持し、移行されたバケットは、組織を作成および切り替えることなく宛先サーバーに表示されない可能性があります。この値は、単一のバケット、完全な移行、またはバックアップと復元に csv ファイルを使用する移行のいずれであっても、すべての形式の復元に使用されます。
- -dir-name DIR_NAME (オプション): 作成するバックアップディレクトリの名前。デフォルトは influxdb-backup-<timestamp> です。まだ存在してはいけません。
- -full (オプション): 完全な復元を実行するかどうか、送信先サーバーのすべてのデータを、トークン、ダッシュボード、ユーザーなどのすべてのキーバリューデータを含む、すべての組織のソースサーバーからのすべてのデータに置き換えます。--src-bucket と を上書きします--dest-bucket。とともに使用する場合--csv、 はバケットのデータとメタデータのみを移行します。デフォルトは false です。
- h, --help : ヘルプメッセージと終了を表示します。
- -log-level LOG_LEVEL (オプション): 実行時に使用されるログレベル。オプションはデバッグ、エラー、情報です。デフォルトは info です。
- -retry-restore-dir RETRY_RESTORE_DIR (オプション): 以前の復元が失敗したときに復元に使用するディレクトリ。バックアップとディレクトリの作成をスキップします。ディレクトリが存在しない場合、失敗します。S3 バケット内のディレクトリである可能性があります。復元が失敗すると、復元に使用できるバックアップディレクトリパスが現在のディレクトリに対して表示されます。S3 バケットは の形式になりますinfluxdb-backups/<s3 bucket>/<backup directory>。デフォルトのバックアップディレクトリ名は ですinfluxdb-backup-<timestamp>。

- -s3-bucket S3_BUCKET (オプション): バックアップファイルの保存に使用する S3 バケットの名前。Linux では、指定された AWS_ACCESS_KEY_IDやAWS_SECRET_ACCESS_KEY環境変数が設定または${HOME}/.aws/credentials存在するなどの my-bucketS3 バケットの名前にすぎません。Windows では、これは などのrclone設定されたリモート名とバケット名ですmy-remote:my-bucket。すべてのバックアップファイルは、作成されたinfluxdb-backups-<timestamp>ディレクトリへの移行後に S3 バケットに残ります。という名前の一時マウントディレクトリinflux-backupsは、このスクリプトが実行されたディレクトリに作成されます。指定しない場合、すべてのバックアップファイルは、このスクリプトが実行される作成されたinfluxdb-backups-<timestamp>ディレクトリにローカルに保存されます。
- -skip-verify (オプション): TLS証明書の検証をスキップします。
- -src-bucket SRC_BUCKET (オプション): ソースサーバー内の InfluxDB バケットの名前。指定されていない場合は、 を指定--fullする必要があります。
- -src-host SRC_HOST (オプション): ソースサーバーのホスト。デフォルトは http://localhost:8086.

前述のように、 rclone --s3-bucketを使用する場合は mountpoint-s3 と が必要ですが、ユーザーが の値を指定しない場合、無視できます。この場合--s3-bucket、バックアップファイルは一意のディレクトリにローカルに保存されます。

移行の概要

前提条件を満たした後 :

1. 移行スクリプトの実行: 任意のターミナルアプリケーションを使用して Python スクリプトを実行して、ソース InfluxDB インスタンスから宛先 InfluxDB インスタンスにデータを転送します。
2. 認証情報の提供: CLIオプションとしてホストアドレスとポートを提供します。
3. データの検証: 次の方法でデータが正しく転送されていることを確認します。

    a. InfluxDB UI の使用とバケットの検査。

    b. を使用してバケットを一覧表示しますinflux bucket list -t <destination token> --host <destination host address> --skip-verify。

    c. を使用して実行influx v1 shell -t <destination token> --host <destination host address> --skip-verifyしSELECT * FROM <migrated bucket>.<retention period>.<measurement name> LIMIT 100 to view contents of a bucket or SELECT COUNT(*) FROM <migrated bucket>.<retention period>.<measurment name>、正しい数のレコードが移行されたことを確認します。

Example 実行例

1. 任意のターミナルアプリを開き、必要な前提条件が適切にインストールされていることを確認します。

```
~ > python3 --version
Python 3.11.5
~ > influx version
Influx CLI 2.7.3 (git: 8b962c7e75) build_date: 2023-04-28T14:22:49Z
~ > s3fs --version
Amazon Simple Storage Service File System V1.92 (commit:unknown) with GnuTLS(gcrypt)
Copyright (C) 2010 Randy Rizun <rrizun@gmail.com>
License GPL2: GNU GPL version 2 <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
~ >
```

2. 移行スクリプトに移動します。

```
~ > cd sample-code/influxdb-sample/migration/influxdb
~/sample-code/influxdb-sample/migration/influxdb > ls *.py
influx_migration.py
~/sample-code/influxdb-sample/migration/influxdb >
```

3. 次の情報を準備します。

   a. 移行するソースバケットの名前。

   b. (オプション) 移行先サーバーの移行されたバケットの新しいバケット名を選択します。

   c. 送信元と送信先の流入インスタンスのルートトークン。

   d. 送信元と送信先の Influx インスタンスのホストアドレス。

   e. (オプション) S3 バケット名と認証情報。 AWS Command Line Interface 認証情報は OS 環境変数で設定する必要があります。

```
# AWS credentials (for timestream testing)
    export AWS_ACCESS_KEY_ID="xxx"
    export AWS_SECRET_ACCESS_KEY="xxx"
```

   f. コマンドを次のように作成します。

```
python3 influx_migration.py --src-bucket [source-bucket-name]  --dest-bucket
 [dest-bucket-name] --src-host [source host] --dest-host [dest host] --s3-
bucket [s3 bucket name](optional) --log-level debug
```

g. スクリプトを実行します。

```
~/sample-code/influxdb-sample/migration/influxdb > python3 influx_migration.py --src-bucket primary-bucket --src-host $INFLUXDB_1_HOST --dest-host $KRO
NOS_HOST --dest-bucket new-bucket-name
```

h. スクリプトの実行が完了するまで待ちます。

i. 新しく移行されたバケットのデータ整合性を確認しますperformance.txt。このファイルは、スクリプトが実行されたディレクトリと同じディレクトリにあり、各ステップにかかる時間に関する基本的な情報が含まれています。

移行シナリオ

Example 例 1: ローカルストレージを使用した簡単な移行

1 つのバケットであるプライマリバケットをソースサーバーから(http://localhost:8086)宛先サーバー に移行します(http://dest-server-address:8086)。

ポート 8086 で InfluxDB インスタンスをホストする両方のマシンTCPへのアクセス (HTTPアクセス用) があり、ソーストークンと宛先トークンの両方がありINFLUX_DEST_TOKEN、それぞれ環境変数INFLUX_SRC_TOKEN と として保存してセキュリティを強化した後 :

```
python3 influx_migration.py --src-bucket primary-bucket --src-host http://
localhost:8086 --dest-host http://dest-server-address:8086
```

出力は次の例に類似したものになります:

```
INFO: influx_migration.py: Backing up bucket data and metadata using the InfluxDB CLI
2023/10/26 10:47:15 INFO: Downloading metadata snapshot
2023/10/26 10:47:15 INFO: Backing up TSM for shard 1
2023/10/26 10:47:15 INFO: Backing up TSM for shard 8245
2023/10/26 10:47:15 INFO: Backing up TSM for shard 8263
[More shard backups . . .]
2023/10/26 10:47:20 INFO: Backing up TSM for shard 8240
2023/10/26 10:47:20 INFO: Backing up TSM for shard 8268
2023/10/26 10:47:20 INFO: Backing up TSM for shard 2
INFO: influx_migration.py: Restoring bucket data and metadata using the InfluxDB CLI
2023/10/26 10:47:20 INFO: Restoring bucket "96c11c8876b3c016" as "primary-bucket"
2023/10/26 10:47:21 INFO: Restoring TSM snapshot for shard 12772
2023/10/26 10:47:22 INFO: Restoring TSM snapshot for shard 12773
[More shard restores . . .]
2023/10/26 10:47:28 INFO: Restoring TSM snapshot for shard 12825
2023/10/26 10:47:28 INFO: Restoring TSM snapshot for shard 12826
```

```
INFO: influx_migration.py: Migration complete
```

ディレクトリinfluxdb-backup-<timestamp>は、スクリプトが実行されたディレクトリに作成され、バックアップファイルを含むディレクトリに保存されます。

Example 例 2: ローカルストレージとデバッグログを使用したフル移行

すべてのバケット、トークン、ユーザー、ダッシュボードを移行し、送信先サーバー内のバケットを削除し、 --confirm-fullオプションを使用してデータベースの完全な移行をユーザーが確認せずに続行する点を除いて、上記と同じです。また、デバッグログ記録を有効にするために、パフォーマンス測定値を確認することもできます。

```
python3 influx_migration.py --full --confirm-full --src-host http://localhost:8086 --
dest-host http://dest-server-address:8086 --log-level debug
```

出力は次の例に類似したものになります:

```
INFO: influx_migration.py: Backing up bucket data and metadata using the InfluxDB CLI
2023/10/26 10:55:27 INFO: Downloading metadata snapshot
2023/10/26 10:55:27 INFO: Backing up TSM for shard 6952
2023/10/26 10:55:27 INFO: Backing up TSM for shard 6953
[More shard backups . . .]
2023/10/26 10:55:36 INFO: Backing up TSM for shard 8268
2023/10/26 10:55:36 INFO: Backing up TSM for shard 2
DEBUG: influx_migration.py: backup started at 2023-10-26 10:55:27 and took 9.41 seconds
 to run.
INFO: influx_migration.py: Restoring bucket data and metadata using the InfluxDB CLI
2023/10/26 10:55:36 INFO: Restoring KV snapshot
2023/10/26 10:55:38 WARN: Restoring KV snapshot overwrote the operator token, ensure
 following commands use the correct token
2023/10/26 10:55:38 INFO: Restoring SQL snapshot
2023/10/26 10:55:39 INFO: Restoring TSM snapshot for shard 6952
2023/10/26 10:55:39 INFO: Restoring TSM snapshot for shard 6953
[More shard restores . . .]
2023/10/26 10:55:49 INFO: Restoring TSM snapshot for shard 8268
2023/10/26 10:55:49 INFO: Restoring TSM snapshot for shard 2
DEBUG: influx_migration.py: restore started at 2023-10-26 10:55:36 and took 13.51
 seconds to run.
INFO: influx_migration.py: Migration complete
```

Example 例 3: CSV、送信先組織、および S3 バケットを使用したフル移行

前の例と同じですが、Linux または Mac を使用して S3 バケットにファイルを保存する場合は、my-s3-bucket。これにより、バックアップファイルのローカルストレージ容量の過負荷を回避できます。

```
python3 influx_migration.py --full --src-host http://localhost:8086 --dest-host http://
dest-server-address:8086 --csv --dest-org MyOrg --s3-bucket my-s3-bucket
```

出力は次の例に類似したものになります:

```
INFO: influx_migration.py: Creating directory influxdb-backups
INFO: influx_migration.py: Mounting influxdb-migration-bucket
INFO: influx_migration.py: Creating directory influxdb-backups/my-s3-bucket/influxdb-
backup-1698352128323
INFO: influx_migration.py: Backing up bucket data and metadata using the InfluxDB v2
 API
INFO: influx_migration.py: Restoring bucket data and metadata from csv
INFO: influx_migration.py: Restoring bucket some-bucket
INFO: influx_migration.py: Restoring bucket another-bucket
INFO: influx_migration.py: Restoring bucket primary-bucket
INFO: influx_migration.py: Migration complete
INFO: influx_migration.py: Unmounting influxdb-backups
INFO: influx_migration.py: Removing temporary mount directory
```

DB インスタンスの設定

このセクションでは、Amazon Timestream for InfluxDB DB インスタンスを設定する方法を示します。DB インスタンスを作成する前に、DB インスタンスを実行する DB インスタンスクラスを決定します。また、 AWS リージョンを選択して DB インスタンスを実行する場所を決定します。次に、DB インスタンスを作成します。

DB パラメータグループを使用して DB インスタンスを設定できます。DB パラメータグループは、1 つ以上の DB インスタンスに適用されるエンジン設定値のコンテナとして機能します。

使用できるパラメータは、DB エンジンと DB エンジンのバージョンによって異なります。DB インスタンスを作成するときに DB パラメータグループを指定できます。DB インスタンスを変更して指定することもできます。

> **Important**
>
> 現時点では、既存のインスタンスのコンピューティング (インスタンスタイプ) およびストレージ (ストレージタイプ) 設定を変更することはできません。

DB インスタンスの作成

コンソールを使用する場合

1. にサインイン AWS Management Console し、Amazon Timestream for InfluxDB を開きます。
2. Amazon Timestream for InfluxDB コンソールの右上隅で、DB インスタンスを作成する AWS リージョンを選択します。
3. ナビゲーションペインで、InfluxDB データベース を選択します。
4. Influx データベースの作成 を選択します。
5. DB インスタンス識別子 。インスタンスを識別する名前を入力します。
6. InfluxDB の基本設定パラメータ ユーザー名、組織、バケット名、パスワード を指定します。

   > **Important**
   >
   > ユーザー名、組織、バケット名、パスワードは、アカウント用に作成される AWS Secrets Manager にシークレットとして保存されます。

   DB インスタンスが利用可能になった後にユーザーパスワードを変更する必要がある場合は、Influx CLIを使用して変更できます。
7.
8. DB インスタンスクラス では、ワークロードのニーズにより適したインスタンスサイズを選択します。
9. DB ストレージクラス では、ニーズに合ったストレージクラスを選択します。いずれの場合も、割り当てられたストレージを設定するだけで済みます。
10. 接続設定セクションで、InfluxDB インスタンスが、InfluxDB DB インスタンスの Timestream への接続を必要とする新しいクライアントと同じサブネットにあることを確認します。DB インスタンスを公開することもできます。
11. Influx データベースの作成 を選択します。

12. データベースリストで、新しい InfluxDB インスタンスの名前を選択して詳細を表示します。DB インスタンスのステータスは、 が使用できるようになるまで 作成 です。
13. ステータスが [Available] (利用可能) に変わったら、DB インスタンスに接続できます。DB インスタンスクラスとストレージの合計によっては、新しいインスタンスを使用できるようになるまで最長 20 分かかることがあります。

の使用 CLI

を使用して DB インスタンスを作成するには AWS Command Line Interface、次のパラメータを使用して `create-db-instance` コマンドを呼び出します。

```
--name
--vpc-subnet-ids
--vpc-security-group-ids
--db-instance-type
--db-storage-type
--username
--organization
--password
--allocated-storage
```

各設定の詳細については、「DB インスタンスの設定」を参照してください。

Example 例: デフォルトのエンジン設定の使用

Linux、macOS、Unix の場合:

```
aws timestream-influxdb create-db-instance \
    --name myinfluxDbinstance \
    --allocated-storage 400 \
    --db-instance-type db.influx.4xlarge \
    --vpc-subnet-ids subnetid1 subnetid2
    --vpc-security-group-ids mysecuritygroup \
    --username masterawsuser \
    --password \
    --db-storage-type InfluxIOIncludedT2
```

Windows の場合:

```
aws timestream-influxdb create-db-instance \
```

```
    --name myinfluxDbinstance \
    --allocated-storage 400 \
    --db-instance-type db.influx.4xlarge \
    --vpc-subnet-ids subnetid1 subnetid2
    --vpc-security-group-ids mysecuritygroup \
    --username masterawsuser \
    --password \
    --db-storage-type InfluxIOIncludedT2
```

の使用 API

を使用して DB インスタンスを作成するには AWS Command Line Interface、次のパラメータを使用して `CreateDBInstance` コマンドを呼び出します。

各設定の詳細については、「DB インスタンスの設定」を参照してください。

> **Important**
>
> を受信するDBInstanceレスポンスオブジェクトの一部 influxAuthParametersSecretArn。これにより、 ARNはアカウントの SecretsManager シークレットに保持されます。InfluxDB DB インスタンスが利用可能になった後にのみ入力されます。シークレットには、`CreateDbInstance`プロセス中に提供される流入認証パラメータが含まれています。このシークレットupdates/modifications/deletionsへの READONLY のコピーは、作成された DB インスタンスには影響しません。このシークレットを削除しても、APIレスポンスは削除されたシークレットを参照しますARN。

Timestream for InfluxDB DB インスタンスの作成が完了したら、Influx をダウンロード、インストール、設定することをお勧めしますCLI。

Influx CLIは、コマンドラインから InfluxDB を操作する簡単な方法を提供します。インストールとセットアップの詳細な手順については、「Influx の使用CLI」を参照してください。

DB インスタンスの設定

DB インスタンスは、コンソール、 `create-db-instance` CLI コマンド、または `CreateDBInstance` Timestream for InfluxDB APIオペレーションを使用して作成できます。

次の表に、DB インスタンスの作成時に選択する設定の詳細を示します。

コンソール設定	説明	CLI オプションと Timestream APIパラメータ
ストレージ割り当て	DB インスタンスに割り当てるストレージの量 (ギビバイト(GiB)単位)。場合によっては、DB インスタンスに、データベースのサイズ以上のストレージを割り当てると、I/O のパフォーマンスが改善することがあります。 詳細については、「InfluxDB インスタンスストレージ」を参照してください。	CLI: `allocated-storage` API: `allocatedstorage`
バケット名	InfluxDb インスタンスを初期化するバケットの名前	CLI: `bucket` API: `bucket`
DB インスタンスのタイプ	DB インスタンスの設定。例えば、db.influx.large DB インスタンスクラスには、16 GiB メモリ、2 vCPUs、メモリ最適化があります。 可能であれば、一般的なクエリワーキングセットをメモリに保持できる大きさの DB インスタンスタイプを選択します。作業セットがメモリに保持されていると、システムによるディスクへの書き込みが回避され、これによりパフォーマンスが向上します。詳細については、「DB インスタンスクラスタイプ」を参照してください。	CLI: `db-instance-type` API: `Dbinstancetype`
DB インスタンス識別子	DB インスタンスの名前。オンプレミスのサーバーに名前を付けるのと同様に、DB インスタンスに名前を付けます。DB インスタンス識別子には最大 63 文字の英数字を含めることができ、選択した AWS リージョンのアカウントに対して一意である必要があります。	CLI: `db-instance-identifier` API: `Dbinstanceidentifier`

コンソール設定	説明	CLI オプションと Timestream APIパラメータ
DB パラメータグループ	DB インスタンスのパラメータグループ。デフォルトのパラメータグループを選択するか、カスタムパラメータグループを作成できます。 詳細については、「」を参照してくださいDB パラメータグループを使用する。	CLI: `db-parameter-group-name` API: `DBParameterGroupName`
ログ配信設定	S3 バケットの名前は、InfluxDB ログが保存されることでした。	CLI: `LogDeliveryConfiguration` API: `log-delivery-configuration`
マルチ AZ デプロイ	[Create a standby instance (スタンバイインスタンスを作成する)] を選択して、フェイルオーバーサポート用に DB インスタンスのパッシブセカンダリレプリカを別のアベイラビリティーゾーンに作成します。本稼働環境のワークロードには、高可用性を維持するためにマルチ AZ をお勧めします。 開発およびテスト用に、[Do not create a standby instance (スタンバイインスタンスを作成しない)] を選択することもできます。 詳細については、「マルチ AZ デプロイの設定と管理」を参照してください。	CLI: `MultiAz` API: `multi-az`
パスワード	これは、InfluxDB Db インスタンスを初期化するためのマスター使用パスワードです。このパスワードを使用して InfluxUI にログインし、オペレータートークンを取得します。	CLI: `password` API: `password`

コンソール設定	説明	CLI オプションと Timestream APIパラメータ
公開アクセス	はい。DB インスタンスにパブリック IP アドレスを付与します。つまり、 の外部からアクセスできますVPC。パブリックにアクセスできるようにするには、DB インスタンスも のパブリックサブネットにある必要がありますVPC。 いいえ。DB インスタンスを 内からのみアクセスできるようにしますVPC。 の外部から DB インスタンスに接続するにはVPC、DB インスタンスにパブリックにアクセスできる必要があります。また、DB インスタンスのセキュリティグループのインバウンドルールを使用してアクセスを許可する必要があります。さらに、他の要件を満たす必要があります。	CLI: `publicly-accessible` API: `PubliclyAccessible`
ストレージタイプ	DB インスタンスのストレージタイプ ワークロードの要件に応じて、3 つの異なるタイプのプロビジョンド流入IOPS込みストレージから選択できます。 * Influx IOPSに含まれる 3000 IOPS * Influx 12000 IOPSを含む IOPS * 16000 INfluxIOPSを含む IOPS 詳細については、「InfluxDB インスタンスストレージ」を参照してください。	CLI: `db-storage-type` API: `DbStorageType`
初期ユーザー名	これは、InfluxDB DB インスタンスを初期化するマスターユーザーになります。このユーザー名を使用して InfluxUI にログインし、オペレータートークンを取得します。	CLI: `username` API: `Username`

コンソール設定	説明	CLI オプションと Timestream APIパラメータ
サブネット	この DB インスタンスに関連付ける vpc サブネット。	CLI: `vpc-subnet-ids` API: `VPCSubnetIds`
VPC セキュリティグループ (ファイアウォール)	DB インスタンスに関連付けるセキュリティグループ。	CLI: `vpc-security-group-ids` API: `VPCSecurityGroupIds`

Amazon Timestream for InfluxDB DB インスタンスへの接続

DB インスタンスに接続する前に、DB インスタンスを作成する必要があります。詳細については、DB インスタンスの作成 を参照してください。Amazon Timestream が DB インスタンスをプロビジョニングしたら、influxDB API、Influx、CLIまたは InfluxDB の互換性のあるクライアントまたはユーティリティを使用して DB インスタンスに接続します。

トピック

- Amazon Timestream for InfluxDB DB インスタンスの接続情報の検索
- データベース認証オプション
- パラメータグループを使用する

Amazon Timestream for InfluxDB DB インスタンスの接続情報の検索

DB インスタンスの接続情報には、エンドポイント、ポート、ユーザー名、パスワード、およびオペレーターやすべてのアクセストークンなどの有効なアクセストークンが含まれます。例えば、InfluxDB DB インスタンスの場合、エンドポイント値が であるとします`influxdb1-123456789.us-east-1.timestream-influxdb.amazonaws.com`。この場合、

ポート値は 8086 で、データベースユーザーは管理者 です。この情報を考慮して、接続文字列に次の値を指定します。

- ホスト名またはホスト名の場合はDNS、 を指定しますinfluxdb1-123456789.us-east-1.timestream-influxdb.amazonaws.com。
- ポートには、8086 を指定します。
- ユーザーには、admin を指定します。
- パスワードには、DB インスタンスの作成時に指定したパスワードを指定します。

> **Important**
>
> InfluxDB Db インスタンスの Timestream を作成すると、 を受け取るDBInstanceレスポンスオブジェクトの一部になりますinfluxAuthParametersSecretArn。これにより、アカウントの SecretsManager シークレットにアークが保持されます。InfluxDB DB インスタンスが利用可能になった後にのみ入力されます。シークレットには、CreateDbInstanceプロセス中に提供される流入認証パラメータが含まれています。このシークレットREADONLYへのコピーは、作成された DB インスタンスには影響しないため、updates/modifications/deletionsこのコピーは です。このシークレットを削除しても、APIレスポンスは削除されたシークレットアークを参照します。

エンドポイントは DB インスタンスごとに一意であり、ポートとユーザーの値はさまざまです。DB インスタンスに接続するには、Influx 、Influx CLIAPI、または InfluxDB と互換性のある任意のクライアントを使用できます。

DB インスタンスの接続情報を検索するには、 AWS マネジメントコンソールを使用します。AWS コマンドラインインターフェイス (AWS CLI) describe-db-instances コマンドまたは Timestream-influxdb APIGetDBInstanceオペレーションを使用することもできます。

の使用 AWS Management Console

1. にサインイン AWS Management Console し、Amazon Timestream コンソール を開きます。
2. ナビゲーションペインで、InfluxDB データベースを選択して DB インスタンスのリストを表示します。
3. DB インスタンスの名前を選択して、その詳細を表示します。

4. 概要セクションで、エンドポイントをコピーします。また、ポート番号を書き留めます。DB インスタンスに接続するには、エンドポイントとポート番号の両方が必要です。

ユーザー名とパスワードの情報を見つける必要がある場合は、設定の詳細タブを選択し、`influxAuthParametersSecretArn` を選択して Secrets Manager にアクセスします。

CLI の使用

- を使用して InfluxDB DB インスタンスの接続情報を検索するには AWS CLI、 コマンドを get-db-instance呼び出します。呼び出しで、DB インスタンス ID、エンドポイント、ポート、シーinfluxAuthParametersクレットをクエリします。

  Linux、macOS、Unix の場合:

  ```
  aws timestream-influxdb get-db-instance --identifier id \
   --query "[name,endpoint,influxAuthParametersSecretArn]"
  ```

  Windows の場合:

  ```
  aws timestream-influxdb get-db-instance --identifier id \
   --query "[name,endpoint,influxAuthParametersSecretArn]"
  ```

  出力は次のようになります。ユーザー名情報にアクセスするには、 を確認する必要がありますInfluxAuthParameterSecret。

  ```
  [
      [
          "mydb",
          "mydb-123456789012.us-east-1.timestream-influxdb.amazonaws.com",
          8086,
      ]
  ]
  ```

