aws

Amazon EMR Serverless ユーザーガイド

Amazon EMR

Amazon EMR: Amazon EMR Serverless ユーザーガイド

Table of Contents

Amazon EMR Serverless とは

Amazon EMR Serverless は、サーバーレスランタイム環境EMRを提供する Amazon のデプロイオプションです。これにより、Apache Spark や Apache Hive などの最新のオープンソースフレームワークを使用する分析アプリケーションの運用が簡素化されます。EMR Serverless では、これらのフレームワークでアプリケーションを実行するためにクラスターを設定、最適化、保護、または運用する必要はありません。

EMR サーバーレスは、データ処理ジョブのリソースの過剰プロビジョニングやプロビジョニング不足を回避するのに役立ちます。EMR Serverless は、アプリケーションが必要とするリソースを自動的に決定し、ジョブを処理するためにこれらのリソースを取得し、ジョブが終了するとリソースを解放します。インタラクティブデータ分析など、アプリケーションが数秒以内に応答を必要とするユースケースでは、アプリケーションの作成時にアプリケーションが必要とするリソースを事前に初期化できます。

EMR Serverless を使用すると、オープンソースの互換性EMR、同時実行性、一般的なフレームワークのランタイムパフォーマンスの最適化など、Amazon の利点を引き続き享受できます。

EMR Serverless は、オープンソースフレームワークを使用してアプリケーションを簡単に運用したいお客様に適しています。ジョブの迅速な起動、自動キャパシティ管理、簡単なコスト管理を提供します。

概念

このセクションでは、EMRサーバーレスユーザーガイドに記載されているEMRサーバーレスの用語と概念について説明します。

リリースバージョン

Amazon EMRリリースは、ビッグデータエコシステムからの一連のオープンソースアプリケーションです。各リリースには、EMRサーバーレスがアプリケーションを実行できるようにデプロイおよび設定するために選択するさまざまなビッグデータアプリケーション、コンポーネント、および機能が含まれています。アプリケーションを作成するときは、そのリリースバージョンを指定する必要があります。アプリケーションで使用する Amazon EMRリリースバージョンとオープンソースフレームワークバージョンを選択します。プレリリースバージョンの詳細については、「」を参照してくださいAmazon EMR Serverless リリースバージョン。

アプリケーション

EMR Serverless を使用すると、オープンソースの分析フレームワークを使用する 1 つ以上の EMR Serverless アプリケーションを作成できます。アプリケーションを作成するには、次の属性を指定する必要があります。

- 使用するオープンソースフレームワークバージョンの Amazon EMRリリースバージョン。リリースバージョンを確認するには、「」を参照してくださいAmazon EMR Serverless リリースバージョン。
- Apache Spark や Apache Hive など、アプリケーションで使用する特定のランタイム。

アプリケーションを作成したら、データ処理ジョブまたはインタラクティブリクエストをアプリケーションに送信できます。

各 EMR Serverless アプリケーションは、他のアプリケーションとは厳密に異なる安全な Amazon Virtual Private Cloud (VPC) で実行されます。さらに、 AWS Identity and Access Management (IAM) ポリシーは、アプリケーションにアクセスできるユーザーとロールを定義します。また、アプリケーションによって発生した使用コストを制御および追跡するための制限を指定することもできます。

以下を実行する必要がある場合は、複数のアプリケーションを作成することを検討してください。

- さまざまなオープンソースフレームワークを使用する
- ユースケースごとに異なるバージョンのオープンソースフレームワークを使用する
- あるバージョンから別のバージョンにアップグレードするときに A/B テストを実行する
- テストシナリオと本番シナリオ用に別々の論理環境を維持する
- 独立したコスト管理と使用状況の追跡により、チームごとに個別の論理環境を提供する
- 異なる line-of-business アプリケーションを分離する

EMR Serverless は、リージョン内の複数のアベイラビリティーゾーンでワークロードを実行する方法を簡素化するリージョンサービスです。EMR Serverless でアプリケーションを使用する方法の詳細については、「」を参照してくださいアプリケーションの操作。

ジョブ実行

ジョブ実行は、EMRサーバーレスアプリケーションに送信されるリクエストで、アプリケーションは非同期的に実行し、完了まで追跡します。ジョブの例としては、Apache Hive アプリケーションに

送信する HiveQL クエリや、Apache Spark アプリケーションに送信する PySpark データ処理スクリプトなどがあります。ジョブを送信するときは、ジョブIAMがアクセスに使用する で作成されたランタイムロールを指定する必要があります。 AWS Amazon S3 オブジェクトなどの リソース。アプリケーションには複数のジョブ実行リクエストを送信でき、ジョブ実行ごとに異なるランタイムロールを使用して にアクセスできます。 AWS リソースの使用料金を見積もることができます。EMR サーバーレスアプリケーションは、ジョブを受信し、複数のジョブリクエストを同時に実行するとすぐにジョブの実行を開始します。EMR Serverless がジョブを実行する方法の詳細については、「」を参照してくださいジョブの実行。

ワーカー

EMR サーバーレスアプリケーションは、内部的にワーカーを使用してワークロードを実行します。これらのワーカーのデフォルトサイズは、アプリケーションタイプと Amazon EMRリリースバージョンに基づいています。ジョブ実行をスケジュールすると、これらのサイズを上書きできます。

ジョブを送信すると、EMRサーバーレスはアプリケーションがジョブに必要なリソースを計算し、ワーカーをスケジュールします。EMR サーバーレスは、ワークロードをタスクに分割し、イメージをダウンロードし、ワーカーをプロビジョニングしてセットアップし、ジョブが終了したらそれらを廃止します。EMR Serverless は、ジョブのすべての段階で必要なワークロードと並列処理に基づいて、ワーカーを自動的にスケールアップまたはスケールダウンします。この自動スケーリングにより、アプリケーションがワークロードを実行するために必要なワーカー数を見積もる必要がなくなります。