アクセストークンの作成

この情報により、インスタンスに接続してアクセストークンを取得または作成できるようになります。これを実現するには、いくつかの方法があります。

CLI の使用

1. まだインストールしていない場合は、influx CLIをダウンロード、インストール、設定します。
2. Influx 設定を設定するときはCLI、 `--username-password`を使用して認証します。

```
influx config create --config-name YOUR_CONFIG_NAME  --host-url "https://
yourinstance.timestream-influxdb.amazonaws.com:8086"  --org yourorg --username-
password admin --active
```

3. influx 認証作成コマンドを使用して、オペレータトークンを再作成します。このプロセスによって古いオペレータートークンが無効になることに注意してください。

```
influx auth create --org kronos --operator
```

4. 演算子トークンを取得したら、influx 認証リストコマンドを使用して、すべてのトークンを表示できます。influx 認証作成コマンドを使用して、すべてのアクセストークンを作成できます。

> **Important**
>
> オペレータトークンを取得するには、まずこのステップを実行してから、InfluxDB APIまたは を使用して新しいトークンを作成する必要がありますCLI。

Influx UI の使用

1. 作成したエンドポイントを使用して Timestream for InfluxDB インスタンスを参照し、InfluxDB UI にログインしてアクセスします。InfluxDB DB インスタンスの作成に使用するユーザー名とパスワードを使用する必要があります。この情報は、 のレスポンスオブジェクト で`influxAuthParametersSecretArn`指定された から取得できます`CreateDbInstance`。

   または、Timestream for InfluxDB 管理コンソールから InfluxUI InfluxDB を開くこともできます。

   a. にサインイン AWS Management Console し、Amazon Timestream for InfluxDB コンソールを で開きますhttps://console.aws.amazon.com/timestream/。

   b. Amazon Timestream for InfluxDB コンソールの右上隅で、DB インスタンスを作成した AWS リージョンを選択します。

c. データベースリストで、InfluxDB インスタンスの名前を選択して詳細を表示します。右上隅で、Open Influx UI を選択します。

2. InfluxUI にログインしたら、データのロードに移動し、左側のナビゲーションバーを使用してAPIトークンを作成します。
3. + を選択しGENERATEAPITOKEN、All Access API Token を選択します。
4. API トークンの説明を入力し、 を選択しますSAVE。
5. 生成されたトークンをコピーし、安全に保管するために保存します。

> **Important**
>
> InfluxUI からトークンを作成する場合、新しく作成されたトークンは 1 回だけ表示されます。コピーしない場合は、再作成する必要があるため、必ずコピーしてください。

InfluxDB の使用 API

- リクエストメソッドを使用して InfluxDB API`/api/v2/authorizations`エンドポイントにPOSTリクエストを送信します。

  リクエストには以下を含めます。

  a. ヘッダー：

     i. 認証: トークン <INFLUX_OPERATOR_TOKEN >

     ii. コンテンツタイプ: application/json

  b. リクエスト本文: 次のプロパティを持つJSON本文：

     i. ステータス：「アクティブ」

     ii. 説明: APIトークンの説明

     iii. orgID : InfluxDB 組織 ID

     iv. アクセス許可: 各オブジェクトが InfluxDB リソースタイプまたは特定のリソースのアクセス許可を表すオブジェクトの配列。各アクセス許可には、次のプロパティが含まれます。

        A. アクション：「読み取り」または「書き込み」

B. resource: アクセス許可を付与する InfluxDB リソースを表すJSONオブジェクト。各リソースには、少なくとも次のプロパティが含まれます: orgID : InfluxDB 組織 ID

C. type: リソースタイプ。InfluxDB リソースタイプの詳細については、the /api/v2/resourcesエンドポイントを使用します。

次の例では、 curlと InfluxDB を使用してすべてのアクセストークンAPIを生成します。

```
export INFLUX_HOST=https://influxdb1-123456789.us-east-1.timestream-
influxdb.amazonaws.com
export INFLUX_ORG_ID=<YOUR_INFLUXDB_ORG_ID>
export INFLUX_TOKEN=<YOUR_INFLUXDB_OPERATOR_TOKEN>

curl --request POST \
"$INFLUX_HOST/api/v2/authorizations" \
  --header "Authorization: Token $INFLUX_TOKEN" \
  --header "Content-Type: text/plain; charset=utf-8" \
  --data '{
    "status": "active",
    "description": "All access token for get started tutorial",
    "orgID": "'"$INFLUX_ORG_ID"'",
    "permissions": [
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "authorizations"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "authorizations"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "buckets"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "buckets"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "dashboards"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "dashboards"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "orgs"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "orgs"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "sources"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "sources"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "tasks"}},
```

```
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"tasks"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"telegrafs"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"telegrafs"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "users"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"users"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"variables"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"variables"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"scrapers"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"scrapers"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"secrets"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"secrets"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"labels"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"labels"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "views"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"views"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"documents"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"documents"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"notificationRules"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"notificationRules"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"notificationEndpoints"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"notificationEndpoints"}},
    {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"checks"}},
    {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"checks"}},
```

```
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "dbrp"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "dbrp"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "notebooks"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "notebooks"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "annotations"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "annotations"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "remotes"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "remotes"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "replications"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "replications"}}
    ]
  }
'
```

データベース認証オプション

Amazon Timestream for InfluxDB では、データベースユーザーを認証するための以下の方法をサポートしています。

- パスワード認証 – DB インスタンスがユーザーアカウントのすべての管理を行います。InfluxUI、CLIまたは Influx を使用して、ユーザーの作成、パスワードの指定、トークンの管理を行いますAPI。
- トークン認証 – DB インスタンスは、ユーザーアカウントのすべての管理を実行します。Influx と Influx を使用して、オペレータートークンを使用してユーザーの作成、パスワードの指定CLI、トークンの管理を行うことができますAPI。

暗号化された接続

アプリケーションから Secure Socket Layer (SSL) または Transport Layer Security (TLS) を使用して、DB インスタンスへの接続を暗号化できます。InfluxDB と Kronos サービスによって作成および管理されるアプリケーション間のTLSハンドシェイクに必要な証明書。証明書が更新されると、インスタンスはユーザーの介入を必要とせずに最新バージョンで自動的に更新されます。

パラメータグループを使用する

データベースパラメータは、データベースの設定方法を指定します。データベースパラメータでは、データベースに割り当てるメモリなどのリソースの量を指定できます。

DB インスタンスをパラメータグループに関連付けることで、データベース設定を管理します。Amazon Timestream for InfluxDB は、デフォルト設定でパラメータグループを定義します。カスタマイズした設定を使用して独自のパラメータグループを定義できます。

パラメータグループの概要

DB パラメータグループは、1 つ以上の DB インスタンスに適用されるエンジン設定値のコンテナとして機能します。

トピック

- デフォルトおよびカスタムパラメータグループ
- DB パラメータグループを作成する
- 静的および動的 DB インスタンスパラメータ
- サポートされるパラメータとパラメータ値

デフォルトおよびカスタムパラメータグループ

DB インスタンスでは DB パラメータグループが使用されます。次のセクションでは、DB インスタンスパラメータグループの設定と管理について説明します。

DB パラメータグループを作成する

AWS Management Console、、 AWS Command Line Interfaceまたは Timestream を使用して、新しい DB パラメータグループを作成できますAPI。

DB パラメータグループ名には、次の制限事項が適用されます。

- 名前は、1～255 の英字、数字、ハイフンである必要があります。
- デフォルトのパラメータグループ名には、`default.InfluxDB.2.7` のようなピリオドを含めることができます。ただし、カスタムパラメータグループ名にはピリオドを含めることはできません。
- 1 字目は文字である必要があります。

- 名前は「dbpg-」で始まることはできません
- 名前の最後にハイフンを使用したり、ハイフンを 2 つ続けて使用したりすることはできません。
- DB パラメータグループを指定せずに DB インスタンスを作成する場合、DB インスタンスは influxDB エンジンのデフォルトを使用します。

デフォルトのパラメータグループのパラメータ設定は変更できません。代わりに、以下を実行できます。

1. 新しいパラメータグループを作成します。
2. 必要なパラメータの設定を変更します。パラメータグループ内のすべての DB エンジンパラメータが変更できるわけではありません。
3. DB インスタンスを更新して、カスタムパラメータグループを使用します。DB インスタンスの更新については、「」を参照してくださいDB インスタンスの更新。

> **Note**
>
> カスタムパラメータグループを使用するように DB インスタンスを変更し、DB インスタンスを起動した場合、Amazon Timestream for InfluxDB は起動プロセスの一環として DB インスタンスを自動的に再起動します。
> 現在、カスタムパラメータグループの作成後は変更できません。パラメータを変更する必要がある場合は、新しいカスタムパラメータグループを作成し、この設定の変更を必要とするインスタンスに割り当てる必要があります。既存の DB インスタンスを更新して新しいパラメータグループを割り当てると、常にすぐに適用され、インスタンスを再起動します。

静的および動的 DB インスタンスパラメータ

InfluxDB DB インスタンスパラメータは常に静的 です。これらは次のように動作します。

静的パラメータを変更し、DB パラメータグループを保存してインスタンスに割り当てると、インスタンスの再起動後にパラメータの変更が自動的に有効になります。

新しい DB パラメータグループを DB インスタンスに関連付けると、Timestream は DB インスタンスが再起動された後にのみ、変更された静的パラメータを適用します。現在、唯一のオプションがすぐに適用されます。

DB パラメータグループの変更については、「DB インスタンスの更新」を参照してください。

サポートされるパラメータとパラメータ値

DB インスタンスでサポートされているパラメータを確認するには、DB インスタンスで使用される DB パラメータグループのパラメータを表示します。詳細については、「DB パラメータグループのパラメータ値の表示」を参照してください。

InfluxDB のオープンソースバージョン でサポートされているすべてのパラメータの詳細については、「InfluxDB 設定オプション」を参照してください。現在、変更できるのは次の InfluxDB パラメータのみです。

パラメータ	説明	デフォルト値	値	有効範囲	注記
flux-log-enabled	Flux クエリの詳細ログを表示するオプションを含める	FALSE	true、false	該当なし	
ログレベル	ログ出力レベル。InfluxDB は、重要度レベルが指定されたレベル以上のログエントリを出力します。	info	デバッグ、情報、エラー	該当なし	
タスクなし	同時に実行できるクエリの数。0 に設定すると、同時クエリの数に制限はありません。	FALSE	true、false	該当なし	
クエリの同時実行	タスクスケジューラを無効にしま	1024		該当なし	

パラメータ	説明	デフォルト値	値	有効範囲	注記
	す。問題のあるタスクで InfluxDB が起動できない場合は、このオプションを使用して、タスクをスケジュールまたは実行せずに InfluxDB を起動します。				
query-queue-size	実行キューで許可されるクエリの最大数。キューの制限に達すると、新しいクエリは拒否されます。0 に設定すると、キュー内のクエリの数に制限はありません。	1024		該当なし	

パラメータ	説明	デフォルト値	値	有効範囲	注記
トレースタイプ	InfluxDB でトレースを有効にし、トレースタイプを指定します。トレースはデフォルトで無効になっています。	""	ログ、jaeger	該当なし	
メトリクスが無効	内部 InfluxDB メトリクスを公開する HTTP /metrics エンドポイントを無効にします。InfluxDB	FALSE		該当なし	

パラメータ	説明	デフォルト値	値	有効範囲	注記
http-idle-timeout	新しいリクエストを待つ間、サーバーが確立された接続を存続させる必要がある最大期間。タイムアウトなしで 0 をに設定します。	3m0s	単位 、hours、mseconds の持続時間milliseconds 。例: durationType=minutes,value=10	時間： -最小: 0 -最大: 256205 分： -最小: 0 -最大: 15372286 秒： -最小: 0 -最大: 922337203 ミリ秒： -最小: 0 -最大: 922337203685	

パラメータ	説明	デフォルト値	値	有効範囲	注記
http-read-header-timeout	サーバーが新しいリクエストのHTTPヘッダーを読み込もうとする必要がある最大期間。タイムアウトしないように 0 を に設定します。	10 秒	単位 、hours、mseconds の持続時間milliseconds 。例: durationType=minutes,value=10	時間： -最小: 0 -最大: 256205 分： -最小: 0 -最大: 15372286 秒： -最小: 0 -最大: 922337203 ミリ秒： -最小: 0 -最大: 922337203685	

パラメータ	説明	デフォルト値	値	有効範囲	注記
http-read-timeout	サーバーが新しいリクエスト全体を読み込もうとする必要がある最大期間。タイムアウトしないように 0 をに設定します。	0	単位 、hours、mseconds の持続時間milliseconds 。例: durationType=minutes,value=10	時間： -最小: 0 -最大: 256205 分： -最小: 0 -最大: 15372286 秒： -最小: 0 -最大: 922337203 ミリ秒： -最小: 0 -最大: 9223372036685	

パラメータ	説明	デフォルト値	値	有効範囲	注記
http-write-timeout	サーバーが書き込みリクエストの処理と応答に費やす最大時間。タイムアウトしないように 0 を に設定します。	0	単位 、hours、mseconds の持続時間milliseconds 。例: durationType=minutes,value=10	時間： -最小: 0 -最大: 256205 分： -最小: 0 -最大: 15372286 秒： -最小: 0 -最大: 922337203 ミリ秒： -最小: 0 -最大: 922337203685	

パラメータ	説明	デフォルト値	値	有効範囲	注記
influxql-max-select-buckets	SELECT ステートメントが作成できる時間バケットごとのグループの最大数。 0では、バケットの数に制限はありません。	0	Long	最小: 0 最大: 9,223,372,036,854,775,807	
influxql-max-select-point	SELECT ステートメントが処理できるポイントの最大数。 0では、ポイントの数に制限はありません。InfluxDB は、ポイントカウントを 1 秒ごとにチェックします (最大数を超えるクエリはすぐに中止されません)。	0	Long	最小: 0 最大: 9,223,372,036,854,775,807	

パラメータ	説明	デフォルト値	値	有効範囲	注記
influxql-max-select-series	SELECT ステートメントが返すことができるシリーズの最大数。 0では、シリーズの数に制限はありません。	0	Long	最小: 0 最大: 9,223,372,036,854,775,807	
pprof が無効になりました	/debug/pprof HTTP エンドポイントを無効にします。このエンドポイントはランタイムプロファイリングデータを提供し、デバッグに役立ちます。	FALSE	ブール値	該当なし	
query-initial-memory-bytes	クエリに割り当てられたメモリの初期バイト数。	0	Long	最小: 0 最大: query-memory-bytes	
query-max-memory-bytes	クエリに使用できるメモリの最大合計バイト数。	0	Long	最小: 0 最大: 9,223,372,036,854,775,807	

パラメータ	説明	デフォルト値	値	有効範囲	注記
query-memory-bytes	新しく作成されたユーザーセッションの生存時間 (TTL) を分単位で指定します。	0	Long	最小: 0 最大: 2,147,483,647	以上である必要があります query-initial-memory-bytes。
セッションの長さ	新しく作成されたユーザーセッションの生存時間 (TTL) を分単位で指定します。	60	整数	最小: 0 最大: 2880	

パラメータ	説明	デフォルト値	値	有効範囲	注記
session-renew-disabled	各リクエストTTLでユーザーのセッションを自動的に延長することを無効にします。デフォルトでは、すべてのリクエストはセッションの有効期限を現在から 5 分後に設定します。無効にすると、セッションは指定されたセッション長の後に期限切れになり、最近アクティブになってもユーザーはログインページにリダイレクトされます。	FALSE	ブール値	該当なし	

パラメータ	説明	デフォルト値	値	有効範囲	注記
storage-cache-max-memory- サイズ	シャードのキャッシュが書き込みの拒否を開始する前に到達できる最大サイズ (バイト単位) 。	1073741824	Long	最小: 0 最大数: 549755813888	インスタンスの合計メモリ容量よりも小さくする必要があります。 合計メモリ容量の 15% 未満に設定することをお勧めします。
storage-cache-snapshot-memory-サイズ	ストレージエンジンがキャッシュのスナップショットを作成し、TSM ファイルへの書き込みを行ってメモリを使用可能にするサイズ (バイト単位) 。	26214400	Long	最小: 0 最大数: 549755813888	storage-cache-max-memoryサイズは より小さくする必要があります。

パラメータ	説明	デフォルト値	値	有効範囲	注記
storage-cache-snapshot-write-コールドデュレーション	シャードが書き込みまたは削除を受信していない場合に、ストレージエンジンがキャッシュをスナップショットし、新しいTSMファイルに書き込む期間。	10m0s	単位 、hours、mseconds の持続時間milliseconds 。例: durationType=minutes,value=10	時間： -最小: 0 -最大: 256205 分： -最小: 0 -最大: 15372286 秒： -最小: 0 -最大: 922337203 ミリ秒： -最小: 0 -最大: 922337203685	

パラメータ	説明	デフォルト値	値	有効範囲	注記
storage-compact-full-write-コールドデュレーション	書き込みや削除を受信していない場合に、ストレージエンジンがシャード内のすべてのTSMファイルを圧縮する期間。	4h0m0s	単位、hours、mseconds の持続時間milliseconds 。例: durationType=minutes,value=10	時間： -最小: 0 -最大: 256205 分： -最小: 0 -最大: 15372286 秒： -最小: 0 -最大: 922337203 ミリ秒： -最小: 0 -最大: 9223372036685	
storage-compact-throughput-burst	TSM 圧縮がディスクに書き込むことができるレート制限 (バイト/秒) 。	50331648	Long	最小: 0 最大: 9,223,372,036,854,775,807	

パラメータ	説明	デフォルト値	値	有効範囲	注記
storage-max-concurrent-compactions	同時に実行できるフル圧縮とレベル圧縮の最大数。の値は、ランタイムruntime.GOMAXPROCS(0) で使用される の 50% 0になります。0 より大きい数値は、圧縮をその値に制限します。この設定は、キャッシュスナップショット作成には適用されません。	0	整数	最小: 0 最大: 64	

パラメータ	説明	デフォルト値	値	有効範囲	注記
storage-max-index-log-file-size	インデックス先行書き込みログ () ファイルがインデックスファイルに圧縮されるサイズ (バイト単位WAL)。サイズを小さくすると、ログファイルの圧縮速度が向上し、書き込みスループットを犠牲にしてヒープ使用量が減少します。	1048576	Long	最小: 0 最大: 9,223,372,036,854,775,807	
storage-no-validate-field-サイズ	受信書き込みリクエストのフィールドサイズ検証をスキップします。	FALSE	ブール値	該当なし	

パラメータ	説明	デフォルト値	値	有効範囲	注記
storage-retention-check-interval	保持ポリシーの適用チェックの間隔。	30m0s	単位 、hours、mseconds の持続時間milliseconds 。例: durationType=minutes,value=10	該当なし	時間： -最小: 0 -最大: 256205 分： -最小: 0 -最大: 15372286 秒： -最小: 0 -最大: 922337203 ミリ秒： -最小: 0 -最大: 9223372036685
storage-series-file-max-concurrent-snapshot-compactions	データベース内のすべてのシリーズパーティションで同時に実行できるスナップショット圧縮の最大数。	0	整数	最小: 0 最大: 64	

パラメータ	説明	デフォルト値	値	有効範囲	注記
storage-series-id-set-キャッシュサイズ	以前に計算されたシリーズ結果を保存するためにTSIインデックスで使用される内部キャッシュのサイズ。キャッシュされた結果は、同じタグキー/値の述語を持つ後続のクエリが実行されたときに再計算する必要がなくなることなく、迅速に返されます。この値を に設定するとキャッシュが無効0になり、クエリのパフォーマンスが低下する可能性があります。	100	Long	最小: 0 最大: 9,223,372,036,854,775,807	

パラメータ	説明	デフォルト値	値	有効範囲	注記
storage-wal-max-concurrent書き込み	同時に試行するWALディレクトリへの最大書き込み数。	0	整数	最小: 0 最大: 256	
storage-wal-max-write- 遅延	WAL ディレクトリへの書き込みリクエストが、WALディレクトリへの同時アクティブ書き込みの最大数に達したときに待機する最大時間。タイムアウト0を無効にするには、 を に設定します。	10m	単位 、hours、mseconds の持続時間milliseconds 。例: durationType=minutes,value=10	時間： -最小: 0 -最大: 256205 分： -最小: 0 -最大: 15372286 秒： -最小: 0 -最大: 922337203 ミリ秒： -最小: 0 -最大: 922337203685	

パラメータ	説明	デフォルト値	値	有効範囲	注記
ui が無効	InfluxDB ユーザーインターフェイス (UI) を無効にします。UI はデフォルトで有効になっています。	FALSE	ブール値	該当なし	

パラメータグループに不適切な設定のパラメータがあると、パフォーマンスが低下したりシステムが不安定になったり、予期しない悪影響が生じることがあります。データベースパラメータを変更するときは、常に注意してください。テスト DB インスタンスでパラメータグループ設定の変更を試してから、これらのパラメータグループの変更を本番 DB インスタンスに適用します。

DB パラメータグループを使用する

DB インスタンスでは DB パラメータグループが使用されます。次のセクションでは、DB インスタンスパラメータグループの設定と管理について説明します。

トピック

- DB パラメータグループを作成する
- DB パラメータグループを DB インスタンスに関連付ける
- DB パラメータグループを一覧表示する
- DB パラメータグループのパラメータ値を表示する

DB パラメータグループを作成する

の使用 AWS Management Console

1. にサインイン AWS Management Console し、Amazon Timestream for InfluxDB コンソール を開きます。
2. ナビゲーションペインで、[パラメータグループ] を選択します。
3. [パラメータグループの作成] を選択します。
4. [グループ名] ボックスに、新しい DB パラメータグループの名前を入力します。

5. [説明] ボックスに、新しい DB パラメータグループの説明を入力します。
6. パラメータを選択して変更し、目的の値を適用します。サポートされているパラメータの詳細については、「」を参照してくださいサポートされるパラメータとパラメータ値。
7. [Save] を選択します。

の使用 AWS Command Line Interface

- を使用して DB パラメータグループを作成するには AWS CLI、次のパラメータを使用して `create-db-parameter-group` コマンドを呼び出します。

```
--db-parameter-group-name <value>
--description <value>
--endpoint_url <value>
--region <value>
--parameters (list) (string)
```

Example 例

各設定の詳細については、「DB インスタンスの設定」を参照してください。この例では、デフォルトのエンジン設定を使用します。

```
aws timestream-influxdb create-db-parameter-group
    --db-parameter-group-name YOUR_PARAM_GROUP_NAME\
    --endpoint-url YOUR_ENDPOINT
    --region YOUR_REGION \
    --parameters
 "InfluxDBv2={logLevel=debug,queryConcurrency=10,metricsDisabled=true}" \" \
    --debug
```

DB パラメータグループを DB インスタンスに関連付ける

カスタマイズした設定を使用して、独自の DB パラメータグループを作成できます。DB パラメータグループは、 AWS Management Console、、 AWS Command Line Interfaceまたは Timestream-InfluxDB を使用して DB インスタンスに関連付けることができますAPI。DB インスタンスを作成または変更するときに、これを行うことができます。

DB パラメータグループの作成については、DB パラメータグループを作成する を参照してください。DB インスタンスの作成については、「DB インスタンスの作成」を参照してください。DB インスタンスの変更については、「」を参照してくださいDB インスタンスの更新。

> **Note**
>
> 新しい DB パラメータグループを DB インスタンスに関連付けると、変更された静的パラメータは DB インスタンスの再起動後にのみ適用されます。現在、即時適用のみがサポートされています。InfluxDB の Timestream は静的パラメータのみをサポートします。

の使用 AWS Management Console

1. にサインイン AWS Management Console し、Amazon Timestream for InfluxDB コンソール を開きます。
2. ナビゲーションペインで、InfluxDB データベース を選択し、変更する DB インスタンスを選択します。
3. [Update] (更新) を選択します。DB インスタンスの更新ページが表示されます。
4. DB パラメータグループの設定を変更します。
5. [Continue] を選択して、変更の概要を確認します。
6. 現在、すぐに適用のみがサポートされています。このオプションは DB インスタンスを再起動するため、場合によっては停止する可能性があります。
7. 確認ページで、変更内容を確認します。正しい場合は、DB インスタンスの更新を選択して変更を保存し、適用します。または、[戻る] を選択して変更を編集するか、キャンセルを選択して変更をキャンセルします。

の使用 AWS Command Line Interface

Linux、macOS、Unix の場合:

```
aws timestream-influxdb update-db-instance
--identifier YOUR_DB_INSTANCE_ID \
--region YOUR_REGION \
--db-parameter-group-identifier YOUR_PARAM_GROUP_ID \
--log-delivery-configuration "{\"s3Configuration\": {\"bucketName\":
 \"${LOGGING_BUCKET}\", \"enabled\": false }}"
```

Windows の場合:

```
aws timestream-influxdb update-db-instance
--identifier YOUR_DB_INSTANCE_ID ^
--region YOUR_REGION ^
--db-parameter-group-identifier YOUR_PARAM_GROUP_ID ^
--log-delivery-configuration "{\"s3Configuration\": {\"bucketName\":
 \"${LOGGING_BUCKET}\", \"enabled\": false }}"
```

DB パラメータグループを一覧表示する

AWS アカウント用に作成した DB パラメータグループを一覧表示できます。

の使用 AWS Management Console

1. にサインイン AWS Management Console し、Amazon Timestream for InfluxDB コンソール を開きます。
2. ナビゲーションペインで、[パラメータグループ] を選択します。
3. DB パラメータグループがリストに表示されます。

の使用 AWS Command Line Interface

AWS アカウントのすべての DB パラメータグループを一覧表示するには AWS Command Line Interface 、 `list-db-parameter-groups` コマンドを使用します。

```
aws timestream-influxdb list-db-parameter-groups --region region
```