事前に初期化された容量

EMR Serverless には、ワーカーを初期化し、数秒で応答できるようにしておく、事前に初期化されたキャパシティ機能が用意されています。この容量により、アプリケーションのワーカーのウォームプールが効果的に作成されます。各アプリケーションにこの機能を設定するには、アプリケーションの `initial-capacity`パラメータを設定します。事前に初期化された容量を設定すると、反復アプリケーションと時間的制約のあるジョブを実装できるように、ジョブをすぐに開始できます。事前に初期化されたワーカーの詳細については、「」を参照してくださいアプリケーションの設定。

EMR Studio

EMR Studio は、EMRサーバーレスアプリケーションの管理に使用できるユーザーコンソールです。最初の EMR Serverless アプリケーションの作成時にアカウントに EMR Studio が存在しない場合、自動的に作成されます。EMR Studio には、Amazon EMRコンソールからアクセスすることも、 または IAM IAM Identity Center を介して ID プロバイダー (IdP) からのフェデレーションアクセスを有

効にすることもできます。これを行うと、ユーザーは Amazon EMRコンソールに直接アクセスすることなく Studio にアクセスし、EMRサーバーレスアプリケーションを管理できます。EMR サーバーレスアプリケーションと EMR Studio の連携の詳細については、EMR Studio コンソールからアプリケーションを操作する「」および「」を参照してくださいEMR Studio コンソールからのジョブの実行。

EMR Serverless の使用を開始するための前提条件

トピック

にサインアップする AWS アカウント

をお持ちでない場合 AWS アカウントで、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き 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。ルートユーザーはすべての にアクセスできます AWS のサービス アカウントの および リソース。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 サインアッププロセスが完了すると、 から確認メールが送信されます。https://aws.amazon.com/ に移動し、マイアカウント を選択すると、いつでも現在のアカウントアクティビティを表示し、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後 AWS アカウント、 をセキュリティで保護する AWS アカウントのルートユーザー、有効化 AWS IAM Identity Center、および 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. にサインインします。AWS Management Console 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。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、「」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。 AWS サインイン ユーザーガイド。
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、「 の仮想MFAデバイスの有効化」を参照してください。 AWS アカウントIAM ユーザーガイドのルートユーザー (コンソール) 。

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、「 の有効化」を参照してください。 AWS IAM Identity Center ()AWS IAM Identity Center ユーザーガイド。
2. IAM Identity Center で、ユーザーに管理アクセス権を付与します。

   の使用に関するチュートリアル IAM アイデンティティセンターディレクトリ ID ソースとして、「デフォルトを使用してユーザーアクセスを設定する」を参照してください。 IAM アイデンティティセンターディレクトリ ()AWS IAM Identity Center ユーザーガイド。

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、「 へのサインイン」を参照してください。 AWS の アクセスポータル AWS サインイン ユーザーガイド。

追加のユーザーにアクセス権を割り当てる

1. IAM Identity Center で、最小特権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、「」の「アクセス許可セットの作成」を参照してください。 AWS IAM Identity Center ユーザーガイド。

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、「」の「グループの追加」を参照してください。 AWS IAM Identity Center ユーザーガイド。

許可を付与する

本番環境では、よりきめ細かなポリシーを使用することをお勧めします。このようなポリシーの例については、「」を参照してくださいEMR Serverless のユーザーアクセスポリシーの例。アクセス管理の詳細については、「 のアクセス管理」を参照してください。 AWS IAM ユーザーガイドの リソース。

サンドボックス環境で EMR Serverless の使用を開始する必要があるユーザーには、次のようなポリシーを使用します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EMRStudioCreate",
            "Effect": "Allow",
            "Action": [
                "elasticmapreduce:CreateStudioPresignedUrl",
                "elasticmapreduce:DescribeStudio",
                "elasticmapreduce:CreateStudio",
                "elasticmapreduce:ListStudios"
            ],
            "Resource": "*"
        },
        {
            "Sid": "EMRServerlessFullAccess",
            "Effect": "Allow",
            "Action": [
                "emr-serverless:*"
            ],
            "Resource": "*"
        },
```

```
        {
            "Sid": "AllowEC2ENICreationWithEMRTags",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:network-interface/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:CalledViaLast": "ops.emr-serverless.amazonaws.com"
                }
            }
        },
        {
            "Sid": "AllowEMRServerlessServiceLinkedRoleCreation",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/*"
        }
    ]
}
```

アクセス権限を付与するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- のユーザーとグループ AWS IAM Identity Center:

  アクセス許可セットを作成します。の「アクセス許可セットの作成」の手順に従います。 AWS IAM Identity Center ユーザーガイド。

- ID プロバイダーIAMを通じて で管理されるユーザー：

  ID フェデレーションのロールを作成します。「 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 (フェデレーション) 」の手順に従います。

- IAM ユーザー：
  - ユーザーが担当できるロールを作成します。「 IAMユーザーガイド」のIAM「 ユーザーのロールの作成」の手順に従います。
  - (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加する。「 ユーザーガイド」の「ユーザーへのアクセス許可の追加 (コンソール) IAM」の指示に従います。

プログラマチックアクセス権を付与する

ユーザーが を操作する場合は、プログラムによるアクセスが必要です AWS の外部 AWS Management Console。 プログラムによるアクセスを許可する方法は、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 （ IAMIdentity Center で管理されるユーザー ）	へのプログラムによるリクエストに署名するには、一時的な認証情報を使用します。 AWS CLI, AWS SDKs、または 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。 • 向けの AWS CLI、「 の設定」を参照してください。 AWS CLI を使用する AWS IAM Identity Center ()AWS Command Line Interface ユーザーガイド。 • [AWS SDKs、 ツール、および AWS APIs、「」のIAM「 Identity Center 認証」を参照してください。 AWS SDKs および ツールリファレンスガイド 。
IAM	へのプログラムによるリクエストに署名するには、一時的な認証情報を使用します。 AWS CLI, AWS SDKs、または AWS APIs.	「 での一時的な認証情報の使用」の手順に従います。 AWSIAM ユーザーガイドの リソース。
IAM	(非推奨) 長期認証情報を使用して へのプログラムによるリクエス	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。

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
	トに署名する AWS CLI, AWS SDKs、または AWS APIs.	• 向けの AWS CLI、「」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。 AWS Command Line Interface ユーザーガイド。 • [AWS SDKs および ツールについては、「」の「長期的な認証情報を使用した認証」を参照してください。 AWS SDKs および ツールリファレンスガイド 。 • [AWS APIs「 IAM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。

のインストールと設定 AWS CLI

EMR Serverless を使用する場合はAPIs、最新バージョンの をインストールする必要があります。 AWS Command Line Interface (AWS CLI) 。 は必要ありません AWS CLI EMR Studio コンソールから EMR Serverless を使用するには、CLI「」の手順に従って、 を使用せずに開始できますコンソールから EMR Serverless の使用を開始する。

を設定するには AWS CLI

1. の最新バージョンをインストールするには AWS CLI for macOS 、Linux、または Windows については、「 の最新バージョンのインストールまたは更新」を参照してください。 AWS CLI.
2. を設定するには AWS CLI および へのアクセスの安全なセットアップ AWS のサービス EMR サーバーレスを含む については、「 を使用したクイック設定`aws configure`」を参照してください。
3. セットアップを確認するには、 DataBrew コマンドプロンプトで次のコマンドを入力します。

```
aws emr-serverless help
```

AWS CLI コマンドはデフォルトの を使用します。 AWS リージョン パラメータまたはプロファイルで設定しない限り、 設定から。を設定するには AWS リージョン パラメータを使用すると、各コマンドに `--region`パラメータを追加できます。

を設定するには AWS リージョン プロファイルを使用して、まず名前付きプロファイルを `~/.aws/config` ファイルまたは `%UserProfile%/.aws/config` ファイル (Microsoft Windows の場合) に追加します。の「名前付きプロファイル」のステップに従います。 AWS CLI。 次に、 AWS リージョン および次の例のようなコマンドを使用したその他の設定。

```
[profile emr-serverless]
aws_access_key_id = ACCESS-KEY-ID-OF-IAM-USER
aws_secret_access_key = SECRET-ACCESS-KEY-ID-OF-IAM-USER
region = us-east-1
output = text
```

コンソールを開きます。

このセクションのコンソール指向のトピックのほとんどは、Amazon EMRコンソール から開始されます。にまだサインインしていない場合 AWS アカウントにサインインし、Amazon EMRコンソールを開き、次のセクションに進み、Amazon の使用を開始しますEMR。

Amazon EMR Serverless の開始方法

このチュートリアルは、サンプルの Spark または Hive ワークロードをデプロイするときに EMR Serverless の使用を開始するのに役立ちます。独自のアプリケーションを作成、実行、デバッグします。このチュートリアルのほとんどの部分でデフォルトのオプションが表示されます。

EMR Serverless アプリケーションを起動する前に、以下のタスクを完了してください。

トピック

EMR Serverless を使用するアクセス許可を付与する

EMR Serverless を使用するには、EMRServerless のアクセス許可を付与するポリシーがアタッチされたユーザーまたはIAMロールが必要です。ユーザーを作成し、そのユーザーに適切なポリシーをアタッチするには、「」の手順に従います許可を付与する。

EMR Serverless 用のストレージを準備する

このチュートリアルでは、S3 バケットを使用して、EMRServerless アプリケーションを使用して実行するサンプルの Spark または Hive ワークロードからの出力ファイルとログを保存します。バケットを作成するには、「Amazon Simple Storage Service コンソールユーザーガイド」の「バケットの作成」の手順に従います。への追加の参照は、新しく作成されたバケットの名前*amzn-s3-demo-bucket*に置き換えます。

Studio を作成してインタラクティブワークロードEMRを実行する

EMR Serverless を使用して EMR Studio でホストされているノートブックを介してインタラクティブクエリを実行する場合は、Workspace を作成するには、S3 バケットと EMR Serverless の最小サービスロールを指定する必要があります。セットアップの手順については、「Amazon EMR管理

ガイド」のEMR「Studio のセットアップ」を参照してください。インタラクティブワークロードの詳細については、「」を参照してくださいEMR Studio を使用して EMR Serverless でインタラクティブワークロードを実行する。

ジョブランタイムロールを作成する

EMR Serverless で実行されるジョブは、実行時に特定の AWS のサービス および リソースに詳細なアクセス許可を提供するランタイムロールを使用します。このチュートリアルでは、パブリック S3 バケットがデータとスクリプトをホストします。バケットは出力*amzn-s3-demo-bucket*を保存します。

ジョブランタイムロールを設定するには、まず信頼ポリシーを使用してランタイムロールを作成し、EMRServerless が新しいロールを使用できるようにします。次に、必要な S3 アクセスポリシーをそのロールにアタッチします。以下の手順は、プロセス全体を示しています。

Console

1. https://console.aws.amazon.com/iam/ で IAM コンソールに移動します。
2. 左のナビゲーションペインで、[ロール] を選択します。
3. [ロールの作成] を選択します。
4. ロールタイプでは、カスタム信頼ポリシーを選択し、次の信頼ポリシーを貼り付けます。これにより、Amazon EMR Serverless アプリケーションに送信されたジョブが AWS のサービス ユーザーに代わって他の にアクセスできるようになります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "emr-serverless.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

5. 次へ を選択してアクセス許可の追加ページに移動し、ポリシーの作成 を選択します。
6. 新しいタブでポリシーの作成ページが開きます。JSON 以下のポリシーを貼り付けます。

> **Important**
>
> 以下のポリシー*amzn-s3-demo-bucket*の を、 で作成された実際のバケット名に置き換えますEMR Serverless 用のストレージを準備する。これは S3 アクセスの基本ポリシーです。ジョブランタイムロールの例の詳細については、「」を参照してくださいAmazon EMR Serverless のジョブランタイムロール。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadAccessForEMRSamples",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::*.elasticmapreduce",
                "arn:aws:s3:::*.elasticmapreduce/*"
            ]
        },
        {
            "Sid": "FullAccessToOutputBucket",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        },
        {
            "Sid": "GlueCreateAndReadDataCatalog",
            "Effect": "Allow",
            "Action": [
```

```
                "glue:GetDatabase",
                "glue:CreateDatabase",
                "glue:GetDataBases",
                "glue:CreateTable",
                "glue:GetTable",
                "glue:UpdateTable",
                "glue:DeleteTable",
                "glue:GetTables",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:CreatePartition",
                "glue:BatchCreatePartition",
                "glue:GetUserDefinedFunctions"
            ],
            "Resource": ["*"]
        }
    ]
}
```