AWS アカウントの特定の DB パラメータグループを返す AWS Command Line Interface には、`get-db-parameter-group` コマンドを使用します。

```
aws timestream-influxdb get-db-parameter-group --region region --identifier identifier
```

DB パラメータグループのパラメータ値を表示する

DB パラメータグループのすべてのパラメータとそれらの値のリストを取得できます。

の使用 AWS Management Console

1. にサインイン AWS Management Console し、Amazon Timestream for InfluxDB コンソール を開きます。
2. ナビゲーションペインで、[パラメータグループ] を選択します。
3. DB パラメータグループがリストに表示されます。
4. パラメータを一覧表示するパラメータグループの名前を選択します。

の使用 AWS Command Line Interface

DB パラメータグループのパラメータ値を表示するには、次の必須パラメータで コマンドを使用します AWS Command Line Interface `get-db-parameters`。

```
--db-parameter-group-name
```

の使用 API

DB パラメータグループのパラメータ値を表示するには、次の必須パラメータで Timestream API `GetDBParameters` コマンドを使用します。

```
DBParameterGroupName
```

DB インスタンスの管理

このセクションでは、最適なパフォーマンス、可用性、モニタリング機能を確保するために、Timestream for InfluxDB インスタンスを管理するさまざまな側面について説明します。データベースインスタンスの設定の更新、マルチ AZ デプロイの処理、フェイルオーバープロセスに関するガイダンスを提供します。また、データベースインスタンスを削除し、InfluxDB インスタンスのログ表示を設定する方法について説明します。

トピック

- DB インスタンスの更新
- DB インスタンスのメンテナンス
- DB インスタンスを削除する
- マルチAZ DB インスタンスのデプロイ
- Timestream InfluxDB インスタンスで InfluxDB ログを表示するセットアップ

DB インスタンスの更新

Timestream for InfluxDB インスタンスの次の設定パラメータを更新できます。

- インスタンスクラス
- デプロイタイプ
- パラメータグループ
- ログ配信設定

> **Important**
>
> 特にデータベースバージョンをアップグレードする場合は、本番稼働用インスタンスを変更する前に、テストインスタンスですべての変更をテストすることをお勧めします。設定を更新する前に、データベースとアプリケーションへの影響を確認してください。一部の変更では、DB インスタンスの再起動が必要になるため、ダウンタイムが発生します。

の使用 AWS Management Console

1. にサインイン AWS Management Console し、Amazon Timestream for InfluxDB コンソール を開きます。
2. ナビゲーションペインで、InfluxDB データベース を選択し、変更する DB インスタンスを選択します。
3. Modify を選択します。
4. DB インスタンスの変更ページで、必要な変更を行います。
5. [続行] を選択して、変更の概要を確認します。
6. [Next (次へ)] を選択します。
7. 変更を確認します。
8. インスタンスの変更を選択して変更を適用します。

> **Note**
>
> これらの変更には Influx DB インスタンスの再起動が必要であり、場合によっては停止を引き起こす可能性があります。

の使用 AWS Command Line Interface

を使用して DB インスタンスを更新するには AWS Command Line Interface、 `update-db-instance` コマンドを呼び出します。DB インスタンス識別子と、変更するオプションの値を指定します。各オプションの詳細については、「DB インスタンスの設定」を参照してください。

Example 例

次のコードは、別の dbparametergroup を設定することで mydbinstance を変更します。変更はすぐに適用されます。

Linux、macOS、Unix の場合:

```
aws timestream-influxdb update-db-instance \
    --identifier mydbinstance \
    --db-instance-type desired-instance-type \
    --deployment-type desired-deployment-type \
    --db-parameter-group-name newparamgroup \
    --port 8086
```

Windows の場合:

```
aws timestream-influxdb update-db-instance ^
    --identifier mydbinstance ^
    --db-instance-type desired-instance-type ^
    --deployment-type desired-deployment-type ^
    --db-parameter-group-name newparamgroup
    --port 8086
```

DB インスタンスのメンテナンス

Amazon Timestream for InfluxDB は、定期的に Amazon Timestream for InfluxDB リソースのメンテナンスを実行します。メンテナンスでは、DB インスタンス内の次のリソースの更新が頻繁に行われます。

- 基盤となるハードウェア
- 基盤となるオペレーティングシステム (OS)
- データベースエンジンのバージョン

通常、オペレーティングシステムのアップデートはセキュリティ問題に関連しています。

一部のメンテナンス項目では、Amazon Timestream for InfluxDB が DB インスタンスを短時間オフラインにする必要があります。リソースをオフラインにする必要があるメンテナンス項目には、必要なオペレーティングシステムやデータベースのパッチが含まれます。セキュリティやインスタンスの信頼性に関連するパッチのみ、必須のパッチ適用として自動的にスケジューリングされます。そのようなパッチ適用はまれであり、通常は数か月に一度です。メンテナンスウィンドウのごくわずかしか必要としないことがほとんどです。

- メンテナンスウィンドウは、インスタンスがホストされているリージョンの現地時間の午前 12 時から午前 4 時の間に毎日行われるように設定されています。
- カスタマーリソースには、その週の 7 つのメンテナンスウィンドウのいずれかで、週に 1 回パッチが適用される場合があります。

DB インスタンスを削除する

DB インスタンスを削除すると、インスタンスの復元可能性とスナップショットの可用性に影響します。以下の問題について考えます。

- InfluxDB リソースのすべての Timestream を削除する場合は、DB インスタンスリソースに請求料金が発生することに注意してください。
- DB インスタンスのステータスを削除すると、その CA 証明書の値は、InfluxDB コンソールの Timestream または AWS Command Line Interface コマンドまたは Timestream APIオペレーションの出力に表示されません。
- DB インスタンスの削除に必要な時間は、削除されるデータの量と、最終的なスナップショットが取得されるかどうかによって異なります。

DB インスタンスは、 AWS Management Console、、 AWS Command Line Interfaceまたは Timestream を使用して削除できますAPI。DB インスタンスの名前を指定する必要があります。

の使用 AWS Management Console

1. にサインイン AWS Management Console し、Amazon Timestream for InfluxDB コンソール を開きます。
2. ナビゲーションペインで、InfluxDB データベース を選択し、削除する DB インスタンスを選択します。

3. [削除] を選択します。
4. ボックスに確認と入力します。
5. [削除] を選択します。

の使用 AWS Command Line Interface

アカウント内の DB インスタンスIDsのインスタンスを検索するには、 `list-db-instances` コマンドを呼び出します。

```
aws timestream-influxdb list-db-instances \
--endpoint-url YOUR_ENDPOINT \
--region YOUR_REGION
```

を使用して DB インスタンスを削除するにはAWSCLI、次のオプションを使用して `delete-db-instance` コマンドを呼び出します。

```
aws timestream-influxdb list-db-instances \
--identifier YOUR_DB_INSTANCE \
```

Example 例

Linux、macOS、Unix の場合:

```
aws timestream-influxdb delete-db-instance \
    --identifier mydbinstance
```

Windows の場合:

```
aws timestream-influxdb delete-db-instance ^
    --identifier mydbinstance
```

マルチAZ DB インスタンスのデプロイ

Amazon Timestream for InfluxDB は、単一のスタンバイ DB インスタンスでマルチ AZ デプロイを使用して、DB インスタンスの高可用性とフェイルオーバーのサポートを提供します。このタイプのデプロイは、マルチ AZ DB インスタンスのデプロイと呼ばれます。Amazon Timestream for InfluxDB は、Amazon フェイルオーバーテクノロジーを使用します。

マルチ AZ DB インスタンスのデプロイでは、Amazon Timestream は同期スタンバイレプリカを別のアベイラビリティーゾーンに自動的にプロビジョニングして維持します。プライマリ DB インスタンスは、同期的にアベイラビリティーゾーン間でスタンバイレプリカにレプリケートされ、データの冗長性が提供されます。高可用性で DB インスタンスを実行すると、DB インスタンスの障害やアベイラビリティーゾーンの中断時に可用性を高めることができます。アベイラビリティーゾーンの詳細については、「AWS リージョンとアベイラビリティーゾーン」を参照してください。

> **Note**
>
> 高可用性のオプションは、読み取り専用シナリオ向けのスケーリングソリューションではありません。スタンバイレプリカを使用してリードトラフィックを処理することはできません。

Amazon Timestream コンソールを使用して、DB インスタンスの作成時に可用性と耐久性の設定セクションでスタンバイインスタンスの作成オプションを指定するだけで、マルチ AZ DB インスタンスのデプロイを作成できます。 AWS Command Line Interface または Amazon Timestream を使用してマルチ AZ DB インスタンスのデプロイを指定することもできますAPI。`create-db-instance` または CLI コマンド、または `CreateDBInstance`APIオペレーションを使用します。

マルチAZ DB デプロイを使用する DBインスタンスは、シングル AZ のデプロイと比較して書き込みとコミットの待ち時間が長くなる可能性があります。これは、同期データレプリケーションが発生しているために起こる可能性があります。デプロイがスタンバイレプリカにフェイルオーバーすると、レイテンシーが変わる可能性がありますが、 AWS はアベイラビリティーゾーン間の低レイテンシーネットワーク接続で設計されています。本稼働ワークロードでは、高速で一貫したパフォーマンスIOPSを実現するために、IOPS付属のストレージ 12K または 16K を使用することをお勧めします。DB インスタンスクラスの詳細については、「DB インスタンスクラス」を参照してください。

マルチ AZ デプロイの設定と管理

InfluxDB マルチ AZ デプロイのタイムストリームには、スタンバイを 1 つだけ設定できます。デプロイにスタンバイ DB インスタンスが 1 つある場合は、マルチ AZ DB インスタンスのデプロイと呼ばれます。マルチ AZ DB インスタンスのデプロイには、フェイルオーバーサポートを提供するスタンバイ DB インスタンスが 1 つありますが、読み取りトラフィックは処理されません。

> **Important**
>
> シングル AZ からマルチ AZ への更新を実行するには、インスタンスに少なくとも 2 つのサブネットが関連付けられている必要があります。インスタンスが作成されると、デプロイモードをシングル AZ からマルチ AZ に変更することはできません。

を使用して AWS Management Console 、DB インスタンスがシングル AZ デプロイかマルチ AZ デプロイかを決定できます。

の使用 AWS Management Console

1. にサインイン AWS Management Console し、Amazon Timestream for InfluxDB コンソール を開きます。
2. ナビゲーションペインで、InfluxDB データベース を選択し、DB 識別子 を選択します。

マルチ AZ DB インスタンス配置には、次の特性があります。

- DB インスタンスの行は 1 行のみ。
- [Role] (ロール) の値は [Instance] (インスタンス) または [Primary] (プライマリ)。
- [Multi-AZ] (マルチ AZ) の値は [Yes] (はい)。

Amazon Timestream のフェイルオーバープロセス

DB インスタンスの計画的または計画外の停止がインフラストラクチャの欠陥に起因する場合、マルチ AZ をオンにすると、Amazon Timestream for InfluxDB は別のアベイラビリティーゾーンのスタンバイレプリカに自動的に切り替わります。フェイルオーバーが完了するまでにかかる時間は、プライマリ DB インスタンスが使用できなくなったときのデータベースアクティビティおよびその他の条件によって異なります。フェイルオーバー時間は通常 60 ~ 120 秒です。ただし、大規模なトランザクションや長期にわたる復旧プロセスによって、フェイルオーバー時間が増加する場合があります。フェイルオーバーが完了すると、Timestream コンソールが新しいアベイラビリティーゾーンを反映するまでにさらに時間がかかる場合があります。

> **Note**
>
> Amazon Timestream はフェイルオーバーを自動的に処理するため、管理上の介入なしでデータベースオペレーションをできるだけ早く再開できます。次の表に記載されている条件のい

ずれかが発生した場合、プライマリ DB インスタンスがスタンバイレプリカに自動的に切り替わります。

フェイルオーバーした理由	説明
Timestream データベースインスタンスの基盤となるオペレーティングシステムは、オフラインオペレーションでパッチ適用されています。	OS パッチまたはセキュリティ更新プログラムのメンテナンス期間中に、フェイルオーバーがトリガーされました。
Timestream マルチ AZ インスタンスのプライマリホストが異常です。	マルチ AZ DB インスタンスのデプロイは、障害のあるプライマリ DB インスタンスを検出し、フェイルオーバーしました。
Timestream マルチ AZ インスタンスのプライマリホストは、ネットワーク接続が失われたため到達できません。	Timestream モニタリングは、プライマリ DB インスタンスへのネットワーク到達可能性の障害を検出し、フェイルオーバーをトリガーしました。
Timestream インスタンスはお客様によって変更されました。	Timesteam for InfluxDB DB インスタンスの変更によりフェイルオーバーがトリガーされました。詳細については、「DB インスタンスの更新」を参照してください。
Timestream マルチ AZ プライマリインスタンスがビジーで応答しません。	プライマリ DB インスタンスが応答しません。次の操作を行うことをお勧めします。* イベントを調べてCPU、 、メモリ、またはスワップスペースの使用量が過剰かどうかを確認してください。* ワークロードを評価して、適切な DB インスタンスクラスを使用しているかどうかを判断します。詳細については、「DB インスタンスクラス」を参照してください。
Timestream マルチ AZ インスタンスのプライマリホストの下にあるストレージボリュームで障害が発生しました。	マルチ AZ DB インスタンスのデプロイは、プライマリ DB インスタンスでストレージの問題を検出し、フェイルオーバーしました。

DNS 名前ルックアップJVMTTLの の設定

フェイルオーバーメカニズムは、スタンバイ DB インスタンスを指すように DB インスタンスのドメインネームシステム (DNS) レコードを自動的に変更します。したがって、DB インスタンスへの既存の接続の再確立が必要になります。Java 仮想マシン (JVM) 環境では、Java DNS キャッシュメカニズムの仕組みにより、JVM設定を再設定する必要がある場合があります。

JVM キャッシュDNS名ルックアップ。がホスト名を IP アドレスにJVM解決すると、 time-to-live () と呼ばれる指定された期間、IP アドレスがキャッシュされますTTL。

AWS リソースは、時折変更されるDNS名前エントリを使用するため、60 秒以下のTTL値JVMで を設定することをお勧めします。これにより、リソースの IP アドレスが変更されると、アプリケーションは を再クエリすることで、リソースの新しい IP アドレスを受信して使用できますDNS。

一部の Java 設定では、 JVMが再起動されるまでDNSエントリが更新されないようにJVMデフォルトTTLが設定されます。したがって、アプリケーションの実行中に AWS リソースの IP アドレスが変更された場合、 を手動で再起動JVMし、キャッシュされた IP 情報が更新されるまで、そのリソースを使用することはできません。この場合、キャッシュされた IP 情報を定期的に更新TTLするように JVMを設定することが重要です。

`networkaddress.cache.ttl` プロパティ値を取得TTLすることで、JVMデフォルトを取得できます。

```
String ttl = java.security.Security.getProperty("networkaddress.cache.ttl");
```

> **Note**
>
> デフォルトTTLは、 のバージョンJVMとセキュリティマネージャーがインストールされているかどうかによって異なります。多くの場合、デフォルトは 60 秒TTL未満JVMsです。このような を使用していてJVM、セキュリティマネージャーを使用していない場合は、このトピックの残りの部分を無視できます。
> JVMの を変更するにはTTL、networkaddress.cache.ttl プロパティ値を設定します。ニーズに応じて、次の方法のいずれかを使用します。
>
> - を使用するすべてのアプリケーションにプロパティ値をグローバルに設定するにはJVM、 `$JAVA_HOME/jre/lib/security/java.security` ファイルnetworkaddress.cache.ttlで を設定します。
>
>   ```
>   networkaddress.cache.ttl=60
>   ```

- アプリケーションに対してのみプロパティをローカルに設定するには、ネットワーク接続を確立する前に、アプリケーションの初期化コードで networkaddress.cache.ttl を設定します。

```
java.security.Security.setProperty("networkaddress.cache.ttl" , "60");
```

Timestream InfluxDB インスタンスで InfluxDB ログを表示するセットアップ

デフォルトではInfluxDB は stdout に移行するログを生成します。詳細については、「InfluxDB ログの管理」を参照してください。

Timestream InfluxDB を介して作成したインスタンスから生成された InfluxDB ログを表示するには、時間単位のログを提供する機会を提供します。これらのログは、インスタンスを作成する前に作成する必要がある指定された S3 バケットに送信されます。

- インスタンスを作成する前に、提供された Amazon S3 バケットは、Timestream InfluxDB Service Principal を次のようにバケットポリシーに提供して、このバケットにログを送信するアクセス許可を Timestream-InfluxDB に付与する必要があります (置き換える *{BUCKET_NAME}* Amazon S3 バケットの実際の名前：

```
{
    "Version": "2012-10-17",
    "Id": "PolicyForInfluxLogs",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "timestream-influxdb.amazonaws.com"
            },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::{BUCKET_NAME}/InfluxLogs/*"
        }
    ]
}
```

- 指定されたバケットは、作成した Timestream InfluxDB インスタンスの同じアカウントと同じリージョンにある必要があります。

以下は、インスタンスを作成して流入ログを受信するために呼び出すことができるコマンドです。

```
aws timestream-influxdb create-db-instance \
    --name myinfluxDbinstance \
    --allocated-storage 400 \
    --db-instance-type db.influx.4xlarge \
    --vpc-subnet-ids subnetid1 subnetid2
    --vpc-security-group-ids mysecuritygroup \
    --username masterawsuser \
    --password \
    --db-storage-type InfluxIOIncludedT2
```

このパラメータの形式は次のとおりです。

```
-- log-delivery-configuration
{
    "S3Configuration": {
      "BucketName": "string",
      "Enabled": true|false
    }
}
```

- このフィールドは必須ではなく、ログ記録はデフォルトでは有効になっていません。
- このフィールドを設定しないことは、ログを有効にしないことと同じです。
- ログは、プレフィックスが の指定されたバケットに送信されますInfluxLogs/。
- インスタンスを作成したら、 update-db-instance API コマンドを使用してログ配信設定を変更できます。

InfluxDB にはさまざまなタイプのログが用意されています。これらは、InfluxDB パラメータを設定することで設定できます。および flux-log-enabledログレベルのパラメータを使用して、インスタンスから出力されるログのタイプを設定します。詳細については、「サポートされるパラメータとパラメータ値」を参照してください。

リソースへのタグとラベルの追加

タグ を使用して、Amazon Timestream for InfluxDB リソースにラベルを付けることができます。タグを使用すると、目的別、所有者別、環境別など、さまざまな方法でリソースを分類できます。タグは、以下のことに役立ちます。

- リソースに割り当てたタグに基づいてリソースをすばやく特定します。
- タグ別に分類された AWS 請求書を参照してください。

タグ付けは、Amazon Elastic Compute Cloud (Amazon EC2) 、Amazon Simple Storage Service (Amazon S3)、Timestream for InfluxDB などの AWS サービスでサポートされています。効率的なタグ付けを行うと、特定のタグを持つサービス間でレポートを作成でき、コストインサイトを得ることができます。

最後に、最適のタグ付け手法に従うことをお勧めします。詳細については、「AWS タグ付け戦略」を参照してください。

タグ指定の制限

タグはそれぞれ、1 つのキーと 1 つの値で構成されており、どちらもお客様側が定義します。以下の制限が適用されます。

- InfluxDB DB インスタンスの Timestream ごとに、同じキーを持つタグを 1 つだけ持つことができます。既存のタグを追加しようとすると、既存のタグ値が新しい値に更新されます。
- 値は、タグカテゴリ内の記述子として機能します。InfluxDB の Timestream では、値を空または null にすることはできません。
- タグのキーと値では、大文字と小文字が区別されます。
- キーの最大長は Unicode 文字で 128 文字です。
- 値の最大長は Unicode 文字で 256 文字です。
- 使用できる文字は、文字、ホワイトスペース、数字、および特殊文字 (+ - = . _ : /) です。
- リソースあたりのタグの最大数は 50 です。
- AWS- 割り当てられたタグ名と値には、割り当てることができない`aws:`プレフィックスが自動的に割り当てられます。 AWS- 割り当てられたタグ名は、タグ制限の 50 にはカウントされません。ユーザー側で割り当てたタグ名は、`user:` というプレフィックスを付けてコスト配分レポートに表示されます。

- 過去にさかのぼってタグを適用することはできません。

Timestream for InfluxDB のセキュリティのベストプラクティス

InfluxDB への書き込みを最適化する

InfluxDB は、他の時系列データベースと同様に、データをリアルタイムで取り込み、処理できるように構築されています。InfluxDB にデータを書き込むときに、システムのパフォーマンスを最大に保つには、次の最適化をお勧めします。

- バッチ書き込み： InfluxDB にデータを書き込むときは、すべての書き込みリクエストに関連するネットワークオーバーヘッドを最小限に抑えるために、データをバッチで書き込みます。最適なバッチサイズは、書き込みリクエストごとに 5,000 行のラインプロトコルです。1 つのリクエストに複数の行を書き込むには、各行プロトコルを新しい行 (\n) で区切る必要があります。
- キーでタグをソートする： データポイントを InfluxDB に書き込む前に、レキシコグラフィーの順序でキーでタグをソートします。

```
measurement,tagC=therefore,tagE=am,tagA=i,tagD=i,tagB=think fieldKey=fieldValue
 1562020262

# Optimized line protocol example with tags sorted by key
measurement,tagA=i,tagB=think,tagC=therefore,tagD=i,tagE=am fieldKey=fieldValue
 1562020262
```

- 可能な限り最も粗い時間精度を使用します： – InfluxDB はナノ秒精度でデータを書き込みますが、データがナノ秒単位で収集されない場合、その精度で書き込む必要はありません。パフォーマンスを向上させるには、タイムスタンプに可能な限り粗い精度を使用します。書き込み精度は、次の場合に指定できます。
  - を使用する場合SDK、ポイントの時間属性を設定する WritePrecision ときに を指定できます。InfluxDB クライアントライブラリの詳細については、「InfluxDB ドキュメント」を参照してください。
  - Telegraf を使用する場合は、Telegraf エージェント設定で時間精度を設定します。精度は、整数 + 単位 (0s、10ms、2us、4s など) の間隔として指定されます。有効な時間単位は、「ns」、「us」、「ms」、「s」です。

```
[agent]
 interval ="10s"
 metric_batch_size="5000"
```

```
  precision = "0s"
```

- gzip 圧縮を使用する： – gzip 圧縮を使用して InfluxDB への書き込みを高速化し、ネットワーク帯域幅を減らします。ベンチマークでは、データが圧縮されると、最大 5 倍の速度向上が示されています。
  - Telegraf を使用する場合は、telelegraf.conf の Influxdb_v2 出力プラグイン設定で content_encoding オプションを gzip に設定します。

```
[[outputs.influxdb_v2]]
  urls = ["http://localhost:8086"]
  # ...
  content_encoding = "gzip"
```

  - クライアントライブラリを使用する場合、各 InfluxDB クライアントライブラリには、書き込みリクエストを圧縮するためのオプションが提供され、デフォルトで圧縮が強制されます。圧縮を有効にする方法は、ライブラリごとに異なります。具体的な手順については、InfluxDB ドキュメントを参照してください。
  - InfluxDB API `/api/v2/write` エンドポイントを使用してデータを書き込む場合は、データを gzip で圧縮し、Content-Encoding ヘッダーを gzip に設定します。

パフォーマンスのための設計

スキーマを設計して、よりシンプルでパフォーマンスの高いクエリを実現します。次のガイドラインは、スキーマがクエリを容易にし、クエリのパフォーマンスを最大化するのに役立ちます。

- クエリする設計： クエリしやすい測定値 、タグキー 、およびフィールドキーを選択します。この目標を達成するには、次の原則に従います。
  - 単純な名前の測定値を使用し、スキーマを正確に記述します。
  - 同じスキーマ内のタグキーとフィールドキーに同じ名前を使用しないでください。
  - タグキーとフィールドキーに予約済み Flux キーワードと特殊文字を使用しないでください。
  - タグには、フィールドを記述し、多くのデータポイントで共通するメタデータが保存されます。
  - フィールドには、一意または高度に可変なデータ、通常は数値データポイントが保存されます。
  - 測定とキーにはデータを含めないでくださいが、データを集約または記述するために使用されます。データはタグ値とフィールド値に保存されます。
- 時系列のカーディナリティをコントロールする ハイシリーズのカーディナリティは、InfluxDB の書き込みおよび読み取りパフォーマンスが低下する主な原因の 1 つです。InfluxDB の高カーディ

ナリティとは、非常に多数の一意のタグ値が存在することを意味します。タグ値は InfluxDB でインデックス化されます。つまり、一意の値が非常に多いとインデックスが大きくなり、データの取り込みとクエリのパフォーマンスが低下する可能性があります。

カーディナリティに関連する潜在的な問題をよりよく理解し、解決するには、次のステップに従います。

- カーディナリティが高い原因を理解する
- バケットのカーディナリティを測定する
- 高いカーディナリティを解決するためのアクションを実行する

- 高シリーズのカーディナリティの原因 InfluxDB は、測定値とタグに基づいてデータをインデックス化し、データ読み取りを高速化します。インデックス付きデータ要素の各セットは、シリーズキー を形成します。一意の IDs、ハッシュ、ランダムな文字列などの可変性の高い情報を含むタグは、多数のシリーズ を引き起こします。これは、ハイシリーズカーディナリティ とも呼ばれます。高シリーズカーディナリティは、InfluxDB のメモリ使用量が高い主なドライバーです。
- シリーズカーディナリティの測定 InfluxDB インスタンスの Timestream でパフォーマンスが遅くなったり、メモリ使用量が増えたりする場合は、バケットのシリーズカーディナリティを測定することをお勧めします。