7. ポリシーの確認ページで、 などのポリシーの名前を入力しますEMRServerlessS3AndGlueAccessPolicy。
8. アクセス許可ポリシーの添付ページを更新し、 を選択しますEMRServerlessS3AndGlueAccessPolicy。
9. 名前、レビュー、作成 ページにロール名 に、ロールの名前を入力します。例えば、 ですEMRServerlessS3RuntimeRole。このIAMロールを作成するには、「ロールの作成」を選択します。

CLI

1. IAM ロールに使用する信頼ポリシーemr-serverless-trust-policy.jsonを含む という名前のファイルを作成します。ファイルには、次のポリシーが含まれている必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "EMRServerlessTrustPolicy",
        "Action": "sts:AssumeRole",
        "Effect": "Allow",
        "Principal": {
            "Service": "emr-serverless.amazonaws.com"
```

```
        }
    }]
}
```

2. という名前のIAMロールを作成しますEMRServerlessS3RuntimeRole。前のステップで作成した信頼ポリシーを使用します。

```
aws iam create-role \
    --role-name EMRServerlessS3RuntimeRole \
    --assume-role-policy-document file://emr-serverless-trust-policy.json
```

出力の ARN に注目してください。ジョブの送信時に新しいロールARNの を使用します。この後、 と呼ばれます*job-role-arn*。

3. ワークロードのIAMポリシーemr-sample-access-policy.jsonを定義する という名前のファイルを作成します。これにより、パブリック S3 バケットに保存されているスクリプトとデータへの読み取りアクセスと、 への読み取り/書き込みアクセスが可能になります*amzn-s3-demo-bucket*。

> **Important**
>
> 以下のポリシー*amzn-s3-demo-bucket*の を、 で作成された実際のバケット名に置き換えEMR Serverless 用のストレージを準備するます。これは AWS Glue および S3 アクセスの基本ポリシーです。ジョブランタイムロールの例の詳細については、「」を参照してくださいAmazon EMR Serverless のジョブランタイムロール。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadAccessForEMRSamples",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::*.elasticmapreduce",
                "arn:aws:s3:::*.elasticmapreduce/*"
            ]
```

```
        },
        {
            "Sid": "FullAccessToOutputBucket",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        },
        {
            "Sid": "GlueCreateAndReadDataCatalog",
            "Effect": "Allow",
            "Action": [
                "glue:GetDatabase",
                "glue:CreateDatabase",
                "glue:GetDataBases",
                "glue:CreateTable",
                "glue:GetTable",Understanding default application behavior,
 including auto-start and auto-stop, as well as maximum capacity and worker
 configurations for configuring an application with &EMRServerless;.
                "glue:UpdateTable",
                "glue:DeleteTable",
                "glue:GetTables",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:CreatePartition",
                "glue:BatchCreatePartition",
                "glue:GetUserDefinedFunctions"
            ],
            "Resource": ["*"]
        }
    ]
}
```

4. ステップ 3 で作成したIAMポリシーファイルEMRServerlessS3AndGlueAccessPolicyを使用して、 という名前のポリシーを作成します。次のステップARNで新しいポリシーARNのを使用するため、出力の に注意してください。

```
aws iam create-policy \
    --policy-name EMRServerlessS3AndGlueAccessPolicy \
    --policy-document file://emr-sample-access-policy.json
```

新しいポリシーを出力ARNに書き留めます。次のステップ*policy-arn*で に置き換えます。

5. IAM ポリシーをジョブランタイムロール EMRServerlessS3AndGlueAccessPolicyにアタッチしますEMRServerlessS3RuntimeRole。

```
aws iam attach-role-policy \
    --role-name EMRServerlessS3RuntimeRole \
    --policy-arn policy-arn
```

コンソールから EMR Serverless の使用を開始する

完了すべきステップ

- ステップ 1: EMR Serverless アプリケーションを作成する
- ステップ 2: ジョブ実行またはインタラクティブワークロードを送信する
- ステップ 3: アプリケーション UI とログを表示する
- ステップ 4: クリーンアップする

ステップ 1: EMR Serverless アプリケーションを作成する

次のように EMR Serverless で新しいアプリケーションを作成します。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/emr で Amazon EMRコンソールを開きます。
2. 左側のナビゲーションペインで、EMRサーバーレスを選択してEMRサーバーレスランディングページに移動します。
3. EMR Serverless アプリケーションを作成または管理するには、EMRStudio UI が必要です。
   - アプリケーション AWS リージョン を作成する に EMR Studio が既にある場合は、アプリケーションの管理を選択して EMR Studio に移動するか、使用するスタジオを選択します。

- アプリケーションを作成する に EMR Studio がない場合は、「開始する」を選択し、「作成」を選択して Studio を起動 AWS リージョン します。EMR Serverless は、アプリケーションを作成および管理できるように EMR Studio を作成します。

4. 新しいタブで開くスタジオ UI の作成で、アプリケーションの名前、タイプ、リリースバージョンを入力します。バッチジョブのみを実行する場合は、「バッチジョブのデフォルト設定のみを使用する」を選択します。インタラクティブワークロードの場合は、「インタラクティブワークロードのデフォルト設定を使用する」を選択します。このオプションを使用すると、インタラクティブ対応アプリケーションでバッチジョブを実行することもできます。必要に応じて、後でこれらの設定を変更できます。

   詳細については、「スタジオの作成」を参照してください。

5. アプリケーションの作成を選択して、最初のアプリケーションを作成します。

次のセクションに進みステップ 2: ジョブ実行またはインタラクティブワークロードを送信する、ジョブ実行またはインタラクティブワークロードを送信します。

ステップ 2: ジョブ実行またはインタラクティブワークロードを送信する

Spark job run

このチュートリアルでは、 PySpark スクリプトを使用して、複数のテキストファイルにわたる一意の単語の出現数を計算します。パブリックで読み取り専用の S3 バケットには、スクリプトとデータセットの両方が保存されます。

Spark ジョブを実行するには

1. 次のコマンドを使用して、サンプルスクリプトwordcount.pyを新しいバケットにアップロードします。

```
aws s3 cp s3://us-east-1.elasticmapreduce/emr-containers/samples/wordcount/
scripts/wordcount.py s3://amzn-s3-demo-bucket/scripts/
```

2. 完了するとステップ 1: EMR Serverless アプリケーションを作成する、EMRStudio のアプリケーションの詳細ページに移動します。そこで、ジョブの送信オプションを選択します。
3. ジョブの送信ページで、以下を実行します。

   - 名前 フィールドに、ジョブ実行を呼び出す名前を入力します。

- ランタイムロールフィールドに、 で作成したロールの名前を入力しますジョブランタイムロールを作成する。
- スクリプトの場所フィールドに、S3 `s3://amzn-s3-demo-bucket/scripts/wordcount.py`として を入力しますURI。
- Script 引数 フィールドに、 と入力します`["s3://amzn-s3-demo-bucket/emr-serverless-spark/output"]`。
- Spark プロパティセクションで、テキストとして編集を選択し、次の設定を入力します。

```
--conf spark.executor.cores=1 --conf spark.executor.memory=4g --
conf spark.driver.cores=1 --conf spark.driver.memory=4g --conf
 spark.executor.instances=1
```

4. ジョブの実行を開始するには、ジョブの送信 を選択します。
5. ジョブ実行タブには、新しいジョブ実行が実行ステータスで表示されます。

Hive job run

チュートリアルのこの部分では、テーブルを作成し、いくつかのレコードを挿入し、カウント集約クエリを実行します。Hive ジョブを実行するには、まず、単一のジョブの一部として実行するすべての Hive クエリを含むファイルを作成し、ファイルを S3 にアップロードし、Hive ジョブを開始するときにこの S3 パスを指定します。

Hive ジョブを実行するには

1. Hive ジョブで実行するすべてのクエリ`hive-query.ql`を含む というファイルを作成します。

```
create database if not exists emrserverless;
use emrserverless;
create table if not exists test_table(id int);
drop table if exists Values__Tmp__Table__1;
insert into test_table values (1),(2),(2),(3),(3),(3);
select id, count(id) from test_table group by id order by id desc;
```

2. 次のコマンドを使用して S3 バケット`hive-query.ql`にアップロードします。

```
aws s3 cp hive-query.ql s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-
query.ql
```

3. 完了するとステップ 1: EMR Serverless アプリケーションを作成する、EMRStudio のアプリケーションの詳細ページに移動します。そこで、ジョブの送信オプションを選択します。
4. ジョブの送信ページで、以下を完了します。
   - 名前 フィールドに、ジョブ実行を呼び出す名前を入力します。
   - ランタイムロールフィールドに、 で作成したロールの名前を入力しますジョブランタイムロールを作成する。
   - スクリプトの場所 フィールドに、S3 `s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-query.ql`として を入力しますURI。
   - Hive プロパティセクションで、テキストとして編集 を選択し、次の設定を入力します。

     ```
     --hiveconf hive.log.explain.output=false
     ```

   - ジョブ設定セクションで、 として編集 JSONを選択し、次の を入力しますJSON。

     ```
     {
        "applicationConfiguration":
        [{
            "classification": "hive-site",
               "properties": {
                   "hive.exec.scratchdir": "s3://amzn-s3-demo-bucket/emr-
     serverless-hive/hive/scratch",
                   "hive.metastore.warehouse.dir": "s3://amzn-s3-demo-bucket/emr-
     serverless-hive/hive/warehouse",
                   "hive.driver.cores": "2",
                   "hive.driver.memory": "4g",
                   "hive.tez.container.size": "4096",
                   "hive.tez.cpu.vcores": "1"
               }
         }]
     }
     ```

5. ジョブの実行を開始するには、ジョブの送信 を選択します。
6. ジョブ実行タブには、新しいジョブ実行が実行ステータスで表示されます。

Interactive workload

Amazon EMR6.14.0 以降では、EMRStudio でホストされているノートブックを使用して、EMRServerless で Spark のインタラクティブワークロードを実行できます。アクセス許

可や前提条件などの詳細については、「」を参照してくださいEMR Studio を使用して EMR Serverless でインタラクティブワークロードを実行する。

アプリケーションを作成し、必要なアクセス許可を設定したら、次の手順を使用して EMR Studio でインタラクティブノートブックを実行します。

1. EMR Studio の Workspaces タブに移動します。Amazon S3 ストレージロケーションと EMR Studio サービスロール を設定する必要がある場合は、画面上部のバナーでスタジオを設定するボタンを選択します。
2. ノートブックにアクセスするには、Workspace を選択するか、新しい Workspace を作成します。クイック起動を使用して、新しいタブで Workspace を開きます。
3. 新しく開いたタブに移動します。左側のナビゲーションからコンピューティングアイコンを選択します。EMR Serverless をコンピューティングタイプ として選択します。
4. 前のセクションで作成したインタラクティブ対応アプリケーションを選択します。
5. Runtime ロールフィールドに、ジョブ実行のために EMR Serverless アプリケーションが引き受けることができるIAMロールの名前を入力します。ランタイムロールの詳細については、「Amazon Serverless ユーザーガイド」の「ジョブランタイムロール」を参照してください。 EMR
6. アタッチ を選択します。これには最大 1 分かかる場合があります。ページはアタッチされると更新されます。
7. カーネルを選択し、ノートブックを起動します。EMR Serverless でサンプルノートブックを参照し、Workspace にコピーすることもできます。サンプルノートブックにアクセスするには、左側のナビゲーションの**{...}**メニューに移動し、ノートブックファイル名`serverless`に があるノートブックをブラウズします。
8. ノートブックでは、ドライバーログリンクと、ジョブをモニタリングするためのメトリクスを提供するリアルタイムインターフェイスである Apache Spark UI へのリンクにアクセスできます。詳細については、「Amazon EMR Serverless ユーザーガイド」の「サーバーレスアプリケーションとジョブのモニタリング」を参照してください。 EMR