  InfluxDB には、Flux と InfluxQL の両方でシリーズカーディナリティを測定できる関数が用意されています。

  - Flux では 関数を使用します。 `influxdb.cardinality()`
  - FluxQL では、 `SHOW SERIES CARDINALITY` コマンドを使用します。

  どちらの場合も、エンジンはデータ内の一意のシリーズキーの数を返します。どの Timestream for InfluxDB インスタンスにも 1,000 万を超えるシリーズキーを持つことはお勧めしません。
- 高シリーズのカーディナリティの原因 バケットのいずれかのカーディナリティが高いことが判明した場合は、いくつかの修正手順を実行して修正できます。
  - タグを確認する： ワークロードがケースを生成していないことを確認します。ほとんどのエントリでタグに一意の値があります。これは、一意のタグ値の数は常に時間の経過とともに増加する場合や、すべてのメッセージがタイムスタンプやタグなどの一意の組み合わせを持つログタイプのメッセージがデータベースに書き込まれる場合に発生する可能性があります。次の Flux コードを使用して、どのタグが高カーディナリティの問題に最も寄与しているかを把握できます。

```
// Count unique values for each tag in a bucketimport "influxdata/influxdb/schema"
```

```
cardinalityByTag = (bucket) => schema.tagKeys(bucket: bucket)
    |> map(
        fn: (r) => ({
            tag: r._value,
            _value: if contains(set: ["_stop", "_start"], value: r._value) then
                0
            else
                (schema.tagValues(bucket: bucket, tag: r._value)
                    |> count()
                    |> findRecord(fn: (key) => true, idx: 0))._value,
        }),
    )
    |> group(columns: ["tag"])
    |> sum()

cardinalityByTag(bucket: "example-bucket")
```

カーディナリティが非常に高い場合、上記のクエリがタイムアウトすることがあります。タイムアウトが発生した場合は、以下のクエリを一度に 1 つずつ実行します。

タグのリストを生成します。

```
// Generate a list of tagsimport "influxdata/influxdb/schema"

schema.tagKeys(bucket: "example-bucket")
```

各タグの一意のタグ値をカウントします。

```
// Run the following for each tag to count the number of unique tag valuesimport
 "influxdata/influxdb/schema"

tag = "example-tag-key"

schema.tagValues(bucket: "my-bucket", tag: tag)
    |> count()
```

これらを異なる時点で実行して、どのタグがより速く成長しているかを特定することをお勧めします。

- スキーマの改善： で説明されているモデリングの推奨事項に従ってくださいTimestream for InfluxDB のセキュリティのベストプラクティス。

- 古いデータを削除または集約してカーディナリティを減らす： ユースケースに、カーディナリティの高い問題の原因となっているすべてのデータが必要かどうかを検討します。このデータが不要になったり、頻繁にアクセスされたりした場合は、集約したり、削除したり、Timestream for Live Analytics などの別のエンジンにエクスポートして、長期のストレージと分析を行うことができます。

トラブルシューティング

「dev」バージョンが認識されないという警告

WARN「最新のバックアップ/復元がサポートされていると仮定して、サーバーによって報告されたバージョンを解析できませんでした」という警告APIsが移行中に表示される可能性があります。この警告は無視できます。

復元ステージ中に移行が失敗しました

復元ステージ中に移行に失敗した場合は、 `--retry-restore-dir`フラグを使用して復元を再試みることができます。以前にバックアップされたディレクトリへのパスを持つ `--retry-restore-dir` フラグを使用して、バックアップステージをスキップし、復元ステージを再試行します。復元中に移行が失敗した場合、移行に使用される作成されたバックアップディレクトリが表示されます。

復元が失敗する可能性のある理由は次のとおりです。

- 無効な InfluxDB 送信先トークン – 送信元インスタンスに存在する、送信元インスタンスと同じ名前のバケット。個々のバケット移行では、 `--dest-bucket`オプションを使用して、移行されたバケットに一意の名前を設定します。
- 送信元ホストまたは送信先ホスト、またはオプションの S3 バケットのいずれかで接続が失敗します。

Amazon Timestream for InfluxDB の基本運用ガイドライン

Amazon Timestream for InfluxDB を使用する際に全員が従うべき基本的な運用ガイドラインを次に示します。Amazon Timestream for InfluxDB サービスレベル契約では、以下のガイドラインに従う必要があります。

- メトリクスを使用して、メモリ、CPU、ストレージの使用状況をモニタリングします。使用状況パターンが変化したとき、またはデプロイの容量に近づいたときに通知 CloudWatch するように Amazon を設定できます。このようにして、システムのパフォーマンスと可用性を維持できます。
- 最大ストレージ容量に近づいたら、DB インスタンスをスケールアップする。アプリケーションからのリクエストの予期しない増加に対応するために、ストレージとメモリにいくらかのバッファがあることが必要です。現時点では、これを実現するために新しいインスタンスを作成し、データを移行する必要があることに注意してください。
- データベースのワークロードでプロビジョニングした I/O がより多く必要になると、フェイルオーバーやデータベース障害後の復旧が遅くなります。DB インスタンスの I/O 処理能力を高めるには、以下のいずれかまたはすべての操作を実行します。
  - I/O 容量が大きい別の DB インスタンスに移行します。
  - Influx IOPS搭載ストレージを既に使用している場合は、 より高いストレージタイプIOPSをプロビジョニングします。
- クライアントアプリケーションが DB インスタンスのドメインネームサービス (DNS) データをキャッシュしている場合は、 (TTL) 値を 30 秒未満に設定します time-to-live。DB インスタンスの基になる IP アドレスは、フェイルオーバー後に変更されている可能性があります。したがって、DNSデータを長時間キャッシュすると、接続が失敗する可能性があります。アプリケーションが、使用されなくなった IP アドレスに接続しようとする可能性があります。

DB インスタンスのRAMレコメンデーション

Amazon Timestream for InfluxDB のパフォーマンスのベストプラクティスは、ワーキングセットがほぼ完全にメモリ内に存在するRAMように十分な割り当てを行うことです。作業セットは、インスタンスで頻繁に使用されるデータとインデックスです。DB インスタンスを使用するほど、作業セットが増大します。

InfluxDB の Timestream のセキュリティ

のクラウドセキュリティが最優先事項 AWS です。お客様は AWS 、最もセキュリティに敏感な組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャの恩恵を受けることができます。

セキュリティは、 AWS とユーザーの間で責任を共有します。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティとして説明しています。

- クラウドのセキュリティ – AWS クラウドで AWS サービスを実行するインフラストラクチャを保護する AWS 責任があります。 は、安全に使用できるサービス AWS も提供します。セキュリティの有効性は、AWS コンプライアンスプログラムの一環として、サードパーティーの審査機関によって定期的にテストおよび検証されています。Timestream for InfluxDB に適用されるコンプライアンスプログラムについては、AWS「コンプライアンスプログラムによる対象範囲内のサービス」を参照してください。
- クラウドのセキュリティ – お客様の責任は、使用する AWS サービスによって決まります。また、お客様は、お客様のデータの機密性、組織の要件、および適用可能な法律および規制などの他の要因についても責任を担います。

このドキュメントは、Timestream for InfluxDB を使用する際に責任共有モデルを適用する方法を理解するのに役立ちます。次のトピックでは、セキュリティとコンプライアンスの目標を達成するために、InfluxDB の Timestream を設定する方法を示します。また、Timestream for InfluxDB リソースのモニタリングと保護に役立つ他の AWSサービスの使用方法についても説明します。

トピック

概要

このドキュメントは、Amazon Timestream for InfluxDB を使用する際に責任共有モデルを適用する方法を理解するのに役立ちます。次のトピックでは、セキュリティとコンプライアンスの目標を達成するために Amazon Timestream for InfluxDB を設定する方法を示します。また、Amazon Timestream for InfluxDB リソースをモニタリングして保護するのに役立つ他の AWS サービスを使用する方法も説明します。

DB インスタンスの Amazon Timestream for InfluxDB リソースとデータベースへのアクセスを管理できます。アクセスの管理に使用する方法は、ユーザーが Amazon Timestream for InfluxDB で実行する必要があるタスクのタイプによって異なります。

- ネットワークアクセスコントロール用の Amazon VPCサービスに基づいて、仮想プライベートクラウド (VPC) で DB インスタンスを実行します。
- AWS Identity and Access Management (IAM) ポリシーを使用して、Amazon Timestream for InfluxDB リソースの管理を許可されているユーザーを決定するアクセス許可を割り当てます。例えば、 IAM を使用して、DB インスタンスの作成、説明、変更、削除、リソースのタグ付け、セキュリティグループの変更を許可されているユーザーを特定できます。
- セキュリティグループを使用して、DB EC2インスタンス上のデータベースに接続できる IP アドレスまたは Amazon インスタンスを制御します。DB インスタンスを初めて作成するときは、関連するセキュリティグループによって指定されたルールを介してのみアクセスできます。
- DB インスタンスで Secure Socket Layer (SSL) または Transport Layer Security (TLS) 接続を使用します。
- InfluxDB エンジンのセキュリティ機能を使用して、DB インスタンスのデータベースにログインできるユーザーを制御します。これらの機能は、データベースがローカルネットワーク上にあるかのように動作します。詳細については、「InfluxDB の Timestream のセキュリティ」を参照してください。

> **Note**
>
> 目的のユースケースに対してのみ、セキュリティを設定する必要があります。Amazon Timestream for InfluxDB が管理するプロセスのセキュリティアクセスを設定する必要はありません。このプロセスには、バックアップの作成、プライマリ DB インスタンスとリードレプリカの間のデータのレプリケートなどがあります。

トピック

一般的なセキュリティ

トピック

アクセス許可

InfluxDB ユーザーには、最小権限のアクセス許可を付与する必要があります。移行時には、オペレータートークンではなく、特定のユーザーに付与されたトークンのみを使用する必要があります。

InfluxDB の Timestream は、アクセスIAM許可を使用してユーザーアクセス許可を制御します。ユーザーには、必要な特定のアクションとリソースへのアクセス権を付与することをお勧めします。詳細については、「最小特権アクセスを付与する」を参照してください。

ネットワークアクセス

Influx 移行スクリプトはローカルで機能し、同じシステム上の 2 つの InfluxDB インスタンス間でデータを移行できますが、移行の主なユースケースは、ローカルネットワークまたはパブリックネットワークのいずれかのネットワーク間でデータを移行することであると想定されます。これにはセキュリティ上の考慮事項が伴います。Influx 移行スクリプトは、デフォルトで TLSが有効になっているインスタンスのTLS証明書を検証します。ユーザーは InfluxDB インスタンスTLSで を有効にし、スクリプト`--skip-verify`のオプションを使用しないことをお勧めします。

許可リストを使用して、ネットワークトラフィックが予想されるソースからのトラフィックに制限することをお勧めします。これを行うには、ネットワークトラフィックを既知の からのみ InfluxDB インスタンスに制限しますIPs。

依存関係

Influx 、InfluxDB CLI、Python、Requests モジュール、`mountpoint-s3`および などのオプションの依存関係など、すべての依存関係の最新バージョンを使用する必要があります`rclone`。

S3 バケット

移行用の一時的なストレージとして S3 バケットを使用する場合は、 を有効にしTLS、バージョニングし、パブリックアクセスを無効にすることをお勧めします。

移行に S3 バケットを使用する

1. を開き AWS Management Console、Amazon Simple Storage Service に移動し、バケット を選択します。
2. 使用するバケットを選択します。
3. [アクセス許可] タブを選択します。
4. [ブロックパブリックアクセス (バケット設定)] で [編集] を選択します。
5. すべてのパブリックアクセスをブロックする をチェックします。
6. [Save changes] (変更の保存) をクリックします。
7. [バケットポリシー] で [編集] を選択します。
8. <example-bucket> をバケット名に置き換えて、接続にTLSバージョン 1.2 以降を使用するようにするには、以下を入力します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnforceTLSv12orHigher",
            "Principal": {
                "AWS": "*"
            },
            "Action": [
                "s3:*"
            ],
            "Effect": "Deny",
            "Resource": [
                "arn:aws:s3:::<example bucket>/*",
                "arn:aws:s3:::<example bucket>"
            ],
            "Condition": {
                "NumericLessThan": {
                    "s3:TlsVersion": 1.2
                }
            }
        }
```

```
    ]
}
```

9. [Save changes] (変更の保存) をクリックします。
10. [プロパティ] タブを選択します。
11. [バケットのバージョニング] で [編集] を選択します。
12. 「 を有効にする」をチェックします。
13. [Save changes] (変更の保存) をクリックします。

Amazon S3 バケットのベストプラクティスについては、「Amazon Simple Storage Service のセキュリティのベストプラクティス」を参照してください。

Amazon Timestream for InfluxDB によるデータベース認証

Amazon Timestream for InfluxDB は、データベースユーザーを認証する 2 つの方法をサポートしています。

パスワードとアクセストークンデータベース認証では、データベースを認証するさまざまな方法を使用します。したがって、特定のユーザーは、1 つの認証方法のみを使用してデータベースにログインできます。どちらの場合も、InfluxDB はユーザーアカウントとAPIトークンのすべての管理を実行します。

パスワード認証

InfluxDB DB インスタンスの作成プロセス中に、組織、ユーザー、パスワードを作成しました。ユーザーには、InfluxDB DB インスタンスの Timestream のすべてのを管理するアクセス許可があります。このユーザー名とパスワードの組み合わせを使用すると、InfluxUI を使用してインスタンス LogIn に入り、InfluxCLI を使用してオペレータトークンを生成できます。

ユーザーの作成、バケットの削除、組織などには、オペレータトークンが必要です。詳細については、「データベース認証オプション」を参照してください。

API トークン

InfluxDB APIトークンは、InfluxDB とクライアントやアプリケーションなどの外部ツールとの間の安全なインタラクションを保証します。API トークンは特定のユーザーに属し、ユーザーの組織内の InfluxDB アクセス許可を識別します。

InfluxDB には 3 種類のAPIトークンがあります。

- オペレータトークン: InfluxDB OSS2.x のすべての組織とすべての組織リソースへの完全な読み取りおよび書き込みアクセスを許可します。サーバー設定の取得など、一部のオペレーションでは、オペレータのアクセス許可が必要です。CLI セットアッププロセスが完了したら、InfluxDB UIapi/v2API、、または Influx を使用してオペレータトークンを手動で作成するには、既存のオペレータトークンまたはユーザー名とパスワードを使用する必要があります。既存の演算子トークンを使用せずに新しい演算子トークンを作成するには、influxd リカバリ認証 を参照してください CLI。

  > **Important**
  >
  > オペレータトークンにはデータベース内のすべての組織への完全な読み取りおよび書き込みアクセスがあるため、組織ごとにオールアクセストークンを作成し、それらを使用して InfluxDB を管理することをお勧めします。これにより、組織全体での偶発的なやり取りを防ぐことができます。

- オールアクセスAPIトークン: 組織内のすべてのリソースへの完全な読み取りおよび書き込みアクセスを許可します。
- 読み取り/書き込みトークン: 組織内の特定のバケットへの読み取りアクセス、書き込みアクセス、またはその両方を許可します。

すべての InfluxDb トークンは有効期限が設定されていない長寿命トークンであるため、オペレーターまたはすべてのアクセストークンを使用してクライアントまたは Telegraf エージェントからモニタリングデータを送信することはお勧めしません。また、ダッシュボードアプリケーションに埋め込むこともお勧めしません。これらのアプリケーションでは、ジョブを完了するために必要なアクセス許可のみを持つ読み取り/書き込みトークンを作成します。influxDB トークンの作成方法の詳細については、「トークンの作成」を参照してください。

シークレット

InfluxDB オペレータトークンはインスタンス設定で生成されます。オールアクセストークンや読み取り/書き込みトークンなどの他の種類のトークンは、Influx CLI、Influx v2API、または Timestream for InfluxDB マルチユーザーローテーション関数を使用して作成できます。API トークンの生成、表示、割り当て、削除方法については、「トークンの管理」を参照してください。

InfluxDB トークンの Timestream は、 を使用して頻繁にローテーション AWS Secrets Manager し、環境変数を介してトークンを保存することをお勧めします。環境変数でのトークンの使用と、InfluxDB ユーザーとトークンの Timestream をローテーションする方法についてはシークレットのローテーション、「トークンの使用」を参照してください。

以下も参照してください。

- Amazon Timestream for InfluxDB のインフラストラクチャセキュリティ
- Timestream for InfluxDB のセキュリティのベストプラクティス

Amazon Timestream for InfluxDB でシークレットを使用する方法

InfluxDB の Timestream は、ユーザーインターフェイスを介したユーザー名とパスワード認証と、最小権限のクライアントとアプリケーション接続のトークン認証情報をサポートしています。InfluxDB ユーザーの Timestream には組織内の`allAccess`アクセス許可があり、トークンには任意のアクセス許可セットを設定できます。安全なAPIトークン管理のベストプラクティスに従って、組織内のきめ細かなアクセスのためのトークンを管理するためにユーザーを作成する必要があります。Timestream for InfluxDB の管理のベストプラクティスに関する追加情報は、Influxdata ドキュメント を参照してください。

AWS Secrets Manager は、データベースの認証情報、APIキー、その他の機密情報を保護するために使用できるシークレットストレージサービスです。次に、コード内で、ハードコードされた認証情報を Secrets Manager へのAPI呼び出しに置き換えることができます。これにより、シークレットはそこにないため、コードを調べている誰かによってシークレットが侵害されないようになります。Secrets Manager の概要については、AWS 「Secrets Manager とは」を参照してください。

データベースインスタンスを作成すると、Timestream for InfluxDB は、マルチユーザーローテーション AWS Lambda 関数で使用するための管理シークレットを自動的に作成します。InfluxDB ユーザーとトークンの Timestream をローテーションするには、ローテーションするユーザーまたはトークンごとに手動で新しいシークレットを作成する必要があります。各シークレットは、Lambda 関数を使用してスケジュールでローテーションするように設定できます。新しいローテーションシークレットを設定するプロセスは、Lambda 関数コードのアップロード、Lambda ロールの設定、新しいシークレットの定義、シークレットローテーションスケジュールの設定で構成されます。

シークレットの内容

シークレットに InfluxDB ユーザー認証情報の Timestream を保存するときは、次の形式を使用します。

シングルユーザー：

```
{
  "engine": "<required: must be set to 'timestream-influxdb'>",
  "username": "<required: username>",
```

```
  "password": "<required: password>",
  "dbIdentifier": "<required: DB identifier>"
}
```

InfluxDB インスタンスの Timestream を作成すると、マルチユーザー Lambda 関数で使用する認証情報を使用して、管理者シークレットが Secrets Manager に自動的に保存されます。adminSecretArn を DB インスタンスの概要ページにあるAuthentication Properties Secret Manager ARN値に設定するか、管理者シークレットARNの に設定します。新しい管理者シークレットを作成するには、関連付けられた認証情報を既に持っている必要があり、認証情報には管理者権限が必要です。

シークレットに InfluxDB トークン認証情報の Timestream を保存するときは、次の形式を使用します。

マルチユーザー：

```
{
  "engine": "<required: must be set to 'timestream-influxdb'>",
  "org": "<required: organization to associate token with>",
  "adminSecretArn": "<required: ARN of the admin secret>",
  "type": "<required: allAccess or operator or custom>",
  "dbIdentifier": "<required: DB identifier>",
  "token": "<required unless generating a new token: token being rotated>",
  "writeBuckets": "<optional: list of bucketIDs for custom type token, must be input
 within plaintext panel, for example ['id1','id2']>",
  "readBuckets": "<optional: list of bucketIDs for custom type token, must be input
 within plaintext panel, for example ['id1','id2']>",
  "permissions": "<optional: list of permissions for custom type token, must be input
 within plaintext panel, for example ['write-tasks','read-tasks']>"
}
```

シークレットに InfluxDB 管理者認証情報の Timestream を保存するときは、次の形式を使用します。

管理者シークレット：

```
{
  "engine": "<required: must be set to 'timestream-influxdb'>",
  "username": "<required: username>",
  "password": "<required: password>",
  "dbIdentifier": "<required: DB identifier>",
```

```
  "organization": "<optional: initial organization>",
  "bucket": "<optional: initial bucket>"
}
```

シークレットの自動ローテーションを有効にするには、シークレットが正しいJSON構造にある必要があります。InfluxDB シークレットの Timestream をローテーションする方法については、シークレットのローテーション「」を参照してください。

シークレットの変更

InfluxDB インスタンス作成プロセスの Timestream 中に生成された認証情報は、アカウントの Secrets Manager シークレットに保存されます。GetDbInstance レスポンスオブジェクトには、Amazon リソースネーム (ARN) をそのようなシークレットに保持`influxAuthParametersSecretArn`する が含まれています。シークレットは、Timestream for InfluxDB インスタンスが使用可能になった後にのみ入力されます。このシークレットupdates/modifications/deletionsへの READONLY のコピーは、作成された DB インスタンスには影響しません。このシークレットを削除しても、APIレスポンスは削除されたシークレット を参照しますARN。

既存のトークン認証情報を保存するのではなく、Timestream for InfluxDB インスタンスに新しいトークンを作成するには、シークレットに`token`値を空白のままにして、`AUTHENTICATION_CREATION_ENABLED`Lambda 環境変数を に設定してマルチユーザーローテーション関数を使用して、非オペレータトークンを作成できます`true`。新しいトークンを作成すると、シークレットで定義されたアクセス許可がトークンに割り当てられ、最初のローテーションが成功した後に変更することはできません。シークレットのローテーションの詳細については、「Rotating AWS Secrets Manager Secrets」を参照してください。

シークレットが削除された場合、InfluxDB インスタンスの Timestream 内の関連付けられたユーザーまたはトークンは削除されません。

シークレットのローテーション

InfluxDB の Timestream シングルユーザーおよびマルチユーザーローテーション Lambda 関数を使用して、InfluxDB ユーザーおよびトークン認証情報の Timestream をローテーションします。シングルユーザー Lambda 関数を使用して、InfluxDB インスタンスの Timestream のユーザー認証情報をローテーションし、マルチユーザー Lambda 関数を使用して、InfluxDB インスタンスの Timestream のトークン認証情報をローテーションします。

シングルユーザーおよびマルチユーザー Lambda 関数を使用したユーザーとトークンのローテーションはオプションです。InfluxDB 認証情報のタイムストリームは期限切れにならず、公開され

た認証情報は DB インスタンスに対する悪意のあるアクションのリスクをもたらします。Secrets Manager で InfluxDB 認証情報の Timestream をローテーションする利点は、公開された認証情報の攻撃ベクトルを次のローテーションサイクルまでの時間枠に制限するセキュリティレイヤーを追加することです。DB インスタンスにローテーションメカニズムがない場合、公開された認証情報は手動で削除されるまで有効です。

指定したスケジュールに従って自動的にシークレットを更新するように Secrets Manager を設定することができます。これにより、長期のシークレットを短期のシークレットに置き換えることが可能となり、侵害されるリスクが大幅に減少します。Secrets Manager によるシークレットのローテーションの詳細については、「Rotation AWS Secrets Manager Secrets」を参照してください。

ユーザーをローテーションする

シングルユーザー Lambda 関数を使用してユーザーをローテーションすると、ローテーションのたびに新しいランダムパスワードがユーザーに割り当てられます。自動ローテーションを有効にする方法の詳細については、「データベース以外の AWS Secrets Manager シークレットの自動ローテーションの設定」を参照してください。

管理者シークレットのローテーション

管理者シークレットをローテーションするには、シングルユーザーローテーション関数を使用します。シークレットには `engine`と `dbIdentifier`の値を追加する必要があります。これらの値は DB 初期化時に自動的に入力されないためです。完全なシークレットテンプレートシークレットの内容については、「」を参照してください。

InfluxDB インスタンスの Timestream の管理シークレットを見つけるには、InfluxDB インスタンスの Timestream の概要ページARNから 管理シークレットを使用します。管理者ユーザーは InfluxDB インスタンスの Timestream のアクセス許可が昇格しているため、InfluxDB 管理シークレットのすべての Timestream をローテーションすることをお勧めします。

Lambda ローテーション関数

新しいシークレットで を使用し、InfluxDB ユーザーの Timestream に必要なフィールドを追加することで、シングルユーザーローテーション関数シークレットの内容を使用して InfluxDB ユーザーの Timestream をローテーションできます。シークレットローテーション Lambda 関数の詳細については、「Lambda 関数によるローテーション」を参照してください。

単一ユーザーローテーション関数は、シークレットで定義された認証情報を使用して Timestream for InfluxDB DB インスタンスで認証し、新しいランダムパスワードを生成し、ユーザーの新しいパス

ワードを設定します。シークレットローテーション Lambda 関数の詳細については、「Lambda 関数によるローテーション」を参照してください。

Lambda 関数実行ロールのアクセス許可

シングルユーザー Lambda 関数のロールとして、次のIAMポリシーを使用します。このポリシーは、Lambda 関数に、InfluxDB ユーザーの Timestream のシークレットローテーションを実行するために必要なアクセス許可を付与します。

IAM ポリシーに以下に記載されているすべての項目を AWS 、アカウントの値に置き換えます。

- {rotating_secret_arn} — ローテーションされるシークレットARNの は、Secrets Manager シークレットの詳細にあります。
- {db_instance_arn} — InfluxDB インスタンスの Timestream ARNは、InfluxDB インスタンスの Timestream の概要ページにあります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "secretsmanager:UpdateSecretVersionStage"
            ],
     "Resource": "{rotating_secret_arn}"
        },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword"
            ],
            "Resource": "*"
        },
        {
            "Action": [
                "timestream-influxdb:GetDbInstance"
            ],
     "Resource": "{db_instance_arn}",
            "Effect": "Allow"
```

```
        }
    ]
}
```