Studio ワークスペースにアプリケーションをアタッチすると、まだ実行されていない場合、アプリケーションは自動的にトリガーされます。また、アプリケーションを事前に起動し、ワークスペースにアタッチする前に準備しておくこともできます。

ステップ 3: アプリケーション UI とログを表示する

アプリケーション UI を表示するには、まずジョブ実行を識別します。Spark UI または Hive Tez UI のオプションは、ジョブタイプに基づいて、そのジョブ実行のオプションの最初の行で使用できます。適切なオプションを選択します。

Spark UI を選択した場合は、Executors タブを選択してドライバーとエグゼキューターのログを表示します。Hive Tez UI を選択した場合は、すべてのタスクタブを選択してログを表示します。

ジョブ実行ステータスが成功 と表示されたら、S3 バケットでジョブの出力を表示できます。

ステップ 4: クリーンアップする

作成したアプリケーションは 15 分間非アクティブになった後も自動停止する必要がありますが、再度使用しないリソースをリリースすることをお勧めします。

アプリケーションを削除するには、アプリケーションのリストページに移動します。作成したアプリケーションを選択し、アクション → 停止を選択してアプリケーションを停止します。アプリケーションが STOPPED状態になったら、同じアプリケーションを選択し、アクション → 削除 を選択します。

Spark ジョブと Hive ジョブの実行例の詳細については、Spark ジョブ「」および「」を参照してくださいHive ジョブ。

の開始方法 AWS CLI

ステップ 1: EMR Serverless アプリケーションを作成する

`emr-serverless create-application` コマンドを使用して、最初の EMR Serverless アプリケーションを作成します。使用するアプリケーションバージョンに関連付けられたアプリケーションタイプと Amazon EMRリリースラベルを指定する必要があります。アプリケーションの名前はオプションです。

Spark

Spark アプリケーションを作成するには、次のコマンドを実行します。

```
aws emr-serverless create-application \
```

```
    --release-label emr-6.6.0 \
    --type "SPARK" \
    --name my-application
```

Hive

Hive アプリケーションを作成するには、次のコマンドを実行します。

```
aws emr-serverless create-application \
    --release-label emr-6.6.0 \
    --type "HIVE" \
    --name my-application
```

出力で返されるアプリケーション ID を書き留めます。ID を使用してアプリケーションを起動し、ジョブの送信中に と呼びます。この後、 と呼ばれます*application-id*。

に進む前にステップ 2: EMR Serverless アプリケーションにジョブ実行を送信する、アプリケーションが `get-application` で CREATED状態になっていることを確認してくださいAPI。

```
aws emr-serverless get-application \
    --application-id application-id
```

EMR Serverless は、リクエストされたジョブに対応するためのワーカーを作成します。デフォルトでは、これらはオンデマンドで作成されますが、アプリケーションの作成時に `initialCapacity`パラメータを設定することで、事前に初期化された容量を指定することもできます。アプリケーションが `maximumCapacity`パラメータで使用できる最大容量の合計を制限することもできます。これらのオプションの詳細については、アプリケーションの設定を参照してください。

ステップ 2: EMR Serverless アプリケーションにジョブ実行を送信する

これで、EMRサーバーレスアプリケーションはジョブを実行する準備が整いました。

Spark

このステップでは、 PySpark スクリプトを使用して、複数のテキストファイルにわたる一意の単語の出現数を計算します。パブリックで読み取り専用の S3 バケットには、スクリプトとデータセットの両方が保存されます。アプリケーションは、Spark ランタイムの出力ファイルとログデータを、作成した S3 バケットの `/output`および `/logs` ディレクトリに送信します。

Spark ジョブを実行するには

1. 次のコマンドを使用して、新しいバケットに実行するサンプルスクリプトをコピーします。

```
aws s3 cp s3://us-east-1.elasticmapreduce/emr-containers/samples/wordcount/
scripts/wordcount.py s3://amzn-s3-demo-bucket/scripts/
```

2. 次のコマンドで、 をアプリケーション ID *application-id*に置き換えます。*job-role-arn* でARN作成したランタイムロールに置き換えますジョブランタイムロールを作成する。置換 *job-run-name* ジョブ実行を呼び出す名前。すべての*amzn-s3-demo-bucket*文字列を作成した Amazon S3 バケットに置き換え、パス/outputに追加します。これにより、EMRサーバーレスがアプリケーションの出力ファイルをコピーできる新しいフォルダがバケットに作成されます。

```
aws emr-serverless start-job-run \
    --application-id application-id \
    --execution-role-arn job-role-arn \
    --name job-run-name \
    --job-driver '{
        "sparkSubmit": {
          "entryPoint": "s3://amzn-s3-demo-bucket/scripts/wordcount.py",
          "entryPointArguments": ["s3://amzn-s3-demo-bucket/emr-serverless-
spark/output"],
          "sparkSubmitParameters": "--conf spark.executor.cores=1
 --conf spark.executor.memory=4g --conf spark.driver.cores=1 --conf
 spark.driver.memory=4g --conf spark.executor.instances=1"
        }
    }'
```

3. 出力 で返されるジョブ実行 ID に注意してください。次のステップで、 をこの ID *job-run-id*に置き換えます。

Hive

このチュートリアルでは、テーブルを作成し、いくつかのレコードを挿入し、カウント集約クエリを実行します。Hive ジョブを実行するには、まず、単一のジョブの一部として実行するすべての Hive クエリを含むファイルを作成し、ファイルを S3 にアップロードし、Hive ジョブを開始するときにこの S3 パスを指定します。

Hive ジョブを実行するには

1. Hive ジョブで実行するすべてのクエリhive-query.qlを含む というファイルを作成します。

```
create database if not exists emrserverless;
use emrserverless;
create table if not exists test_table(id int);
drop table if exists Values__Tmp__Table__1;
insert into test_table values (1),(2),(2),(3),(3),(3);
select id, count(id) from test_table group by id order by id desc;
```