トークンのローテーション

新しいシークレットで を使用し、InfluxDB トークンの Timestream に必要なフィールドを追加することで、マルチユーザーローテーション関数シークレットの内容で InfluxDB トークンの Timestream をローテーションできます。シークレットローテーション Lambda 関数の詳細については、「Lambda 関数によるローテーション」を参照してください。

InfluxDB マルチユーザー Lambda 関数の Timestream を使用して、InfluxDB トークンの Timestream をローテーションできます。Lambda 設定trueでAUTHENTICATION_CREATION_ENABLED環境変数を に設定して、トークンの作成を有効にします。新しいトークンを作成するには、シークレット値の シークレットの内容 を使用します。新しいシークレットでtokenキーと値のペアを省略し、typeを に設定するかallAccess、特定のアクセス許可を定義してタイプを に設定しますcustom。ローテーション関数は、最初のローテーションサイクル中に新しいトークンを作成します。ローテーション後にシークレットを編集してトークンのアクセス許可を変更することはできません。それ以降のローテーションでは、DB インスタンスで設定されたアクセス許可が使用されます。

Lambda ローテーション関数

マルチユーザーローテーション関数は、管理者シークレットの管理者認証情報を使用して新しいアクセス許可の同一のトークンを作成することで、トークン認証情報をローテーションします。Lambda 関数は、置換トークンを作成し、新しいトークン値をシークレットに保存し、古いトークンを削除する前に、シークレット内のトークン値を検証します。Lambda 関数が新しいトークンを作成している場合、まずAUTHENTICATION_CREATION_ENABLED環境変数が に設定されていることtrue、シークレットにトークン値がないこと、およびトークンタイプがタイプ演算子ではないことを確認します。

Lambda 関数実行ロールのアクセス許可

マルチユーザー Lambda 関数のロールとして、次のIAMポリシーを使用します。このポリシーは、Lambda 関数に、InfluxDB トークンの Timestream のシークレットローテーションを実行するために必要なアクセス許可を付与します。

IAM ポリシーに以下に記載されているすべての項目を AWS 、アカウントの値に置き換えます。

- {rotating_secret_arn} — ローテーションされるシークレットARNの は、Secrets Manager シークレットの詳細にあります。

- {authentication_properties_admin_secret_arn } — InfluxDB 管理シークレットの Timestream ARN は、InfluxDB インスタンスの Timestream の概要ページにあります。
- {db_instance_arn} — InfluxDB インスタンスの Timestream ARNは、InfluxDB インスタンスの Timestream の概要ページにあります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "secretsmanager:UpdateSecretVersionStage"
            ],
     "Resource": "{rotating_secret_arn}"
        },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
     "Resource": "{authentication_properties_admin_secret_arn}"
        },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword"
            ],
            "Resource": "*"
        },
        {
            "Action": [
                "timestream-influxdb:GetDbInstance"
            ],
     "Resource": "{db_instance_arn}",
            "Effect": "Allow"
        }
    ]
}
```

InfluxDB の Timestream でのデータ保護

責任 AWS 共有モデル 、Amazon Timestream for InfluxDB のデータ保護に適用されます。このモデルで説明されているように、 AWS は、すべての を実行するグローバルインフラストラクチャを保護する責任があります AWS クラウド。お客様は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。また、使用する AWS のサービス のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、「データプライバシーFAQ」を参照してください。欧州でのデータ保護の詳細については、AWS 「 責任共有モデル」とGDPRAWS 「セキュリティブログ」のブログ記事を参照してください。

データ保護の目的で、認証情報を保護し AWS アカウント 、 AWS IAM Identity Center または AWS Identity and Access Management () を使用して個々のユーザーを設定することをお勧めしますIAM。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。1.2 が必要でTLS、1.3 TLS をお勧めします。
- で APIとユーザーアクティビティの ログ記録を設定します AWS CloudTrail。証 CloudTrail 跡を使用して AWS アクティビティをキャプチャする方法については、AWS CloudTrail 「 ユーザーガイド」の CloudTrail 「証跡の操作」を参照してください。
- AWS 暗号化ソリューションと、 内のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。
- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは AWS を介して にアクセスするときに FIPS 140-3 検証済みの暗号化モジュールが必要な場合はAPI、FIPSエンドポイントを使用します。利用可能なFIPSエンドポイントの詳細については、「連邦情報処理標準 (FIPS) 140-3」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報は、タグ、または名前フィールドなどの自由形式のテキストフィールドに配置しないことを強くお勧めします。これには、コンソール、、API AWS CLIまたは を使用して、Timestream for InfluxDB またはその他の AWS のサービス を操作する場合が含まれます AWS SDKs。名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。URL を外部サーバーに提供する場合は、そのサーバーへのリクエストを検証URLするために認証情報を に含めないことを強くお勧めします。

Timestream for InfluxDB データ保護トピックの詳細については、保管時の暗号化やキー管理など、以下の利用可能なトピックのいずれかを選択してください。

トピック

- 保管中の暗号化
- 送信中の暗号化

保管中の暗号化

保管中の InfluxDB 暗号化のタイムストリームは、 AWS Key Management Service （ AWS KMS ） に保存されている暗号化キーを使用して保管中のすべてのデータを暗号化することで、セキュリティを強化します。この機能は、機密データの保護における負担と複雑な作業を減らすのに役立ちます。保管時に暗号化することで、セキュリティを重視したアプリケーションを構築して、暗号化のコンプライアンスと規制の厳格な要件を満たすことができます。

- 暗号化は、InfluxDB DB インスタンスの Timestream でデフォルトで有効になっており、オフにすることはできません。業界標準の AES-256 暗号化アルゴリズムは、使用されるデフォルトの暗号化アルゴリズムです。
- AWS KMS は、InfluxDB の Timestream に保存中の暗号化に使用されます。
- 暗号化を使用するように DB インスタンスクライアントアプリケーションを変更する必要はありません。

送信中の暗号化

InfluxDB の Timestream のデータはすべて、転送中に暗号化されます。デフォルトでは、Timestream for InfluxDB との間のすべての通信は、Transport Layer Security (TLS) 暗号化を使用して保護されます。

Amazon Timestream for InfluxDB との間のトラフィックは、サポートされているTLSバージョン 1.2 または 1.3 を使用して保護されます。

Amazon Timestream for InfluxDB の ID とアクセスの管理

AWS Identity and Access Management （ IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を認証 (サインイン) し、誰に

Timestream for InfluxDB リソースの使用を許可する (アクセス許可を持つ) かを制御します。IAM は追加料金なしで AWS のサービス 使用できる です。

トピック

- アイデンティティを使用した認証
- ポリシーを使用したアクセスの管理
- Amazon Timestream for InfluxDB の での仕組み IAM
- Amazon Timestream for InfluxDB のアイデンティティベースのポリシーの例
- Amazon Timestream for InfluxDB の ID とアクセスのトラブルシューティング
- での DB インスタンスへのアクセスの制御 VPC
- Amazon Timestream for InfluxDB のサービスリンクロールの使用
- AWS Amazon Timestream for InfluxDB の マネージドポリシー
- VPC エンドポイントを介した InfluxDB の Timestream への接続

アイデンティティを使用した認証

認証は、アイデンティティ認証情報 AWS を使用して にサインインする方法です。として、IAMユーザーとして AWS アカウントのルートユーザー、またはIAMロールを引き受けることで、認証 (にサインイン AWS) される必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。 AWS IAM Identity Center （ IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook の認証情報は、フェデレーティッド ID の例です。フェデレーティッド ID としてサインインすると、管理者は以前に IAMロールを使用して ID フェデレーションをセットアップしていました。フェデレーション AWS を使用して にアクセスすると、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、 AWS サインイン ユーザーガイドの「 へのサインイン方法 AWS アカウント」を参照してください。

AWS プログラムで にアクセスする場合、 はソフトウェア開発キット (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、認証情報を使用してリクエストに暗号化して署名します。 AWS ツールを使用しない場合は、自分でリクエストに署名する必要があります。推奨される方法を使用してリクエストに署名する方法については、 IAM ユーザーガイドのAWS API「リクエストの署名バージョン 4」を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。例えば、 では、アカウントのセキュリティを高めるために多要素認証 (MFA) を使用する AWS ことをお勧めします。詳細については、AWS IAM Identity Center 「 ユーザーガイド」の「多要素認証」とIAM「 ユーザーガイド」のAWS 「多要素認証IAM」を参照してください。

IAM ユーザーとグループ

IAM ユーザーとは、1 人のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内の ID です。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を持つIAMユーザーを作成する代わりに、一時的な認証情報に依存することをお勧めします。ただし、IAMユーザーとの長期的な認証情報を必要とする特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、IAM「 ユーザーガイド」の「長期的な認証情報を必要とするユースケースのアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAMユーザーのコレクションを指定する ID です。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、という名前のグループがありIAMAdmins、そのグループにIAMリソースを管理するアクセス許可を付与できます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時認証情報が提供されます。詳細については、「 ユーザーガイド」のIAM「ユーザーのユースケース」を参照してください。 IAM

IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内の ID です。ユーザーと似ていますがIAM、特定の人物には関連付けられていません。でIAMロールを一時的に引き受けるには AWS Management Console、ユーザーからIAMロール (コンソール) に切り替えることができます。または AWS API オペレーションを AWS CLI 呼び出すか、カスタム を使用してロールを引き受けることができますURL。ロールを使用する方法の詳細については、 IAM ユーザーガイドの「ロールを引き受ける方法」を参照してください。

IAM 一時的な認証情報を持つ ロールは、次の状況で役立ちます。

- フェデレーションユーザーアクセス – フェデレーティッド ID に許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーティッド ID が認証されると、その ID は

ロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションのロールの詳細については、 IAM ユーザーガイドの「サードパーティー ID プロバイダーのロールの作成」を参照してください。IAM Identity Center を使用する場合は、アクセス許可セットを設定します。ID が認証された後にアクセスできる内容を制御するために、IAMIdentity Center はアクセス許可セットを のロールに関連付けますIAM。アクセス許可セットの詳細については、「AWS IAM Identity Center ユーザーガイド」の「アクセス許可セット」を参照してください。

- 一時的なIAMユーザーアクセス許可 – IAM ユーザーまたはロールは、特定のタスクに対して異なるアクセス許可を一時的に引き受けるIAMロールを引き受けることができます。
- クロスアカウントアクセス — IAMロールを使用して、別のアカウントの誰か (信頼できるプリンシパル) が自分のアカウントのリソースにアクセスすることを許可できます。クロスアカウントアクセスを許可する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、 (プロキシとしてロールを使用する代わりに) リソースに直接ポリシーをアタッチできます。クロスアカウントアクセスのロールとリソースベースのポリシーの違いについては、IAM「 ユーザーガイド」の「 のクロスアカウントリソースアクセスIAM」を参照してください。
- クロスサービスアクセス — 他の の機能 AWS のサービス を使用するものもあります AWS のサービス。例えば、 サービスで呼び出しを行う場合、そのサービスが Amazon でアプリケーションを実行EC2したりAmazon S3にオブジェクトを保存したりするのが一般的です。サービスでは、呼び出し元プリンシパルの許可、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用して でアクションを実行すると AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、を呼び出すプリンシパルのアクセス許可と AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストのリクエストを使用します。FAS リクエストは、サービスが他の AWS のサービス または リソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するための権限が必要です。FAS リクエストを行う際のポリシーの詳細については、「アクセスセッションの転送」を参照してください。
  - サービスロール – サービスロールは、ユーザーに代わってアクションを実行するためにサービスが引き受けるIAMロールです。IAM 管理者は、 内からサービスロールを作成、変更、削除できますIAM。詳細については、IAM「 ユーザーガイド」の「 にアクセス許可を委任するロールの作成 AWS のサービス」を参照してください。
  - サービスにリンクされたロール – サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカ

ウント 、 サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールのアクセス許可を表示することはできますが、編集することはできません。

- Amazon で実行されているアプリケーション EC2 – IAMロールを使用して、EC2インスタンスで実行され、 AWS CLI または AWS API リクエストを行うアプリケーションの一時的な認証情報を管理できます。これは、EC2インスタンス内にアクセスキーを保存するよりも望ましいです。EC2インスタンスに AWS ロールを割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロファイルには ロールが含まれており、EC2インスタンスで実行されているプログラムが一時的な認証情報を取得できるようにします。詳細については、IAM「 ユーザーガイド」のIAM「ロールを使用して Amazon EC2インスタンスで実行されているアプリケーションにアクセス許可を付与する」を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御するには、ポリシー AWS を作成し、 AWS ID またはリソースにアタッチします。ポリシーは AWS 、アイデンティティまたはリソースに関連付けられているときにアクセス許可を定義するオブジェクトです。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッション) がリクエストを行うときに、これらのポリシー AWS を評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。ほとんどのポリシーはJSONドキュメント AWS として に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、IAM「 ユーザーガイド」のJSON「ポリシーの概要」を参照してください。

管理者はポリシーを使用して AWS JSON、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。必要なリソースに対してアクションを実行するアクセス許可をユーザーに付与するには、IAM管理者はIAMポリシーを作成できます。その後、管理者はIAMポリシーをロールに追加し、ユーザーはロールを引き受けることができます。

IAM ポリシーは、オペレーションの実行に使用する方法に関係なく、アクションのアクセス許可を定義します。例えば、`iam:GetRole`アクションを許可するポリシーがあるとします。そのポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS からロール情報を取得できますAPI。

アイデンティティベースのポリシー

ID ベースのポリシーは、IAMユーザー、ユーザーのグループ、ロールなどの ID にアタッチできるJSONアクセス許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行でき

るアクション、リソース、および条件をコントロールします。ID ベースのポリシーを作成する方法については、IAM「 ユーザーガイド」の「カスタマーマネージドポリシーによるカスタムIAMアクセス許可の定義」を参照してください。

アイデンティティベースのポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。マネージドポリシーは、 内の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには AWS 、管理ポリシーとカスタマー管理ポリシーが含まれます。管理ポリシーまたはインラインポリシーを選択する方法については、 IAM ユーザーガイドの「管理ポリシーとインラインポリシーの選択」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースにアタッチするJSONポリシードキュメントです。リソースベースのポリシーの例としては、IAMロール信頼ポリシー と Amazon S3 バケットポリシー があります。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスをコントロールできます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーIAMでは、 から AWS 管理ポリシーを使用することはできません。

アクセスコントロールリスト (ACLs)

アクセスコントロールリスト (ACLs) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするためのアクセス許可を持っているかを制御します。ACLs はリソースベースのポリシーに似ていますが、JSONポリシードキュメント形式は使用していません。

Amazon S3、および Amazon VPCは AWS WAF、 をサポートするサービスの例ですACLs。の詳細についてはACLs、「Amazon Simple Storage Service デベロッパーガイド」の「アクセスコントロールリスト (ACL) 概要」を参照してください。

その他のポリシータイプ

AWS は、追加であまり一般的ではないポリシータイプをサポートします。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 – アクセス許可の境界は、アイデンティティベースのポリシーがIAMエンティティ (IAMユーザーまたはロール) に付与できる最大アクセス許可を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。Principal フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界の詳細については、IAM「 ユーザーガイド」のIAM「エンティティのアクセス許可の境界」を参照してください。
- サービスコントロールポリシー (SCPs) – SCPs は、 の組織または組織単位 (OU) の最大アクセス許可を指定するJSONポリシーです AWS Organizations。 AWS Organizations は、ビジネスが所有する複数の をグループ化して一元管理するためのサービス AWS アカウント です。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCPs) をアカウントの一部またはすべてに適用できます。は、各 を含むメンバーアカウントのエンティティのアクセス許可SCPを制限します AWS アカウントのルートユーザー。Organizations と の詳細についてはSCPs、AWS Organizations 「 ユーザーガイド」の「サービスコントロールポリシー」を参照してください。
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、「 ユーザーガイド」の「セッションポリシー」を参照してください。 IAM

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。が複数のポリシータイプが関与する場合にリクエストを許可するかどうか AWS を決定する方法については、「 ユーザーガイド」の「ポリシー評価ロジック」を参照してください。 IAM

Amazon Timestream for InfluxDB の での仕組み IAM

IAM Amazon Timestream for InfluxDB で使用できる機能

IAM 機能	InfluxDB サポートのタイムストリーム
アイデンティティベースのポリシー	あり

IAM 機能	InfluxDB サポートのタイムストリーム
リソースベースのポリシー	なし
ポリシーアクション	あり
ポリシーリソース	Yes
ポリシー条件キー	不可
ACLs	なし
ABAC (ポリシーのタグ)	可能
一時的な認証情報	あり
プリンシパル権限	あり
サービスロール	いいえ
サービスリンクロール	可能

Timestream for InfluxDB およびその他の AWS サービスがほとんどのIAM機能でどのように機能するかの概要については、 IAM ユーザーガイドのAWS 「 で機能する のサービスIAM」を参照してください。

Timestream for InfluxDB のアイデンティティベースのポリシー

アイデンティティベースのポリシーのサポート: あり

ID ベースのポリシーは、IAMユーザー、ユーザーのグループ、ロールなどの ID にアタッチできるJSONアクセス許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。ID ベースのポリシーを作成する方法については、IAM「 ユーザーガイド」の「カスタマーマネージドポリシーによるカスタムIAMアクセス許可の定義」を参照してください。

IAM ID ベースのポリシーでは、許可または拒否されたアクションとリソース、およびアクションが許可または拒否される条件を指定できます。プリンシパルは、それが添付されているユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシー

で使用できるすべての要素については、IAM「 ユーザーガイド」のIAMJSON「ポリシー要素リファレンス」を参照してください。

Timestream for InfluxDB のアイデンティティベースのポリシーの例

InfluxDB アイデンティティベースのポリシーの Timestream の例を表示するには、「」を参照してくださいAmazon Timestream for InfluxDB のアイデンティティベースのポリシーの例。

InfluxDB の Timestream 内のリソースベースのポリシー

リソースベースのポリシーのサポート: なし

リソースベースのポリシーは、リソースにアタッチするJSONポリシードキュメントです。リソースベースのポリシーの例としては、IAMロール信頼ポリシー と Amazon S3 バケットポリシー があります。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスをコントロールできます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

クロスアカウントアクセスを有効にするには、リソースベースのポリシーのプリンシパルとして、別のアカウントのアカウントまたはIAMエンティティ全体を指定できます。リソースベースのポリシーにクロスアカウントのプリンシパルを追加しても、信頼関係は半分しか確立されない点に注意してください。プリンシパルとリソースが異なる にある場合 AWS アカウント、信頼されたアカウントのIAM管理者は、プリンシパルエンティティ (ユーザーまたはロール) にリソースへのアクセス許可も付与する必要があります。IAM 管理者は、アイデンティティベースのポリシーをエンティティにアタッチすることで権限を付与します。ただし、リソースベースのポリシーで、同じアカウントのプリンシパルへのアクセス権が付与されている場合は、アイデンティティベースのポリシーをさらに付与する必要はありません。詳細については、IAM「 ユーザーガイド」の「 のクロスアカウントリソースアクセスIAM」を参照してください。

InfluxDB の Timestream のポリシーアクション

ポリシーアクションのサポート: あり

管理者はポリシーを使用して AWS JSON、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

JSON ポリシーの Action要素は、ポリシー内のアクセスを許可または拒否するために使用できるアクションを記述します。ポリシーアクションは通常、関連付けられた AWS APIオペレーションと同じ名前です。一致するAPIオペレーションがないアクセス許可のみのアクションなど、いくつかの例外があります。また、ポリシーに複数のアクションが必要なオペレーションもあります。これらの追加アクションは、依存アクションと呼ばれます。

このアクションは、関連付けられたオペレーションを実行するための権限を付与するポリシーで使用されます。

InfluxDB アクションの Timestream のリストを確認するには、「サービス認証リファレンス」の「InfluxDB の Amazon Timestream で定義されるアクション」を参照してください。

InfluxDB の Timestream のポリシーアクションは、アクションの前に次のプレフィックスを使用します。

```
timestream-influxdb
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [
      "timestream-influxdb:action1",
      "timestream-influxdb:action2"
         ]
```

ワイルドカード (*) を使用して複数アクションを指定できます。例えば、Describe という単語で始まるすべてのアクションを指定するには、次のアクションを含めます。