2. 次のコマンドを使用して S3 バケットhive-query.qlにアップロードします。

```
aws s3 cp hive-query.ql s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-
query.ql
```

3. 次のコマンドで、 を独自のアプリケーション ID *application-id*に置き換えます。*job-role-arn* でARN作成したランタイムロールに置き換えますジョブランタイムロールを作成する。すべての*amzn-s3-demo-bucket*文字列を作成した Amazon S3 バケットに置き換え、パス/logsに /outputと を追加します。これにより、バケットに新しいフォルダが作成され、EMRサーバーレスはアプリケーションの出力ファイルとログファイルをコピーできます。

```
aws emr-serverless start-job-run \
    --application-id application-id \
    --execution-role-arn job-role-arn \
    --job-driver '{
        "hive": {
          "query": "s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-
query.ql",
          "parameters": "--hiveconf hive.log.explain.output=false"
        }
    }' \
    --configuration-overrides '{
      "applicationConfiguration": [{
        "classification": "hive-site",
          "properties": {
            "hive.exec.scratchdir": "s3://amzn-s3-demo-bucket/emr-serverless-
hive/hive/scratch",
            "hive.metastore.warehouse.dir": "s3://amzn-s3-demo-bucket/emr-
serverless-hive/hive/warehouse",
```

```
            "hive.driver.cores": "2",
            "hive.driver.memory": "4g",
            "hive.tez.container.size": "4096",
            "hive.tez.cpu.vcores": "1"
            }
        }],
        "monitoringConfiguration": {
          "s3MonitoringConfiguration": {
            "logUri": "s3://amzn-s3-demo-bucket/emr-serverless-hive/logs"
          }
        }
    }'
```

4. 出力で返されるジョブ実行 ID に注意してください。次のステップで、 をこの ID *job-run-id*に置き換えます。

ステップ 3: ジョブ実行の出力を確認する

ジョブの実行は通常、完了までに 3～5 分かかります。

Spark

次のコマンドを使用して、Spark ジョブの状態を確認できます。

```
aws emr-serverless get-job-run \
    --application-id application-id \
    --job-run-id job-run-id
```

ログの送信先を に設定するとs3://*amzn-s3-demo-bucket*/emr-serverless-spark/logs、この特定のジョブ実行のログが で確認できますs3://*amzn-s3-demo-bucket*/emr-serverless-spark/logs/applications/*application-id*/jobs/*job-run-id*。

Spark アプリケーションの場合、EMRServerless はイベントログを 30 秒ごとに S3 ログ送信先のsparklogsフォルダにプッシュします。ジョブが完了すると、ドライバーとエグゼキューターの Spark ランタイムログが、 driverや などのワーカータイプによって適切に名前が付けられたフォルダにアップロードされますexecutor。 PySpark ジョブの出力が にアップロードされますs3://*amzn-s3-demo-bucket*/output/。

Hive

次のコマンドを使用して、Hive ジョブの状態を確認できます。

```
aws emr-serverless get-job-run \
    --application-id application-id \
    --job-run-id job-run-id
```

ログの送信先を に設定するとs3://*amzn-s3-demo-bucket*/emr-serverless-hive/logs、この特定のジョブのログを で確認できますs3://*amzn-s3-demo-bucket*/emr-serverless-hive/logs/applications/*application-id*/jobs/*job-run-id*。

Hive アプリケーションの場合、EMRServerless は Hive ドライバーを S3 ログ送信先の HIVE_DRIVER フォルダに、Tez タスクログを TEZ_TASKフォルダに継続的にアップロードします。ジョブの実行が SUCCEEDED状態に達すると、Hive クエリの出力は、 の monitoringConfigurationフィールドで指定した Amazon S3 ロケーションで使用可能になりますconfigurationOverrides。

ステップ 4: クリーンアップする

このチュートリアルの操作が完了したら、作成したリソースの削除を検討してください。再度使用しないリソースをリリースすることをお勧めします。

アプリケーションを削除する

アプリケーションを削除するには、次のコマンドを使用します。

```
aws emr-serverless delete-application \
    --application-id application-id
```

S3 ログバケットを削除する

S3 ログ記録と出力バケットを削除するには、次のコマンドを使用します。*amzn-s3-demo-bucket* を、 で作成された S3 バケットの実際の名前に置き換えEMR Serverless 用のストレージを準備するます。

```
aws s3 rm s3://amzn-s3-demo-bucket --recursive
aws s3api delete-bucket --bucket amzn-s3-demo-bucket
```

ジョブランタイムロールを削除する

ランタイムロールを削除するには、ロールからポリシーをデタッチします。その後、ロールとポリシーの両方を削除できます。

```
aws iam detach-role-policy \
    --role-name EMRServerlessS3RuntimeRole \
    --policy-arn policy-arn
```

ロールを削除するには、次のコマンドを使用します。

```
aws iam delete-role \
    --role-name EMRServerlessS3RuntimeRole
```

ロールにアタッチされたポリシーを削除するには、次のコマンドを使用します。

```
aws iam delete-policy \
    --policy-arn policy-arn
```

Spark ジョブと Hive ジョブの実行例の詳細については、Spark ジョブ「」および「」を参照してくださいHive ジョブ。

アプリケーションの操作

このセクションでは、Amazon EMR Serverless アプリケーションと Spark AWS CLI および Hive エンジンのデフォルトを操作する方法について説明します。