```
"Action": "timestream-influxdb:Describe*"
```

InfluxDB の Timestream のポリシーリソース

ポリシーリソースのサポート: あり

管理者はポリシーを使用して AWS JSON、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Resource JSON ポリシー要素は、アクションが適用されるオブジェクトを指定します。ステートメントには、Resource または NotResource 要素を含める必要があります。ベストプラクティス

として、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの許可と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

InfluxDB リソースタイプの Timestream とその のリストを確認するにはARNs、「サービス認証リファレンス」の「InfluxDB の Amazon Timestream で定義されるリソース」を参照してください。各リソースARNの を指定できるアクションについては、InfluxDB の Amazon Timestream で定義されるアクション」を参照してください。

InfluxDB の Timestream のポリシー条件キー

サービス固有のポリシー条件キーをサポート： いいえ

管理者はポリシーを使用して AWS JSON、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルが、どのリソースに対してどのような条件下でアクションを実行できるかということです。

Condition 要素 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。イコールや未満などの 条件演算子 を使用して条件式を作成することで、ポリシーの条件とリクエスト内の値を一致させることができます。

1 つのステートメントに複数の Condition 要素を指定する場合、または 1 つの Condition 要素に複数のキーを指定する場合、 AWS では AND 論理演算子を使用してそれらを評価します。1 つの条件キーに複数の値を指定すると、 は論理ORオペレーションを使用して条件 AWS を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば、IAMユーザー名でタグ付けされている場合にのみ、リソースにアクセスするアクセス許可をIAMユーザーに付与できます。詳細については、「 ユーザーガイド」のIAM「ポリシー要素: 変数とタグ」を参照してください。 IAM

AWS は、グローバル条件キーとサービス固有の条件キーをサポートします。すべての AWS グローバル条件キーを確認するには、 ユーザーガイドのAWS 「グローバル条件コンテキストキー」を参照してください。 IAM

InfluxDB の Timestream のアクセスコントロールリスト (ACLs)

をサポートACLs： なし

アクセスコントロールリスト (ACLs) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするアクセス許可を持っているかを制御します。ACLs はリソースベースのポリシーに似ていますが、JSONポリシードキュメント形式は使用しません。

Timestream for InfluxDB を使用した属性ベースのアクセスコントロール (ABAC)

サポート ABAC (ポリシー内のタグ): はい

属性ベースのアクセスコントロール (ABAC) は、属性に基づいてアクセス許可を定義する認証戦略です。では AWS、これらの属性はタグ と呼ばれます。IAM エンティティ (ユーザーまたはロール) と多くの AWS リソースにタグをアタッチできます。エンティティとリソースのタグ付けは、 の最初のステップですABAC。次に、プリンシパルのタグがアクセスしようとしているリソースのタグと一致するときに、オペレーションを許可するABACポリシーを設計します。

ABAC は、急速に成長している環境で役立ち、ポリシー管理が面倒になる状況に役立ちます。

タグに基づいてアクセスを管理するには、`aws:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値はありです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

の詳細についてはABAC、「 ユーザーガイド」のABAC「 認可によるアクセス許可の定義」を参照してください。 IAM を設定する手順を含むチュートリアルを表示するにはABAC、 IAM ユーザーガイドの「属性ベースのアクセスコントロールを使用する (ABAC) 」を参照してください。

InfluxDB の Timestream での一時的な認証情報の使用

一時的な認証情報のサポート: あり

一時的な認証情報を使用してサインインすると機能 AWS のサービス しない場合があります。一時的な認証情報 AWS のサービス を使用する方法などの詳細については、 IAM ユーザーガイドの AWS のサービス を使用する方法IAMを参照してください。

ユーザー名とパスワード以外の AWS Management Console 方法で にサインインする場合、一時的な認証情報を使用します。例えば、会社のシングルサインオン (SSO) リンク AWS を使用して にアクセスすると、そのプロセスによって一時的な認証情報が自動的に作成されます。また、ユーザーと

してコンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えの詳細については、 IAM ユーザーガイドの「ユーザーからIAMロールへの切り替え (コンソール)」を参照してください。

AWS CLI または を使用して、一時的な認証情報を手動で作成できます AWS API。その後、これらの一時的な認証情報を使用して にアクセスできます AWS。長期的なアクセスキーを使用する代わりに、一時的な認証情報を動的に生成 AWS することをお勧めします。詳細については、「」の「一時的なセキュリティ認証情報IAM」を参照してください。

Timestream for InfluxDB のクロスサービスプリンシパルアクセス許可

転送アクセスセッションをサポート (FAS): はい

IAM ユーザーまたはロールを使用して でアクションを実行すると AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可と AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストのリクエストを使用します。FAS リクエストは、サービスが他の AWS のサービス または リソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するための権限が必要です。FAS リクエストを行う際のポリシーの詳細については、「アクセスセッションの転送」を参照してください。

InfluxDB の Timestream のサービスロール

サービスロールのサポート: なし

サービスロールは、ユーザーに代わってアクションを実行するためにサービスが引き受けるIAMロールです。IAM 管理者は、 内からサービスロールを作成、変更、削除できますIAM。詳細については、IAM「 ユーザーガイド」の「 にアクセス許可を委任するロールの作成 AWS のサービス」を参照してください。

> **Warning**
>
> サービスロールのアクセス許可を変更すると、InfluxDB 機能の Timestream が中断される可能性があります。Timestream for InfluxDB がガイダンスを提供する場合にのみ、サービスロールを編集します。

Timestream for InfluxDB のサービスにリンクされたロール

サービスリンクロールのサポート: あり

サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、 サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールのアクセス許可を表示することはできますが、編集することはできません。

サービスにリンクされたロールの作成または管理の詳細については、AWS 「 と連携するサービス IAM」を参照してください。表の中から、[Service-linked role] (サービスにリンクされたロール) 列に Yes と記載されたサービスを見つけます。サービスリンクロールに関するドキュメントをサービスで表示するには、はい リンクを選択します。

Amazon Timestream for InfluxDB のアイデンティティベースのポリシーの例

デフォルトでは、ユーザーとロールには、InfluxDB リソースの Timestream を作成または変更するアクセス許可がありません。また、、 AWS Command Line Interface （ AWS CLI ） AWS Management Console、または を使用してタスクを実行することはできません AWS API。必要なリソースに対してアクションを実行するアクセス許可をユーザーに付与するには、IAM管理者はIAMポリシーを作成できます。その後、管理者はIAMポリシーをロールに追加し、ユーザーはロールを引き受けることができます。

これらのポリシードキュメント例を使用して IAM ID ベースのJSONポリシーを作成する方法については、IAM「 ユーザーガイド」のIAM「ポリシーの作成 (コンソール) 」を参照してください。

ARNs 各リソースタイプの の形式など、InfluxDB の Timestream で定義されるアクションとリソースタイプの詳細については、「サービス認証リファレンス」の「Amazon Timestream for InfluxDB のアクション、リソース、および条件キー」を参照してください。

トピック

- ポリシーのベストプラクティス
- InfluxDB コンソールでの Timestream の使用
- 自分の権限の表示をユーザーに許可する
- 1 つの Amazon S3 バケットへのアクセス
- すべてのオペレーションを許可する
- DB インスタンスの作成、説明、削除、更新

ポリシーのベストプラクティス

ID ベースのポリシーは、アカウント内の InfluxDB リソースの Timestream を作成、アクセス、または削除できるかどうかを決定します。これらのアクションを実行すると、 AWS アカウントに料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS マネージドポリシーを開始し、最小権限のアクセス許可に移行 – ユーザーとワークロードへのアクセス許可の付与を開始するには、多くの一般的なユースケースのアクセス許可を付与するAWS マネージドポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義することで、アクセス許可をさらに減らすことをお勧めします。詳細については、IAM「 ユーザーガイド」の「 AWS 管理ポリシー」またはAWS 「 ジョブ機能の 管理ポリシー」を参照してください。
- 最小権限のアクセス許可を適用する – IAMポリシーでアクセス許可を設定する場合、タスクの実行に必要なアクセス許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用してアクセス許可を適用する方法の詳細については、IAM「 ユーザーガイド」の「 のポリシーとアクセス許可IAM」を参照してください。
- IAM ポリシーの条件を使用してアクセスをさらに制限する – ポリシーに条件を追加して、アクションとリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを を使用して送信する必要があることを指定できますSSL。また、 などの特定の を通じてサービスアクションが使用されている場合 AWS のサービス、 条件を使用してサービスアクションへのアクセスを許可することもできます AWS CloudFormation。詳細については、「 ユーザーガイド」のIAMJSON「ポリシー要素: 条件」を参照してください。 IAM
- IAM Access Analyzer を使用してIAMポリシーを検証し、安全で機能的なアクセス許可を確保する – IAM Access Analyzer は、ポリシーがポリシー言語 (JSON) とIAMベストプラクティスに準拠するように、新規および既存のIAMポリシーを検証します。IAM Access Analyzer には、安全で機能的なポリシーの作成に役立つ 100 を超えるポリシーチェックと実用的なレコメンデーションが用意されています。詳細については、IAM「 ユーザーガイド」のIAM「Access Analyzer でポリシーを検証する」を参照してください。
- 多要素認証が必要 (MFA) – でIAMユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、 をオンにMFAしてセキュリティを強化します。API オペレーションが呼び出されるMFAタイミングを要求するには、ポリシーにMFA条件を追加します。詳細については、「 ユーザーガイド」の「 によるセキュアAPIアクセスMFA」を参照してください。 IAM

のベストプラクティスの詳細についてはIAM、「 ユーザーガイド」の「 のセキュリティのベストプラクティスIAM」を参照してください。 IAM

InfluxDB コンソールでの Timestream の使用

Amazon Timestream for InfluxDB コンソールにアクセスするには、最小限のアクセス許可のセットが必要です。これらのアクセス許可により、 内の InfluxDB リソースの Timestream の詳細を一覧表示および表示できます AWS アカウント。最小限必要な許可よりも制限が厳しいアイデンティティベースのポリシーを作成すると、そのポリシーを持つエンティティ (ユーザーまたはロール) に対してコンソールが意図したとおりに機能しません。

AWS CLI または のみを呼び出すユーザーに対して、最小限のコンソールアクセス許可を付与する必要はありません AWS API。代わりに、実行しようとしているAPIオペレーションに一致するアクションにのみアクセスを許可します。

ユーザーとロールが引き続き Timestream for InfluxDB コンソールを使用できるようにするには、Timestream for InfluxDB `ConsoleAccess`または `ReadOnly` AWS マネージドポリシーをエンティティにアタッチします。詳細については、「 ユーザーガイド」の「ユーザーへのアクセス許可の追加」を参照してください。 IAM

自分の権限の表示をユーザーに許可する

この例では、IAMユーザーがユーザー ID にアタッチされているインラインポリシーとマネージドポリシーを表示できるようにするポリシーを作成する方法を示します。このポリシーには、コンソールで、または AWS CLI または を使用してプログラムでこのアクションを実行するアクセス許可が含まれています AWS API。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
```

```
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

1 つの Amazon S3 バケットへのアクセス

この例では、 AWS アカウント内のIAMユーザーに Amazon S3 バケットの 1 つへのアクセス権を付与しますexamplebucket。また、ユーザーがオブジェクトを追加、更新、および削除できるようにします。

このポリシーでは、ユーザーに s3:PutObject、s3:GetObject、s3:DeleteObject のアクセス許可を付与するだけでなく、s3:ListAllMyBuckets、s3:GetBucketLocation、および s3:ListBucket のアクセス許可も付与します。これらが、コンソールで必要とされる追加のアクセス許可です。またコンソール内のオブジェクトのコピー、カット、貼り付けを行うためには、s3:PutObjectAcl および s3:GetObjectAcl アクションが必要となります。コンソールを使用して、ユーザーにアクセス許可を付与し、テストする例の解説については、「チュートリアル例: ユーザーポリシーを使用したバケットへのアクセスのコントロール」を参照してください。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ListBucketsInConsole",
         "Effect":"Allow",
         "Action":[
            "s3:ListAllMyBuckets"
         ],
```

```
            "Resource":"arn:aws:s3:::*"
        },
        {
            "Sid":"ViewSpecificBucketInfo",
            "Effect":"Allow",
            "Action":[
                "s3:ListBucket",
                "s3:GetBucketLocation"
            ],
            "Resource":"arn:aws:s3:::examplebucket"
        },
        {
            "Sid":"ManageBucketContents",
            "Effect":"Allow",
            "Action":[
                "s3:PutObject",
                "s3:PutObjectAcl",
                "s3:GetObject",
                "s3:GetObjectAcl",
                "s3:DeleteObject"
            ],
            "Resource":"arn:aws:s3:::examplebucket/*"
        }
    ]
}
```

すべてのオペレーションを許可する

以下は、Timestream for InfluxDB のすべてのオペレーションを許可するサンプルポリシーです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream-influxdb:*"
            ],
            "Resource": "*"
        }
    ]
}
```

DB インスタンスの作成、説明、削除、更新

次のサンプルポリシーでは、ユーザーが DB インスタンス を作成、説明、削除、更新できますsampleDB。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream-influxdb:CreateDbInstance",
                "timestream-influxdb:GetDbInstance",
                "timestream-influxdb:DeleteDbInstance",
                "timestream-influxdb:UpdateDbInstance"
            ],
            "Resource": "arn:aws:timestream-influxdb:us-east-1:<account_ID>:dbinstance/
sampleDB"
        }
    ]
}
```

Amazon Timestream for InfluxDB の ID とアクセスのトラブルシューティング

以下の情報は、InfluxDB および の Timestream を使用する際に発生する可能性のある一般的な問題を診断して修正するのに役立ちますIAM。

トピック

- InfluxDB の Timestream でアクションを実行する権限がありません
- AWS アカウント外のユーザーに Timestream for InfluxDB リソースへのアクセスを許可したい

InfluxDB の Timestream でアクションを実行する権限がありません

がアクションを実行する権限がないと AWS Management Console 指示した場合は、管理者に連絡してサポートを依頼する必要があります。担当の管理者はお客様のユーザー名とパスワードを発行した人です。

以下のエラー例は、mateojackson ユーザーがコンソールを使用して架空の *my-example-widget* リソースに関する詳細情報を表示しようとしているが、架空の timestream-influxdb:*GetWidget* 許可がないという場合に発生します。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 timestream-influxdb:GetWidget on resource: my-example-widget
```

この場合、Mateo は、timestream-influxdb:*GetWidget* アクションを使用して *my-example-widget* リソースへのアクセスが許可されるように、管理者にポリシーの更新を依頼します。

AWS アカウント外のユーザーに Timestream for InfluxDB リソースへのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACLs) をサポートするサービスでは、これらのポリシーを使用して、リソースへのアクセスをユーザーに許可できます。

詳細については、以下を参照してください。

- での DB インスタンスへのアクセスの制御 VPC
- Timestream for InfluxDB がこれらの機能をサポートしているかどうかについては、「Amazon Timestream for InfluxDB が と連携する方法IAM」を参照してください。
- 所有している AWS アカウント間でリソースへのアクセスを提供する方法については、IAM「 ユーザーガイド」の「所有している別の AWS アカウントのIAMユーザーへのアクセスを提供する」を参照してください。
- サードパーティー AWS アカウントにリソースへのアクセスを提供する方法については、 IAM ユーザーガイドの「サードパーティーが所有する AWS アカウントへのアクセスを提供する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、 IAM ユーザーガイドの「外部認証されたユーザーへのアクセスを提供する (ID フェデレーション)」を参照してください。
- クロスアカウントアクセスにロールとリソースベースのポリシーを使用する方法の違いについては、IAM「 ユーザーガイド」のIAM「ロールとリソースベースのポリシーの違い」を参照してください。

での DB インスタンスへのアクセスの制御 VPC

Amazon Virtual Private Cloud (Amazon VPC) を使用すると、Amazon Timestream for InfluxDB DB インスタンスなどの AWS リソースを仮想プライベートクラウド () に起動できますVPC。Amazon を

使用するとVPC、仮想ネットワーク環境を制御できます。独自の IP アドレスの範囲を選択し、サブネットを作成してルーティングおよびアクセス制御リストを設定できます。

VPC セキュリティグループは、 内の DB インスタンスへのアクセスを制御しますVPC。各VPC セキュリティグループルールは、特定のソースVPCがそのVPCセキュリティグループに関連付けられている 内の DB インスタンスにアクセスできるようにします。ソースは、アドレスの範囲 (203.0.113.0/24 など) または別のVPCセキュリティグループです。VPC セキュリティグループをソースとして指定することで、ソースVPCセキュリティグループを使用するすべてのインスタンス (通常はアプリケーションサーバー) からの受信トラフィックを許可します。DB インスタンスへの接続を試みる前に、ユースケースVPC用に を設定します。で DB インスタンスにアクセスするための一般的なシナリオを次に示しますVPC。

内の DB インスタンスは、同じ 内の Amazon EC2インスタンスによってVPCアクセスされます。 VPC

での DB インスタンスの一般的な用途VPCは、同じ のEC2インスタンスで実行されているアプリケーションサーバーとデータを共有することですVPC。EC2 インスタンスは、DB インスタンスとやり取りするアプリケーションを使用してウェブサーバーを実行する場合があります。

の DB インスタンスは、別の のEC2インスタンスによってVPCアクセスされます。 VPC

場合によっては、DB インスタンスは、アクセスに使用するEC2インスタンスVPCとは異なる にあります。その場合は、VPCピアリングを使用して DB インスタンスにアクセスできます。

インターネット経由でクライアントアプリケーションによってVPCアクセスされる の DB インスタンス

インターネット経由でクライアントアプリケーションVPCから の DB インスタンスにアクセスするには、単一のパブリックサブネットVPCを使用して を設定し、パブリックサブネットを使用して DB インスタンスを作成します。また、インターネット経由の通信を有効にするVPCように でインターネットゲートウェイを設定します。の外部から DB インスタンスに接続するには VPC、DB インスタンスにパブリックにアクセスできる必要があります。また、DB インスタンスのセキュリティグループのインバウンドルールを使用してアクセスを許可し、その他の要件を満たしている必要があります。

VPC セキュリティグループの詳細については、Amazon Virtual Private Cloud ユーザーガイド」の「セキュリティグループ」を参照してください。

InfluxDB DB インスタンスの Timestream に接続する方法の詳細については、「」を参照してくださいAmazon Timestream for InfluxDB DB インスタンスへの接続。

セキュリティグループのシナリオ

での DB インスタンスの一般的な用途VPCは、同じ の Amazon EC2インスタンスで実行されているアプリケーションサーバーとデータを共有することです。これはVPC、 の外部にあるクライアントアプリケーションによってアクセスされますVPC。このシナリオでは、InfluxDB の Timestream とのVPCページ、 AWS Management Console または InfluxDB の Timestream と EC2APIオペレーションを使用して、必要なインスタンスとセキュリティグループを作成します。

1. VPC セキュリティグループ (などsg-0123ec2example) を作成し、クライアントアプリケーションの IP アドレスをソースとして使用するインバウンドルールを定義します。このセキュリティグループにより、クライアントアプリケーションVPCは、このセキュリティグループを使用する のEC2インスタンスに接続できます。
2. アプリケーションのEC2インスタンスを作成し、前のステップで作成したVPCセキュリティグループ (sg-0123ec2example) にEC2インスタンスを追加します。
3. 2 番目のVPCセキュリティグループ (などsg-6789rdsexample) を作成し、ステップ 1 (sg-0123ec2example) で作成したVPCセキュリティグループをソースとして指定して、新しいルールを作成します。
4. 新しい DB インスタンスを作成し、前のステップで作成したVPCセキュリティグループ (sg-6789rdsexample) に DB インスタンスを追加します。DB を作成するときは、ステップ 3 で作成したVPCセキュリティグループ (sg-6789rdsexample) ルールで指定されたものと同じポート番号を使用します。

VPC セキュリティグループの作成

VPC コンソールを使用して、DB インスタンスVPCのセキュリティグループを作成できます。セキュリティグループの作成の詳細については、Amazon Virtual Private Cloud ユーザーガイド」の「セキュリティグループ」を参照してください。

セキュリティグループを DB インスタンスと関連付ける

セキュリティグループを DB インスタンスに関連付けるには、InfluxDB コンソールの Timestream の更新、InfluxDB の UpdateDBInstance Timestream API、または update-db-instance AWS CLI コマンドを使用します。

次のCLI例では、特定のVPCセキュリティグループを関連付け、DB インスタンスから DB セキュリティグループを削除します。

```
aws timestream-influxdb update-db-instance --identifier dbName --vpc-security-group-
ids sg-ID
```

DB インスタンスの変更については、「DB インスタンスの更新」を参照してください。

Amazon Timestream for InfluxDB のサービスリンクロールの使用

Amazon Timestream for InfluxDB は AWS Identity and Access Management 、 (IAM) サービスにリンクされたロール を使用します。サービスにリンクされたロールは、Amazon Timestream for InfluxDB などの AWS サービスに直接リンクされる一意のタイプのIAMロールです。Amazon Timestream for InfluxDB サービスにリンクされたロールは、Amazon Timestream for InfluxDB によって事前定義されています。これには、サービスが dbinstances に代わって AWS サービスを呼び出すために必要なすべてのアクセス許可が含まれます。

サービスにリンクされたロールを使用すると、必要なアクセス許可を手動で追加する必要がなくなるため、Amazon Timestream for InfluxDB の設定が簡単になります。ロールは AWS アカウント内に既に存在しますが、Amazon Timestream for InfluxDB のユースケースにリンクされており、事前定義されたアクセス許可があります。これらのロールを引き受けることができるのは InfluxDB 用 Amazon Timestream のみであり、これらのロールのみが事前定義されたアクセス許可ポリシーを使用できます。ロールを削除するには、まず関連リソースを削除します。これにより、Amazon Timestream for InfluxDB リソースにアクセスするために必要なアクセス許可を誤って削除できないため、Amazon Timestream for InfluxDB リソースが保護されます。

サービスにリンクされたロールをサポートする他のサービスの詳細については、「 AWS と連携するサービスIAM」を参照し、「サービスにリンクされたロール」列で「はい」のサービスを探します。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[Yes] (はい) リンクを選択します。

目次

- サービスにリンクされたロールの削除 (IAM コンソール)
- サービスにリンクされたロールの削除 (IAM CLI)
- サービスにリンクされたロールの削除 (IAM API)
- InfluxDB サービスリンクロールの Amazon Timestream でサポートされているリージョン

Amazon Timestream for InfluxDB のサービスにリンクされたロールのアクセス許可

Amazon Timestream for InfluxDB は、 AmazonTimestreamInfluxDBServiceRolePolicy という名前のサービスにリンクされたロールを使用します。このポリシーにより、Timestream for InfluxDB はクラスターの管理に必要な AWS リソースをユーザーに代わって管理できます。

AmazonTimestreamInfluxDBServiceRolePolicy サービスにリンクされたロールのアクセス許可ポリシーにより、Amazon Timestream for InfluxDB は指定されたリソースに対して次のアクションを実行できます。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "DescribeNetworkStatement",
   "Effect": "Allow",
   "Action": [
    "ec2:DescribeSubnets",
    "ec2:DescribeVpcs",
    "ec2:DescribeNetworkInterfaces"
   ],
   "Resource": "*"
  },
  {
   "Sid": "CreateEniInSubnetStatement",
   "Effect": "Allow",
   "Action": [
    "ec2:CreateNetworkInterface"
   ],
   "Resource": [
    "arn:aws:ec2:*:*:subnet/*",
    "arn:aws:ec2:*:*:security-group/*"
   ]
  },
  {
   "Sid": "CreateEniStatement",
```

```
    "Effect": "Allow",
    "Action": [
     "ec2:CreateNetworkInterface"
    ],
    "Resource": "arn:aws:ec2:*:*:network-interface/*",
    "Condition": {
     "Null": {
      "aws:RequestTag/AmazonTimestreamInfluxDBManaged": "false"
     }
    }
   },
   {
    "Sid": "CreateTagWithEniStatement",
    "Effect": "Allow",
    "Action": [
     "ec2:CreateTags"
    ],
    "Resource": "arn:aws:ec2:*:*:network-interface/*",
    "Condition": {
     "Null": {
      "aws:RequestTag/AmazonTimestreamInfluxDBManaged": "false"
     },
     "StringEquals": {
      "ec2:CreateAction": [
       "CreateNetworkInterface"
      ]
     }
    }
   },
   {
    "Sid": "ManageEniStatement",
    "Effect": "Allow",
    "Action": [
     "ec2:CreateNetworkInterfacePermission",
     "ec2:DeleteNetworkInterface"
    ],
    "Resource": "arn:aws:ec2:*:*:network-interface/*",
    "Condition": {
     "Null": {
      "aws:ResourceTag/AmazonTimestreamInfluxDBManaged": "false"
     }
    }
   },
   {
```

```
    "Sid": "PutCloudWatchMetricsStatement",
    "Effect": "Allow",
    "Action": [
     "cloudwatch:PutMetricData"
    ],
    "Condition": {
     "StringEquals": {
      "cloudwatch:namespace": [
       "AWS/Timestream/InfluxDB",
       "AWS/Usage"
      ]
     }
    },
    "Resource": [
     "*"
    ]
   },
   {
    "Sid": "ManageSecretStatement",
    "Effect": "Allow",
    "Action": [
     "secretsmanager:CreateSecret",
     "secretsmanager:DeleteSecret"
    ],
    "Resource": [
     "arn:aws:secretsmanager:*:*:secret:READONLY-InfluxDB-auth-parameters-*"
    ],
    "Condition": {
     "StringEquals": {
      "aws:ResourceAccount": "${aws:PrincipalAccount}"
     }
    }
   }
  ]
}
```

IAMエンティティがサービスにリンクされたロールを作成AmazonTimestreamInfluxDBServiceRolePolicyできるようにするには

次のポリシーステートメントをそのIAMエンティティのアクセス許可に追加します。

```
{
    "Effect": "Allow",
```

```
    "Action": [
        "iam:CreateServiceLinkedRole",
        "iam:PutRolePolicy"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/
timestreamforinfluxdb.amazonaws.com/AmazonTimestreamInfluxDBServiceRolePolicy*",
    "Condition": {"StringLike": {"iam:AWS ServiceName":
 "timestreamforinfluxdb.amazonaws.com"}}
}
```

IAMエンティティがサービスにリンクされたロールを削除AmazonTimestreamInfluxDBServiceRolePolicyできるようにするには

次のポリシーステートメントをそのIAMエンティティのアクセス許可に追加します。

```
{
    "Effect": "Allow",
    "Action": [
        "iam:DeleteServiceLinkedRole",
        "iam:GetServiceLinkedRoleDeletionStatus"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/
timestreamforinfluxdb.amazonaws.com/AmazonTimestreamInfluxDBServiceRolePolicy*",
    "Condition": {"StringLike": {"iam:AWS ServiceName":
 "timestreamforinfluxdb.amazonaws.com"}}
}
```

または、 AWS マネージドポリシーを使用して、Amazon Timestream for InfluxDB へのフルアクセスを提供することもできます。

サービスにリンクされたロールの作成 (IAM)

サービスリンクロールを手動で作成する必要はありません。DB インスタンスを作成すると、Amazon Timestream for InfluxDB がサービスにリンクされたロールを作成します。

このサービスリンクロールを削除した後で再度作成する必要が生じた場合は、同じ方法でアカウントにロールを再作成できます。DB インスタンスを作成すると、Amazon Timestream for InfluxDB はサービスにリンクされたロールを再度作成します。

Amazon Timestream for InfluxDB のサービスリンクロールの説明の編集

Amazon Timestream for InfluxDB では、サービスにリンクされたロールを編集AmazonTimestreamInfluxDBServiceRolePolicyすることはできません。サービスリンクロールを作成

した後は、多くのエンティティによってロールが参照される可能性があるため、ロール名を変更することはできません。ただし、 を使用してロールの説明を編集できますIAM。

サービスにリンクされたロールの説明の編集 (IAM コンソール)

IAM コンソールを使用して、サービスにリンクされたロールの説明を編集できます。

サービスにリンクされたロールの説明を編集するには (コンソール)

1. IAM コンソールの左側のナビゲーションペインで、ロール を選択します。
2. 変更するロールの名前を選択します。
3. ロールの説明の右端にある編集を選択します。
4. ボックスに新しい説明を入力し、保存を選択します。

サービスにリンクされたロールの説明の編集 (IAM CLI)

からIAMオペレーションを使用して AWS Command Line Interface 、サービスにリンクされたロールの説明を編集できます。

サービスにリンクされたロールの説明を変更するには (CLI)

1. (オプション) ロールの現在の説明を表示するには、 IAMオペレーション AWS CLI に を使用しますget-role。

   Example

   ```
   $ aws iam get-role --role-name AmazonTimestreamInfluxDBServiceRolePolicy
   ```

   CLI オペレーションでロールを参照するにはARN、 ではなくロール名を使用します。例えば、ロールにARN次の がある場合arn:aws:iam::123456789012:role/myrole、ロールを と参照しますmyrole。
2. サービスにリンクされたロールの説明を更新するには、 IAMオペレーション AWS CLI に を使用しますupdate-role-description。

   Linux と MacOS

   ```
   $ aws iam update-role-description \
       --role-name AmazonTimestreamInfluxDBServiceRolePolicy \
       --description "new description"
   ```

Windows

```
$ aws iam update-role-description ^
    --role-name AmazonTimestreamInfluxDBServiceRolePolicy ^
    --description "new description"
```

サービスにリンクされたロールの説明の編集 (IAM API)

を使用してIAMAPI、サービスにリンクされたロールの説明を編集できます。

サービスにリンクされたロールの説明を変更するには (API)

1. (オプション) ロールの現在の説明を表示するには、 IAMAPIオペレーションを使用します。GetRole.

   Example

   ```
   https://iam.amazonaws.com/
      ?Action=GetRole
      &RoleName=AmazonTimestreamInfluxDBServiceRolePolicy
      &Version=2010-05-08
      &AUTHPARAMS
   ```

2. ロールの説明を更新するには、 IAMAPIオペレーションを使用します。UpdateRoleDescription.

   Example

   ```
   https://iam.amazonaws.com/
      ?Action=UpdateRoleDescription
      &RoleName=AmazonTimestreamInfluxDBServiceRolePolicy
      &Version=2010-05-08
      &Description="New description"
   ```

Amazon Timestream for InfluxDB のサービスリンクロールの削除

サービスリンクロールが必要な機能またはサービスが不要になった場合には、そのロールを削除することをお勧めします。そうすることで、モニタリングや保守が積極的に行われていない未使用のエンティティを排除できます。ただし、削除する前に、サービスにリンクされた役割をクリーンアップする必要があります。

Amazon Timestream for InfluxDB は、サービスにリンクされたロールを削除しません。

サービスにリンクされたロールのクリーンアップ

IAM を使用してサービスにリンクされたロールを削除する前に、まずロールにリソース (クラスター) が関連付けられていないことを確認します。

サービスにリンクされたロールにIAMコンソールでアクティブなセッションがあるかどうかを確認するには

1. にサインイン AWS Management Console し、 でIAMコンソールを開きますhttps://console.aws.amazon.com/iam/。
2. IAM コンソールの左側のナビゲーションペインで、ロール を選択します。次に、ロールの名前 (チェックボックスではない) AmazonTimestreamInfluxDBServiceRolePolicyを選択します。
3. 選択したロールの 概要 ページで、アクセスアドバイザー タブを選択します。
4. アクセスアドバイザー タブで、サービスにリンクされたロールの最新のアクティビティを確認します。

サービスにリンクされたロールの削除 (IAM コンソール)

IAM コンソールを使用して、サービスにリンクされたロールを削除できます。

サービスにリンクされたロールを削除するには (コンソール)

1. にサインイン AWS Management Console し、 でIAMコンソールを開きますhttps://console.aws.amazon.com/iam/。
2. IAM コンソールの左側のナビゲーションペインで、ロール を選択します。ロール名または行そのものではなく、削除するロール名の横にあるチェックボックスをオンにします。
3. ページ上部にある ロールのアクション で ロールの削除 を選択します。
4. 確認ページで、最後にアクセスしたサービスデータを確認します。このデータには、選択した各ロールが最後に AWS サービスにアクセスした日時が表示されます。これは、そのロールが現在アクティブであるかどうかを確認するのに役立ちます。先に進む場合は、Yes, Delete] (はい、削除する) を選択し、削除するサービスにリンクされたロールを送信します。
5. IAM コンソールの通知を見て、サービスにリンクされたロールの削除の進行状況をモニタリングします。IAM サービスにリンクされたロールの削除は非同期であるため、削除のためにロールを送信すると、削除タスクが成功または失敗する可能性があります。タスクが失敗した場合

は、通知から View details] (詳細を表示) または View Resources] (リソースを表示) を選択して、削除が失敗した理由を知ることができます。

サービスにリンクされたロールの削除 (IAM CLI)

から IAM オペレーションを使用して AWS Command Line Interface 、サービスにリンクされたロールを削除できます。

サービスにリンクされたロールを削除するには (CLI)

1. 削除するサービスにリンクされたロールの名前が分からない場合、以下のコマンドを入力します。このコマンドは、アカウントのロールとその Amazon リソースネーム (ARNs) を一覧表示します。

   ```
   $ aws iam get-role --role-name role-name
   ```

   CLI オペレーションでロールを参照するにはARN、 ではなくロール名を使用します。例えば、ロールに がある場合ARN`arn:aws:iam::123456789012:role/myrole`、ロールを と呼びます**myrole**。

2. サービスにリンクされているロールは、使用されている、または関連するリソースがある場合は削除できないため、削除リクエストを送信する必要があります。これらの条件が満たされない場合、そのリクエストは拒否される可能性があります。レスポンスから `deletion-task-id` を取得して、削除タスクのステータスを確認する必要があります。サービスにリンクされたロールの削除リクエストを送信するには、以下を入力します。

   ```
   $ aws iam delete-service-linked-role --role-name role-name
   ```

3. 削除タスクのステータスを確認するには、以下を入力します。

   ```
   $ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
   ```

   削除タスクのステータスは、`NOT_STARTED`、`IN_PROGRESS`、`SUCCEEDED`、または `FAILED` となります。削除が失敗した場合は、失敗した理由がコールによって返され、トラブルシューティングが可能になります。

サービスにリンクされたロールの削除 (IAM API)

を使用してIAMAPI、サービスにリンクされたロールを削除できます。

サービスにリンクされたロールを削除するには (API)

1. サービスにリンクされたロールの削除リクエストを送信するには、 を呼び出します。DeleteServiceLinkedRole。 リクエストで、ロール名を指定します。

   サービスにリンクされているロールは、使用されている、または関連するリソースがある場合は削除できないため、削除リクエストを送信する必要があります。これらの条件が満たされない場合、そのリクエストは拒否される可能性があります。レスポンスから DeletionTaskId を取得して、削除タスクのステータスを確認する必要があります。
2. 削除のステータスを確認するには、 を呼び出します。GetServiceLinkedRoleDeletionStatus。 リクエストで、 を指定しますDeletionTaskId。

   削除タスクのステータスは、NOT_STARTED、IN_PROGRESS、SUCCEEDED、または FAILED となります。削除が失敗した場合は、失敗した理由がコールによって返され、トラブルシューティングが可能になります。

InfluxDB サービスリンクロールの Amazon Timestream でサポートされているリージョン

Amazon Timestream for InfluxDB は、サービスが利用可能なすべてのリージョンでサービスにリンクされたロールの使用をサポートしています。詳細については、AWS サービスエンドポイントを参照してください。

AWS Amazon Timestream for InfluxDB の マネージドポリシー

ユーザー、グループ、ロールにアクセス許可を追加するには、自分でポリシーを記述するよりも、AWS 管理ポリシーを使用する方が簡単です。必要なアクセス許可のみをチームに提供するIAMカスタマー管理ポリシーを作成するには、時間と専門知識が必要です。すぐに開始するには、 AWS マネージドポリシーを使用できます。これらのポリシーは一般的なユースケースを対象としており、 AWS アカウントで利用できます。 AWS 管理ポリシーの詳細については、「 ユーザーガイド」のAWS 「 管理ポリシー」を参照してください。 IAM

AWS サービスは、 AWS マネージドポリシーを維持および更新します。 AWS マネージドポリシーのアクセス許可は変更できません。サービスでは、新しい機能を利用できるようにするために、

AWS マネージドポリシーに権限が追加されることがあります。この種類の更新は、ポリシーがアタッチされている、すべてのアイデンティティ (ユーザー、グループおよびロール) に影響を与えます。新しい機能が立ち上げられた場合や、新しいオペレーションが使用可能になった場合に、各サービスが AWS マネージドポリシーを更新する可能性が最も高くなります。サービスは AWS 管理ポリシーからアクセス許可を削除しないため、ポリシーの更新によって既存のアクセス許可が損なわれることはありません。

さらに、 は、複数の サービスにまたがるジョブ関数の管理ポリシー AWS をサポートしています。例えば、 ReadOnlyAccess AWS マネージドポリシーは、すべての AWS サービスとリソースへの読み取り専用アクセスを提供します。サービスが新機能を起動すると、 は新しいオペレーションとリソースの読み取り専用アクセス許可 AWS を追加します。ジョブ関数ポリシーのリストと説明については、「 ユーザーガイド」のAWS 「ジョブ関数のマネージドポリシー」を参照してください。 IAM

AWS マネージドポリシー： AmazonTimestreamInfluxDBServiceRolePolicy

アカウントの ID に AmazonTimestreamInfluxDBServiceRolePolicy AWS マネージドポリシーをアタッチすることはできません。このポリシーは、 AWS TimestreamforInfluxDB サービスにリンクされたロールの一部です。このロールにより、サービスはアカウント内のネットワークインターフェイスとセキュリティグループを管理できます。

InfluxDB の Timestream は、このポリシーのアクセス許可を使用して、EC2セキュリティグループとネットワークインターフェイスを管理します。これは、InfluxDB DB インスタンスの Timestream を管理するために必要です。

このポリシーを JSON 形式で確認するには、「」を参照してくださいAmazonTimestreamInfluxDBServiceRolePolicy。

AWS Amazon Timestream for InfluxDB の マネージドポリシー

AWS は、 によって作成および管理されるスタンドアロンIAMポリシーを提供することで、多くの一般的なユースケースに対処します AWS。マネージドポリシーは、一般的ユースケースに必要な許可を付与することで、どの許可が必要なのかをユーザーが調査する必要をなくすることができます。詳細については、「 ユーザーガイド」のAWS 「 マネージドポリシー」を参照してください。 IAM

アカウント内のユーザーにアタッチできる次の AWS 管理ポリシーは、Timestream for InfluxDB に固有です。

AmazonTimestreamInfluxDBFullAccess

ID IAM にAmazonTimestreamInfluxDBFullAccessポリシーをアタッチできます。このポリシーは、InfluxDB のすべての Timestream リソースへのフルアクセスを許可する管理アクセス許可を付与します。

また、独自のカスタムIAMポリシーを作成して、Amazon Timestream for InfluxDB APIアクションのアクセス許可を許可することもできます。これらのカスタムポリシーは、これらのアクセス許可を必要とするIAMユーザーまたはグループにアタッチできます。

このポリシーを JSON 形式で確認するには、「」を参照してくださいAmazonTimestreamInfluxDBFullAccess。

AWS マネージドポリシーへの InfluxDB 更新のタイムストリーム

このサービスがこれらの変更の追跡を開始してからの、Timestream for InfluxDB の AWS マネージドポリシーの更新に関する詳細を表示します。このページの変更に関する自動アラートについては、InfluxDB ドキュメントの Timestream 履歴ページのRSSフィードにサブスクライブします。

変更	説明	日付
AmazonTimestreamInfluxDBFullAccess - 既存ポリシーへの更新	既存の AmazonTimestreamInfluxDBFullAccess マネージドポリシーに ec2:DescribeRouteTables アクションを追加しました。このアクションは、ルートテーブルの説明に使用されます。	10/08/2024
AWS マネージドポリシー：AmazonTimestreamInfluxDBServiceRolePolicy - 新しいポリシー	Amazon Timestream for InfluxDB に、サービスがアカウント内のネットワークインターフェイスとセキュリティ	03/14/2024

変更	説明	日付
	グループを管理できるようにする新しいポリシーが追加されました。	
AmazonTimestreamInfluxDBFullAccess - 新しいポリシー	Amazon Timestream for InfluxDB は、Amazon Timestream InfluxDB インスタンスを作成、更新、削除、一覧表示し、パラメータグループを作成、一覧表示するための完全な管理アクセスを提供する新しいポリシーを追加しました。	03/14/2024

VPC エンドポイントを介した InfluxDB の Timestream への接続

仮想プライベートクラウド () のプライベートインターフェイスエンドポイントを介して、Timestream for InfluxDB に直接接続できますVPC。インターフェイスVPCエンドポイントを使用する場合、 VPCと Timestream for InfluxDB 間の通信は AWS 、ネットワーク内で完全に実行されます。

InfluxDB の Timestream は、 を搭載した Amazon Virtual Private Cloud (Amazon VPC) エンドポイントをサポートしていますAWS PrivateLink。各VPCエンドポイントは、VPCサブネット内のプライベート IP アドレスを持つ 1 つ以上の Elastic Network Interface (ENIs) によって表されます。

インターフェイスVPCエンドポイントは、インターネットゲートウェイ、NATデバイス、VPN接続、または AWS Direct Connect 接続なしで、 を Timestream for InfluxDB VPCに直接接続します。のインスタンスは、Timestream for InfluxDB と通信するためにパブリック IP アドレスを必要としVPCません。

リージョン

Timestream for InfluxDB は、Timestream for InfluxDB がサポートされているすべての AWS リージョン でVPCエンドポイントとVPCエンドポイントポリシーをサポートします。

トピック

- InfluxDB VPCエンドポイントの Timestream に関する考慮事項

- InfluxDB 用の Timestream のVPCエンドポイントの作成
- InfluxDB VPCエンドポイントの Timestream への接続
- VPC エンドポイントへのアクセスの制御
- ポリシーステートメントでのVPCエンドポイントの使用
- VPC エンドポイントのログ記録

InfluxDB VPCエンドポイントの Timestream に関する考慮事項

Timestream for InfluxDB のインターフェイスVPCエンドポイントを設定する前に、 AWS PrivateLink ガイドの「インターフェイスエンドポイントのプロパティと制限」トピックを確認してください。

VPC エンドポイントの InfluxDB サポートのタイムストリームには、以下が含まれます。

- VPC エンドポイントを使用して、 からすべての Timestream for InfluxDB APIオペレーションを呼び出すことができますVPC。
- AWS CloudTrail ログを使用して、VPCエンドポイントを介した Timestream for InfluxDB リソースの使用を監査できます。詳細については、「VPC エンドポイントのログ記録」を参照してください。

InfluxDB 用の Timestream のVPCエンドポイントの作成

Amazon VPCコンソールまたは Amazon を使用して、Timestream for InfluxDB VPC のVPCエンドポイントを作成できますAPI。詳細については、「AWS PrivateLink ガイド」の「インターフェイスエンドポイントを作成」を参照してください。

- InfluxDB の Timestream のVPCエンドポイントを作成するには、次のサービス名を使用します。

```
com.amazonaws.region.timestream-influxdb
```

例えば、米国西部 (オレゴン) リージョン (us-west-2) では、サービス名は次のようになります。

```
com.amazonaws.us-west-2.timestream-influxdb
```

VPC エンドポイントの使用を容易にするために、VPCエンドポイントのプライベートDNS名を有効にできます。DNS 名前を有効にする オプションを選択すると、InfluxDB ホスト名の標準

の Timestream がVPCエンドポイントに解決されます。 DNS例えば、 `https://timestream-influxdb.us-west-2.amazonaws.com`はサービス名 に接続されたVPCエンドポイントに解決されます`com.amazonaws.us-west-2.timestream-influxdb`。

このオプションを使用すると、VPCエンドポイントの使用が容易になります。 AWS SDKs と は、デフォルトで InfluxDB の標準 Timestream DNSホスト名 AWS CLI を使用するため、アプリケーションやコマンドURLでVPCエンドポイントを指定する必要はありません。

詳細については、「AWS PrivateLink ガイド」の「インターフェイスエンドポイントを介したサービスへアクセスする」を参照してください。

InfluxDB VPCエンドポイントの Timestream への接続

AWS SDK、、 AWS CLI または を使用して、VPCエンドポイントを介して InfluxDB の Timestream に接続できます AWS Tools for PowerShell。VPC エンドポイントを指定するには、そのDNS名前を使用します。

VPC エンドポイントの作成時にプライベートホスト名を有効にした場合、CLIコマンドやアプリケーション設定URLでVPCエンドポイントを指定する必要はありません。InfluxDB ホスト名の標準の Timestream はVPCエンドポイントに解決されます。 DNS AWS CLI と はデフォルトでこのホスト名 SDKsを使用するため、 VPC エンドポイントを使用して、スクリプトとアプリケーションを変更することなく、InfluxDB リージョンエンドポイントの Timestream に接続できます。

プライベートホスト名を使用するには、 の `enableDnsHostnames`および `enableDnsSupport` 属性を に設定VPCする必要があります`true`。これらの属性を設定するには、 ModifyVpcAttributeオペレーションを使用します。詳細については、「Amazon VPCユーザーガイド」の「 のDNS属性の表示と更新VPC」を参照してください。

VPC エンドポイントへのアクセスの制御

Timestream for InfluxDB のVPCエンドポイントへのアクセスを制御するには、VPCエンドポイントポリシーをVPCエンドポイントにアタッチします。エンドポイントポリシーは、プリンシパルがVPCエンドポイントを使用して、InfluxDB リソースの Timestream で InfluxDB オペレーションの Timestream を呼び出すことができるかどうかを決定します。

VPC エンドポイントポリシーは、エンドポイントの作成時に作成でき、VPCエンドポイントポリシーはいつでも変更できます。VPC 管理コンソール、または CreateVpcEndpoint または ModifyVpcEndpointオペレーションを使用します。AWS CloudFormation テンプレート を使用してVPCエンドポイントポリシーを作成および変更することもできます。VPC 管理コンソールの使用

については、「 AWS PrivateLink ガイド」の「インターフェイスエンドポイントの作成」と「インターフェイスエンドポイントの変更」を参照してください。

> **Note**
>
> InfluxDB の Timestream は、2020 年 7 月以降のVPCエンドポイントポリシーをサポートしています。VPC その日付より前に作成された InfluxDB の Timestream のエンドポイントにはデフォルトのVPCエンドポイントポリシー がありますが、いつでも変更できます。

トピック

- VPC エンドポイントポリシーについて
- デフォルトのVPCエンドポイントポリシー
- VPC エンドポイントポリシーの作成
- VPC エンドポイントポリシーの表示

VPC エンドポイントポリシーについて

VPC エンドポイントを使用して InfluxDB の Timestream リクエストを成功させるには、プリンシパルに 2 つのソースからのアクセス許可が必要です。

- IAM ポリシーは、 リソースで オペレーションを呼び出すアクセス許可をプリンシパルに付与する必要があります。
- VPC エンドポイントポリシーは、エンドポイントを使用してリクエストを行うアクセス許可をプリンシパルに付与する必要があります。

デフォルトのVPCエンドポイントポリシー

すべてのVPCエンドポイントにはVPCエンドポイントポリシーがありますが、ポリシーを指定する必要はありません。ポリシーを指定しない場合、デフォルトのエンドポイントポリシーでは、エンドポイント上のすべてのリソースのすべてのプリンシパルによるすべてのオペレーションが許可されます。

ただし、Timestream for InfluxDB リソースの場合、プリンシパルにはIAMポリシーから オペレーションを呼び出すアクセス許可も必要です。したがって、実際には、デフォルトのポリシーでは、プリンシパルにリソースに対して オペレーションを呼び出すアクセス許可がある場合は、エンドポイントを使用して呼び出すこともできます。

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Principal": "*",
      "Resource": "*"
    }
  ]
}
```

プリンシパルが許可されたオペレーションのサブセットにのみVPCエンドポイントを使用できるようにするには、VPCエンドポイントポリシー を作成または更新します。

VPC エンドポイントポリシーの作成

VPC エンドポイントポリシーは、プリンシパルがVPCエンドポイントを使用してリソースに対してオペレーションを実行するアクセス許可を持っているかどうかを決定します。Timestream for InfluxDB リソースの場合、プリンシパルにはIAMポリシー からオペレーションを実行するアクセス許可も必要です。

各VPCエンドポイントポリシーステートメントには、次の要素が必要です。

- アクションを実行できるプリンシパル
- 実行可能なアクション
- アクションを実行できるリソース

ポリシーステートメントはVPCエンドポイントを指定しません。代わりに、ポリシーがアタッチされているすべてのVPCエンドポイントに適用されます。詳細については、「Amazon VPCユーザーガイド」のVPC「エンドポイントを使用したサービスへのアクセスの制御」を参照してください。

AWS CloudTrail は、 VPCエンドポイントを使用するすべてのオペレーションを記録します。

VPC エンドポイントポリシーの表示

VPC エンドポイントのエンドポイントポリシーを表示するには、 VPC 管理コンソールまたはDescribeVpcEndpointsオペレーションを使用します。

次の AWS CLI コマンドは、指定されたエンドポイント ID を持つVPCエンドポイントのポリシーを取得します。

このコマンドを使用する前に、サンプルのエンドポイント ID をアカウントの有効なものに置き換えてください。

```
$ aws ec2 describe-vpc-endpoints \
--query 'VpcEndpoints[?VpcEndpointId==`vpc-endpoint-id`].[PolicyDocument]'
--output text
```

ポリシーステートメントでのVPCエンドポイントの使用

リクエストがエンドポイントから送信された場合、VPCまたはVPCエンドポイントを使用した場合に、Timestream for InfluxDB リソースとオペレーションへのアクセスを制御できます。これを行うには、IAMポリシー で次のいずれかのグローバル条件キーを使用します。

- aws:sourceVpce 条件キーを使用して、VPCエンドポイントに基づいてアクセスを許可または制限します。
- aws:sourceVpc 条件キーを使用して、プライベートエンドポイントをホストVPCする に基づいてアクセスを許可または制限します。

> **Note**
>
> VPC エンドポイントに基づいてキーポリシーとIAMポリシーを作成するときは注意してください。ポリシーステートメントでリクエストが特定の VPCまたはVPCエンドポイントから送信されることが要求されている場合、ユーザーに代わって Timestream for InfluxDB リソースを使用する統合 AWS サービスからのリクエストが失敗する可能性があります。
> また、リクエストが Amazon VPCエンドポイント から送信された場合、aws:sourceIP条件キーは有効ではありません。VPC エンドポイントへのリクエストを制限するには、aws:sourceVpceまたは aws:sourceVpc条件キーを使用します。詳細については、「 ガイド」のVPC「エンドポイントおよびVPCエンドポイントサービスのアイデンティティとアクセスの管理」を参照してください。 AWS PrivateLink

これらのグローバル条件キーを使用して、特定のリソースに依存しない CreateDbInstance などのオペレーションへのアクセスを制御できます。

VPC エンドポイントのログ記録

AWS CloudTrail は、 VPCエンドポイントを使用するすべてのオペレーションを記録します。InfluxDB の Timestream へのリクエストがVPCエンドポイントを使用する場合、VPCエンドポイ

ント ID はリクエストを記録するAWS CloudTrail ログエントリに表示されます。エンドポイント ID を使用して、InfluxDB エンドポイントの Timestream の使用を監査できます。 VPC

ただし、 CloudTrail ログには、他のアカウントのプリンシパルによってリクエストされたオペレーションや、他のアカウントの Timestream for InfluxDB リソースとエイリアスに対する Timestream for InfluxDB オペレーションのリクエストは含まれません。また、 を保護するためにVPC、VPCエンドポイントポリシー によって拒否されたが、それ以外の場合は許可されていたリクエストは に記録されませんAWS CloudTrail。

InfluxDB の Timestream でのログ記録とモニタリング

モニタリングは、Timestream for InfluxDB と AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。マルチポイント障害が発生した場合に簡単にデバッグできるように、 AWS ソリューションのすべての部分からモニタリングデータを収集する必要があります。ただし、InfluxDB の Timestream のモニタリングを開始する前に、以下の質問に対する回答を含むモニタリング計画を作成する必要があります。

- モニタリングの目的は何ですか?
- どのリソースをモニタリングしますか?
- どのくらいの頻度でこれらのリソースをモニタリングしますか?
- どのモニタリングツールを利用しますか?
- 誰がモニタリングタスクを実行しますか?
- 問題が発生したときに誰が通知を受け取りますか?

次のステップでは、さまざまな時間およびさまざまな負荷条件でパフォーマンスを測定することで、環境内の InfluxDB の通常の Timestream パフォーマンスのベースラインを確立します。InfluxDB の Timestream をモニタリングするときは、履歴モニタリングデータを保存して、現在のパフォーマンスデータと比較し、通常のパフォーマンスパターンとパフォーマンス異常を特定し、問題に対処する方法を考案できるようにします。

ベースラインを確立するには、少なくとも次の項目をモニタリングする必要があります。

- システムエラー。リクエストでエラーが発生したかどうかを判断できます。

トピック

- モニタリングツール
- を使用した InfluxDB API呼び出しの Timestream のログ記録 AWS CloudTrail

モニタリングツール

AWS には、InfluxDB の Timestream のモニタリングに使用できるさまざまなツールが用意されています。これらのツールの一部はモニタリングを行うように設定できますが、一部のツールは手動による介入が必要です。モニタリングタスクをできるだけ自動化することをお勧めします。

トピック

自動モニタリングツール

次の自動モニタリングツールを使用して、InfluxDB の Timestream をモニタリングし、問題が発生したときに報告できます。

- Amazon CloudWatch アラーム – 指定した期間に 1 つのメトリクスを監視し、複数の期間にわたって特定のしきい値に対するメトリクスの値に基づいて 1 つ以上のアクションを実行します。アクションは、Amazon Simple Notification Service (Amazon SNS) トピックまたは Amazon EC2 Auto Scaling ポリシーに送信される通知です。 CloudWatch アラームは、特定の状態にあるという理由だけでアクションを呼び出すわけではありません。状態は、指定された期間変更され、維持されている必要があります。詳細については、「Amazon によるモニタリング CloudWatch」を参照してください。

手動モニタリングツール

InfluxDB の Timestream をモニタリングするもう 1 つの重要な部分には、 CloudWatch アラームでカバーされない項目を手動でモニタリングすることが含まれます。InfluxDB の Timestream 、CloudWatch Trusted Advisor、およびその他の AWS Management Console ダッシュボードには、at-a-glance AWS 環境の状態が表示されます。

- CloudWatch ホームページには、以下が表示されます。
  - 現在のアラームとステータス
  - アラームとリソースのグラフ
  - サービスのヘルスステータス

  さらに、 CloudWatch を使用して以下を実行できます。
  - 重視するサービスをモニタリングするためのカスタマイズしたダッシュボードを作成します

- メトリクスデータをグラフ化して、問題のトラブルシューティングを行い、傾向を確認する
- すべての AWS リソースメトリクスを検索して参照する
- 問題があることを通知するアラームを作成/編集する

を使用した InfluxDB API呼び出しの Timestream のログ記録 AWS CloudTrail

Timestream for InfluxDB は AWS CloudTrail、ユーザー、ロール、または Timestream for InfluxDB のサービスによって実行されたアクションの記録を提供する AWS サービス と統合されています。は、Timestream for InfluxDB をイベントとしてAPI呼び出すデータ定義言語 (DDL) CloudTrail を取得します。 InfluxDB キャプチャされる呼び出しには、InfluxDB コンソールの Timestream からの呼び出しと、InfluxDB オペレーションの Timestream へのコード呼び出しが含まれます。 API証跡を作成する場合、InfluxDB の CloudTrail Timestream のイベントなど、Amazon Simple Storage Service (Amazon S3) バケットへのイベントの継続的な配信を有効にすることができます。 InfluxDB 証跡を設定しない場合でも、最新のイベントは イベント履歴 の CloudTrail コンソールで表示できます。によって収集された情報を使用して CloudTrail、InfluxDB の Timestream に対して行われたリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時、その他の詳細を確認できます。

の詳細については CloudTrail、AWS CloudTrail 「 ユーザーガイド」を参照してください。

での InfluxDB 情報のタイムストリーム CloudTrail

CloudTrail AWS アカウントを作成すると、 はアカウントで有効になります。InfluxDB の Timestream でアクティビティが発生すると、そのアクティビティは CloudTrail イベント履歴の他の AWS サービスイベントとともにイベントに記録されます。 AWS アカウントで最近のイベントを表示、検索、ダウンロードできます。詳細については、「イベント履歴 での CloudTrail イベントの表示」を参照してください。

Timestream for InfluxDB のイベントなど、 AWS アカウント内のイベントの継続的な記録については、証跡を作成します。証跡により CloudTrail 、 はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成すると、証跡はすべてのAWSリージョンに適用されます。証跡は、 AWS パーティション内のすべてのリージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、 CloudTrail ログで収集されたイベントデータをさらに分析して対処するように、他の AWS サービスを設定できます。

詳細については、『AWS CloudTrail ユーザーガイド:』の以下のトピックを参照してください。

- 追跡作成の概要

- CloudTrail サポートされているサービスと統合
- の Amazon SNS Notifications の設定 CloudTrail
- 複数のリージョンからの CloudTrail ログファイルの受信
- 複数のアカウントからの CloudTrail ログファイルの受信
- データイベントのログ記録

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。アイデンティティ情報は、以下を判別するのに役立ちます:

- リクエストがルートまたは AWS Identity and Access Management （ IAM) ユーザー認証情報で行われたかどうか
- リクエストが、ロールとフェデレーティッドユーザーのどちらの一時的なセキュリティ認証情報を使用して送信されたか
- リクエストが別の AWS サービスによって行われたかどうか

詳細については、CloudTrail userIdentity「 要素」を参照してください。

Amazon Timestream for InfluxDB のコンプライアンス検証

サードパーティーの監査者は、複数のコンプライアンスプログラムの一環として、Amazon Timestream for InfluxDB のセキュリティと AWS コンプライアンスを評価します。これには以下が含まれます。

- GDPR
- HIPAA
- PCI
- SOC

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、AWS のサービス「コンプライアンスプログラムによるスコープ」の」の「」を参照し、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS「コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードすることができます AWS Artifact。詳細については、「」の AWS Artifact」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、データの機密性、会社のコンプライアンス目的、および適用される法律と規制によって決まります。 は、コンプライアンスに役立つ以下のリソース AWS を提供します。

- セキュリティとコンプライアンスのクイックスタートガイド – これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境 AWS を にデプロイする手順について説明します。
- Amazon Web Services HIPAA のセキュリティとコンプライアンスのためのアーキテクチャ – このホワイトペーパーでは、企業が AWS を使用して HIPAA対象アプリケーションを作成する方法について説明します。

> **Note**
>
> すべての AWS のサービス がHIPAA対象とは限りません。詳細については、HIPAA「対象サービスリファレンス」を参照してください。

- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界とロケーションに適用される場合があります。
- AWS カスタマーコンプライアンスガイド – コンプライアンスの観点から責任共有モデルを理解します。このガイドでは、ガイダンスを保護し AWS のサービス 、複数のフレームワーク (米国国立標準技術研究所 (NIST) 、 Payment Card Industry Security Standards Council ()、国際標準化機構 (ISO) などPCI) のセキュリティコントロールにマッピングするためのベストプラクティスをまとめています。
- AWS Config デベロッパーガイドのルールによるリソースの評価 – この AWS Config サービスは、リソース設定が内部プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。
- AWS Security Hub – これにより AWS のサービス 、 内のセキュリティ状態を包括的に確認できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールのリストについては、Security Hub のコントロールリファレンスを参照してください。
- Amazon GuardDuty – これにより AWS アカウント、疑わしいアクティビティや悪意のあるアクティビティがないか環境をモニタリングすることで、、、ワークロード、コンテナ、データに対する潜在的な脅威 AWS のサービス を検出できます。 GuardDuty は、特定のコンプライアンスフレームワークで義務付けられている侵入検出要件を満たすことでDSS、 PCI などのさまざまなコンプライアンス要件に対応するのに役立ちます。

- AWS Audit Manager – これにより AWS のサービス 、 AWS 使用状況を継続的に監査し、リスクと規制や業界標準へのコンプライアンスの管理を簡素化できます。

Amazon Timestream for InfluxDB の耐障害性

AWS グローバルインフラストラクチャは、 AWS リージョンとアベイラビリティーゾーンを中心に構築されています。 AWS リージョンは、低レイテンシー、高スループット、および冗長性の高いネットワークに接続されている、物理的に分離および分離された複数のアベイラビリティーゾーンを提供します。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性が高く、フォールトトレラントで、スケーラブルです。

AWS リージョンとアベイラビリティーゾーンの詳細については、AWS 「グローバルインフラストラクチャ」を参照してください。

Amazon Timestream for InfluxDB は、可用性と耐久性をサポートするために、内部バックアップを定期的に取得し、24 時間保持します。スナップショットは削除中に取得され、復元をサポートするために 30 日間保持されます。これらにアクセスまたは使用するには、 AWS サポート でチケットをファイルします。

マルチ AZ リカバリ機能を使用してインスタンスを作成できます。詳細については、「マルチ AZ DB インスタンスのデプロイ」を参照してください。

Amazon Timestream for InfluxDB のインフラストラクチャセキュリティ

マネージドサービスである Amazon Timestream for InfluxDB は、Amazon Web Services: セキュリティプロセスの概要ホワイトペーパーで説明されている AWS グローバルネットワークセキュリティ手順によって保護されています。

AWS 公開されたコントロールプレーンAPI呼び出しを使用して、ネットワーク経由で InfluxDB の Timestream にアクセスします。詳細については、「コントロールプレーン」と「データプレーン」を参照してください。クライアントは Transport Layer Security (TLS) 1.2 以降をサポートする必要があります。1.2 または TLS 1.3 をお勧めします。また、クライアントは、エフェメラルディフィヘルマン (PFS) や楕円曲線エフェメラルディフィヘルマン () など、完全なフォワードシークレット (DHE) を持つ暗号スイートもサポートする必要がありますECDHE。これらのモードは、Java 7 以降など、ほとんどの最新システムでサポートされています。

さらに、リクエストは、アクセスキー ID とプリンIAMシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時セキュリティ認証情報を生成し、リクエストに署名することもできます。

InfluxDB の Timestream は、InfluxDB インスタンスの Timestream が存在する特定の AWS リージョンにトラフィックが分離されるように設計されています。

セキュリティグループ

セキュリティグループにより DB インスタンスに対する送受信トラフィックへのアクセスを制御します。デフォルトでは、ネットワークアクセスは DB インスタンスに対してオフになっています。セキュリティグループで IP アドレス範囲、ポート、またはセキュリティグループからのアクセスを許可するルールを指定できます。Ingress ルールを設定したら、そのセキュリティグループに関連付けられているすべての DB インスタンスに、同じルールが適用されます。

詳細については、「での DB インスタンスへのアクセスの制御 VPC」を参照してください。

Timestream for InfluxDB の設定と脆弱性の分析

設定と IT コントロールは、 AWS お客様とお客様の責任を共有します。詳細については、 AWS 「 責任共有モデル」を参照してください。責任共有モデルに加えて、InfluxDB ユーザーの Timestream は以下に注意する必要があります。

- 関連するクライアント側の依存関係を使用して、クライアントアプリケーションにパッチを適用するのはお客様の責任です。
- お客様は、必要に応じてペネトレーションテストを検討する必要があります (https://aws.amazon.com/security/「ペネトレーションテスト/」を参照)。

InfluxDB の Timestream でのインシデント対応

Amazon Timestream for InfluxDB サービスインシデントは、 Personal Health Dashboard で報告されます。ダッシュボードの詳細については、 AWS Health こちら を参照してください。

InfluxDB の Timestream は、 を使用したレポートをサポートしています AWS CloudTrail。詳細については、「を使用した InfluxDB API呼び出しの Timestream の口グ記録 AWS CloudTrail」を参照してください。

Amazon Timestream for InfluxDB APIおよびインターフェイスVPCエンドポイント (AWS PrivateLink)

インターフェイスAPIエンドポイント を作成することで、 VPCと Amazon Timestream for InfluxDB コントロールプレーンエンドポイント間のプライベート接続を確立できます。 VPC インターフェイスエンドポイントは を搭載していますAWS PrivateLink。 AWS PrivateLink を使用すると、インターネットゲートウェイ、NATデバイス、VPN接続、または AWS Direct Connect 接続なしで、InfluxDB APIオペレーション用の Amazon Timestream にプライベートにアクセスできます。

のインスタンスは、Amazon Timestream for InfluxDB APIエンドポイントと通信するためにパブリック IP アドレスを必要としVPCません。また、インスタンスで InfluxDB APIオペレーションに使用可能な Timestream を使用するためにパブリック IP アドレスは必要ありません。VPC と Amazon Timestream for InfluxDB 間のトラフィックは、Amazon ネットワークを離れません。各インターフェイスエンドポイントは、サブネット内の 1 つ以上の Elastic Network Interface によって表されます。Elastic Network Interface の詳細については、「Amazon ユーザーガイド」の「Elastic Network Interface」を参照してください。 EC2

- VPC エンドポイントの詳細については、「Amazon ユーザーガイド」の「インターフェイスVPCエンドポイント (AWS PrivateLink)」を参照してください。 VPC
- InfluxDB APIオペレーションの Timestream の詳細については、「InfluxDB APIオペレーションの Timestream」を参照してください。

インターフェイスVPCエンドポイントを作成した後、エンドポイントのプライベートDNSホスト名を有効にすると、InfluxDB エンドポイントのデフォルトの Timestream (https://timestream-influxb.*Region*.amazonaws.com) はVPCエンドポイントを解決します。プライベートDNSホスト名を有効にしない場合、Amazon VPCは次の形式で使用できるDNSエンドポイント名を提供します。

```
VPC_Endpoint_ID.timestream-influxb.Region.vpce.amazonaws.com
```

詳細については、「Amazon VPCユーザーガイド」の「インターフェイスVPCエンドポイント (AWS PrivateLink)」を参照してください。InfluxDB の Timestream は、 内のすべてのAPIアクションへの呼び出しをサポートしますVPC。

> **Note**
>
> プライベートDNSホスト名は、 内の 1 つのVPCエンドポイントでのみ有効にできます VPC。追加のVPCエンドポイントを作成する場合は、プライベートDNSホスト名を無効にする必要があります。

VPC エンドポイントに関する考慮事項

Amazon Timestream for InfluxDB VPCエンドポイントのインターフェイスAPIエンドポイントを設定する前に、Amazon VPCユーザーガイドの「インターフェイスエンドポイントのプロパティと制限」を確認してください。Amazon Timestream for InfluxDB リソースの管理に関連するすべての Timestream for InfluxDB APIオペレーションは、 VPCを使用して から利用できます AWS PrivateLink。VPC エンドポイントポリシーは、InfluxDB エンドポイントの Timestream でサポートされています。 APIデフォルトでは、エンドポイントを介して Timestream for InfluxDB APIオペレーションへのフルアクセスが許可されます。詳細については、「Amazon VPCユーザーガイド」のVPC「エンドポイントを使用したサービスへのアクセスの制御」を参照してください。

InfluxDB の Timestream のインターフェイスVPCエンドポイントの作成 API

Amazon Timestream for InfluxDB のVPCエンドポイントは、Amazon VPCコンソールまたは APIを使用して作成できます AWS CLI。詳細については、「Amazon ユーザーガイド」の「インターフェイスエンドポイントの作成」を参照してください。 VPC

インターフェイスVPCエンドポイントを作成したら、エンドポイントのプライベートDNSホスト名を有効にできます。これを行うと、デフォルトの Amazon Timestream for InfluxDB エンドポイント (https://timestream-influxb.*Region*.amazonaws.com) はVPCエンドポイントを解決します。詳細については、「Amazon VPCユーザーガイド」の「インターフェイスエンドポイントを介したサービスへのアクセス」を参照してください。

Amazon Timestream for InfluxDB のVPCエンドポイントポリシーの作成 API

InfluxDB の Timestream へのアクセスを制御するエンドポイントポリシーをVPCエンドポイントにアタッチできますAPI。本ポリシーでは、以下を規定します。

- アクションを実行できるプリンシパル。
- 実行可能なアクション。
- このアクションを実行できるリソース。

詳細については、「Amazon VPCユーザーガイド」のVPC「エンドポイントを使用したサービスへのアクセスの制御」を参照してください。

Example VPC InfluxDB APIアクションの Timestream のエンドポイントポリシー

以下は、Timestream for InfluxDB のエンドポイントポリシーの例ですAPI。エンドポイントにアタッチされると、このポリシーは、すべてのリソースのすべてのプリンシパルに対して、InfluxDB APIアクションのリストされた Timestream へのアクセスを許可します。

```
{
 "Statement": [{
  "Principal": "*",
  "Effect": "Allow",
  "Action": [
   "timestream-influxb:CreateDbInstance",
   "timestream-influxb:UpdateDbInstance"
  ],
  "Resource": "*"
 }]
}
```

Example VPC 指定された AWS アカウントからのすべてのアクセスを拒否するエンドポイントポリシー

次のVPCエンドポイントポリシーは AWS アカウントを拒否します *123456789012* エンドポイントを使用したリソースへのすべてのアクセス。このポリシーは、他のアカウントからのすべてのアクションを許可します。

```
{
 "Statement": [{
   "Action": "*",
   "Effect": "Allow",
   "Resource": "*",
   "Principal": "*"
  },
  {
   "Action": "*",
   "Effect": "Deny",
   "Resource": "*",
   "Principal": {
    "AWS": [
     "123456789012"
```

```
    ]
   }
  }
 ]
}
```

Timestream for InfluxDB のセキュリティのベストプラクティス

Amazon Timestream for InfluxDB には、独自のセキュリティポリシーを開発および実装する際に考慮すべきセキュリティ機能が多数用意されています。以下のベストプラクティスは一般的なガイドラインであり、完全なセキュリティソリューションを説明するものではありません。これらのベストプラクティスはお客様の環境に適切ではないか、十分ではない場合があるため、これらは処方箋ではなく、有用な考慮事項と見なしてください。

最小特権アクセスの実装

アクセス許可を付与するときは、InfluxDB リソースの Timestream に対するアクセス許可を取得するユーザーを決定します。これらのリソースで許可したい特定のアクションを有効にするのも、お客様になります。このため、タスクの実行に必要なアクセス許可のみを付与する必要があります。最小特権アクセスの実装は、セキュリティリスクと、エラーや悪意によってもたらされる可能性のある影響の低減における基本になります。

IAM ロールを使用する

プロデューサーおよびクライアントアプリケーションは、InfluxDB DB インスタンスの Timestream にアクセスするための有効な認証情報を持っている必要があります。 AWS 認証情報をクライアントアプリケーションや Amazon S3 バケットに直接保存しないでください。これらは自動的にローテーションされない長期的な認証情報であり、漏洩するとビジネスに大きな影響が及ぶ場合があります。

代わりに、 IAMロールを使用してプロデューサーおよびクライアントアプリケーションの一時的な認証情報を管理し、InfluxDB DB インスタンスの Timestream にアクセスする必要があります。ロールを使用するときは、他のリソースにアクセスするために長期的な認証情報 (ユーザー名とパスワード、またはアクセスキーなど) を使用する必要がありません。

詳細については、 IAM ユーザーガイドの以下のトピックを参照してください。

- IAM ロール
- ロールの一般的なシナリオ: ユーザー、アプリケーション、およびサービス

AWS Identity and Access Management（IAM) アカウントを使用して、Amazon Timestream for InfluxDB APIオペレーション、特に Amazon Timestream for InfluxDB リソースを作成、変更、削除するオペレーションへのアクセスを制御します。このようなリソースには、DB インスタンス、セキュリティグループ、パラメータグループが含まれます。

- Amazon Timestream for InfluxDB リソースを管理するユーザーごとに、ユーザーを含め、個々のユーザーを作成します。Amazon Timestream for InfluxDB リソースの管理に AWS ルート認証情報を使用しないでください。
- それぞれの職務の実行に最低限必要になる一連のアクセス許可を各ユーザーに付与します。
- IAM グループを使用して、複数のユーザーのアクセス許可を効果的に管理します。
- IAM 認証情報のローテーションを定期的に行います。
- Amazon Timestream for InfluxDB のシークレットを自動的にローテーションするように AWS Secrets Manager を設定します。 InfluxDB 詳細については、AWS 「Secrets Manager ユーザーガイド」の「Secrets Manager シークレットのローテーション」を参照してください。 AWS また、 AWS Secrets Manager から認証情報をプログラムで取得することもできます。詳細については、「Secrets Manager ユーザーガイド」の「シークレット値の取得」を参照してください。 AWS
- を使用して、InfluxDB 流入APIトークンの Timestream を保護しますAPI トークン。

依存リソースでのサーバー側の暗号化の実装

保管中のデータと転送中のデータは、InfluxDB の Timestream で暗号化できます。詳細については、「送信中の暗号化」を参照してください。

CloudTrail を使用してAPI通話をモニタリングする

InfluxDB の Timestream は AWS CloudTrail、InfluxDB の Timestream でユーザー、ロール、または AWS のサービスによって実行されたアクションの記録を提供するサービスである と統合されています。

によって収集された情報を使用して CloudTrail、InfluxDB の Timestream に対して行われたリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時、その他の詳細を確認できます。

詳細については、「the section called “を使用した通話の LiveAnalytics API Timestream のログ記録 AWS CloudTrail”」を参照してください。

Amazon Timestream for InfluxDB はコントロールプレーン CloudTrail イベントをサポートしていますが、データプレーンはサポートしていません。詳細については、「コントロールプレーン」と「データプレーン」を参照してください。

パブリックアクセシビリティ

Amazon VPCサービスに基づいて仮想プライベートクラウド (VPC) 内で DB インスタンスを起動すると、その DB インスタンスのパブリックアクセシビリティをオンまたはオフにできます。作成する DB インスタンスにパブリック IP アドレスに解決されるDNS名前があるかどうかを指定するには、Public Accessibility パラメータを使用します。このパラメータを使用すると、DB インスタンスへのパブリックアクセスがあるかどうかを指定できます。

DB インスタンスが にあるVPCが、パブリックにアクセスできない場合は、接続または AWS Direct Connect 接続を使用して AWS Site-to-Site VPNプライベートネットワークからアクセスすることもできます。

DB インスタンスにパブリックにアクセス可能な場合は、サービス関連の脅威の拒否を防止または軽減するための措置を講じてください。詳細については、「サービス拒否攻撃の概要」と「ネットワークの保護」を参照してください。

API リファレンス

Amazon Timestream for InfluxDB の完全なリストと詳細についてはAPIs、「Amazon Timestream for InfluxDB APIs」を参照してください。

すべての AWS サービスに共通するエラーコードについては、 AWS サポートセクション を参照してください。

ドキュメント履歴

変更	説明	日付
ドキュメントのみの更新	Quotas トピックを更新して、デフォルトのクォータとシステム制限を分離しました。	2024 年 10 月 22 日
Amazon Timestream がクエリインサイトをサポートするようになりました	Timestream には、クエリの最適化、パフォーマンスの向上、コスト削減に役立つクエ	2024 年 10 月 22 日

	リインサイト機能のサポートが含まれるようになりました。	
Amazon Timestream for InfluxDB が既存のポリシーに更新されました。	Amazon Timestream for InfluxDB が、ルートテーブルを記述するための `ec2:DescribeRouteTables` アクションを既存の`AmazonTimestreamInfluxDBFullAccess` 管理ポリシーに追加しました	2024 年 10 月 8 日
`AmazonTimestreamReadOnlyAccess` – 既存のポリシーの更新	の Timestream LiveAnalytics は、 AWS アカウント 設定を記述するための`DescribeAccountSettings` アクセス許可を `AmazonTimestreamReadOnlyAccess` マネージドポリシーに追加しました。	2024 年 6 月 3 日
Amazon Timestream for が Timestream コンピューティングユニット (TCUs) をサポートする LiveAnalytics ようになりました	Amazon Timestream for LiveAnalytics には、クエリのニーズに割り当てられたコンピューティング容量を測定する Timestream コンピューティングユニット (TCUs) のサポートが含まれています。	2024 年 4 月 29 日

新しいポリシーが追加されました	Amazon Timestream for InfluxDB に 2 つの新しいポリシーが追加されました。1 つは、サービスがアカウント内のネットワークインターフェイスとセキュリティグループを管理できるようにするポリシーです。詳細については、「」を参照してくださいAmazonTimestreamInfluxDBServiceRolePolicy。Amazon Timestream InfluxDB インスタンスを作成、更新、削除、一覧表示し、パラメータグループを作成、一覧表示するための完全な管理アクセスを提供するもう 1 つの です。詳細については、「」を参照してくださいAmazonTimestreamInfluxDBFullAccess。	2024 年 3 月 14 日
Amazon Timestream for InfluxDB が一般公開されました。	このドキュメントでは、Amazon Timestream for InfluxDB の初期リリースについて説明します。	2024 年 3 月 14 日

Amazon Timestream for LiveAnalytics Query イベントは、 で利用できます。 AWS CloudTrail	Amazon Timestream for は、クエリAPIデータイベントをに発行する LiveAnalytics ようになりました AWS CloudTrail。お客様は、 AWS アカウントで行われたすべてのクエリAPIリクエストを監査し、リクエストを行ったIAMユーザー/ロール、リクエストが行われた日時、クエリされたデータベースとテーブル、リクエストのクエリ ID などの情報を確認できます。	2023 年 9 月 12 日
の Amazon Timestream LiveAnalytics UNLOAD	Amazon Timestream for LiveAnalytics では、 がクエリ結果を S3 UNLOADにエクスポートできるようになりました。	2023 年 5 月 12 日
既存のポリシー LiveAnalytics を更新するための Amazon Timestream。	マネージドポリシーに追加されたバッチロードアクセス許可。	2023 年 2 月 24 日
LiveAnalytics バッチロード用の Amazon Timestream。	Amazon Timestream for がバッチロード機能をサポートする LiveAnalytics ようになりました。	2023 年 2 月 24 日
Amazon Timestream for がをサポートする LiveAnalytics ようになりました AWS Backup。	Amazon Timestream for がをサポートする LiveAnalytics ようになりました AWS Backup。	2022 年 12 月 14 日

AWS マネージドポリシー LiveAnalytics の更新のための Amazon Timestream	既存の AWS マネージドポリシーの更新など LiveAnalytics、 の マネージドポリシーと Amazon Timestream に関する新しい情報。	2021 年 11 月 29 日
の Amazon Timestream がスケジュールされたクエリ LiveAnalytics をサポート	Amazon Timestream for では、スケジュールに基づいて、ユーザーに代わってクエリの実行がサポートされる LiveAnalytics ようになりました。	2021 年 11 月 29 日
Amazon Timestream for は、マグネティックストア LiveAnalytics をサポートしています。	Amazon Timestream for では、テーブルの書き込みにマグネティックストレージの使用がサポートされる LiveAnalytics ようになりました。	2021 年 11 月 29 日
LiveAnalytics マルチメジャーレコードの Amazon Timestream。	Amazon Timestream for は、時系列データを保存するためのよりコンパクトな形式をサポートする LiveAnalytics ようになりました。	2021 年 11 月 29 日
AWSマネージドポリシー LiveAnalytics の更新のための Amazon Timestream	既存の AWS マネージドポリシーの更新など LiveAnalytics、 の マネージドポリシーと Amazon Timestream に関する新しい情報。	2021 年 5 月 24 日
Amazon Timestream for LiveAnalytics が欧州 (フランクフルト) リージョンで利用可能になりました。	Amazon Timestream for LiveAnalytics が欧州 (フランクフルト) リージョン () で一般公開されましたeu-central-1 。	2021 年 4 月 23 日

Amazon Timestream for LiveAnalytics がVPCエンドポイント () をサポートするようになりましたAWS PrivateLink。	Amazon Timestream for では、VPCエンドポイント () の使用がサポートされる LiveAnalytics ようになりましたAWS PrivateLink。	2021 年 3 月 23 日
Amazon Timestream がクロステーブルクエリをサポートするようになりました。	の Amazon Timestream を使用して LiveAnalytics 、クロステーブルクエリを実行できます。	2021 年 2 月 10 日
Amazon Timestream for では、拡張クエリ実行統計がサポートされ LiveAnalytics るようになりました。	Amazon Timestream for は、スキャンされたデータ量などの拡張クエリ実行統計をサポートする LiveAnalytics ようになりました。	2021 年 2 月 10 日
Amazon Timestream for は、高度な時系列関数をサポートする LiveAnalytics ようになりました。	LiveAnalytics の Amazon Timestream を使用して、派生、積分、相関などの高度な時系列関数を使用してSQLクエリを実行できます。	2021 年 2 月 10 日
Amazon Timestream for LiveAnalytics が HIPAA、ISO、および PCI に準拠するようになりました。	HIPAA、、および PCI準拠のインフラストラクチャを必要とするワークロード LiveAnalytics に対して、Amazon Timestream ISOを使用できるようになりました。	2021 年 1 月 27 日

Amazon Timestream for は、オープンソースのテレグラフと grafana をサポートする LiveAnalytics ようになりました。	メトリクスの収集と報告にオープンソースのプラグイン駆動型サーバーエージェントである Telegraf と、データベースのオープンソース分析とモニタリングプラットフォームである Grafana を、用の Amazon Timestream で使用できるようになりました LiveAnalytics。	2020 年 11 月 25 日
Amazon Timestream for LiveAnalytics が一般公開されました。	このドキュメントでは、 の Amazon Timestream の初期リリースについて説明します LiveAnalytics。	2020 年 9 月 30 日