トピック

アプリケーションの状態

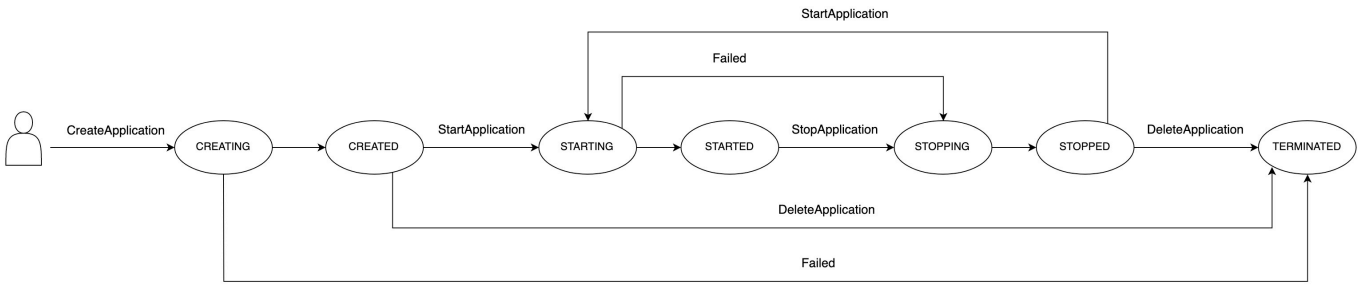
EMR Serverless でアプリケーションを作成すると、アプリケーションの実行は `CREATING`状態になります。その後、以下の状態を経由して完了 (コード `0` で終了) または失敗 (0 以外のコードで終了) します。

アプリケーションには次の状態があります。

状態	説明
[作成中]	アプリケーションは準備中であり、まだ使用する準備ができていません。
作成日	アプリケーションは作成されていますが、容量はまだプロビジョニングされていません。アプリケーションを変更して、初期容量設定を変更できます。
スタート	アプリケーションが起動し、キャパシティをプロビジョニングしています。

状態	説明
開始済み	アプリケーションは新しいジョブを受け入れる準備ができています。アプリケーションは、この状態にある場合にのみジョブを受け入れます。
停止中	すべてのジョブが完了し、アプリケーションはその容量を解放します。
停止	アプリケーションが停止し、アプリケーション上でリソースが実行されていません。アプリケーションを変更して、初期容量設定を変更できます。
終了済み	アプリケーションが終了し、アプリケーションリストに表示されません。

次の図は、EMRサーバーレスアプリケーションの状態の軌道を示しています。



EMR Studio コンソールからアプリケーションを操作する

EMR Studio コンソールから、EMRサーバーレスアプリケーションを作成、表示、管理できます。EMR Studio コンソールに移動するには、「コンソール から開始する」の手順に従ってください。

アプリケーションの作成

アプリケーションの作成 ページでは、以下の手順に従って EMR Serverless アプリケーションを作成できます。

1. 名前 フィールドに、アプリケーションを呼び出す名前を入力します。
2. Type フィールドで、アプリケーションのタイプとして Spark または Hive を選択します。
3. Release version フィールドで、EMRリリース番号を選択します。
4. アーキテクチャオプションで、使用する命令セットアーキテクチャを選択します。詳細については、「Amazon EMR Serverless アーキテクチャオプション」を参照してください。

   - arm64 — 64 ビットARMアーキテクチャ、Graviton プロセッサを使用
   - x86_64 — 64 ビット x86 アーキテクチャ、x86 ベースのプロセッサを使用
5. 残りのフィールドには、デフォルト設定とカスタム設定の 2 つのアプリケーション設定オプションがあります。これらのフィールドはオプションです。

   デフォルト設定 — デフォルト設定では、事前に初期化された容量でアプリケーションをすばやく作成できます。これには、Spark 用のドライバー 1 つとエグゼキューター 1 つ、Hive 用のドライバー 1 つと Tez タスク 1 つが含まれます。デフォルト設定では、 へのネットワーク接続は有効になりませんVPCs。アプリケーションは、アイドル状態が 15 分間続くと停止するように設定され、ジョブの送信時に自動起動します。

   カスタム設定 — カスタム設定では、次のプロパティを変更できます。

   - 事前初期化された容量 — ドライバーとエグゼキューターまたは Hive Tez タスクワーカーの数、および各ワーカーのサイズ。
   - アプリケーション制限 — アプリケーションの最大容量。
   - アプリケーション動作 — アプリケーションの自動起動および自動停止動作。
   - ネットワーク接続 — VPCリソースへのネットワーク接続。
   - タグ — アプリケーションに割り当てることができるカスタムタグ。

   事前初期化された容量、アプリケーションの制限、およびアプリケーションの動作の詳細については、「」を参照してくださいアプリケーションの設定。ネットワーク接続の詳細については、「」を参照してくださいVPC アクセスの設定。
6. アプリケーションを作成するには、アプリケーションの作成 を選択します。

アプリケーションを一覧表示する

既存のすべての EMR Serverless アプリケーションは、アプリケーションのリストページから表示できます。アプリケーションの名前を選択して、そのアプリケーションの詳細ページに移動できます。

アプリケーションの管理

アプリケーションに対して、アプリケーションのリストページまたは特定のアプリケーションの詳細ページから、次のアクションを実行できます。

アプリケーションを起動する

アプリケーションを手動で起動するには、このオプションを選択します。

アプリケーションの停止

アプリケーションを手動で停止するには、このオプションを選択します。アプリケーションを停止するには、実行中のジョブがない必要があります。アプリケーション状態の移行の詳細については、「」を参照してくださいアプリケーションの状態。

アプリケーションを設定する

アプリケーションの設定ページからアプリケーションのオプション設定を編集します。ほとんどのアプリケーション設定を変更できます。例えば、アプリケーションのリリースラベルを変更して Amazon の別のバージョンにアップグレードしたりEMR、アーキテクチャを x86_64 から arm64 に切り替えることができます。その他のオプション設定は、アプリケーションの作成ページのカスタム設定セクションにある設定と同じです。アプリケーション設定の詳細については、「」を参照してくださいアプリケーションの作成。

アプリケーションを削除する

アプリケーションを手動で削除するには、このオプションを選択します。削除するには、アプリケーションを停止する必要があります。アプリケーション状態の移行の詳細については、「」を参照してくださいアプリケーションの状態。

でのアプリケーションの操作 AWS CLI

から AWS CLI、個々のアプリケーションを作成、説明、削除できます。また、すべてのアプリケーションを一覧表示して、一目で表示することもできます。このセクションでは、これらのアクションを実行する方法について説明します。アプリケーションの起動、停止、更新などのアプリケーションオペレーションの詳細については、EMR「サーバーレスAPIリファレンス」を参照してくだ

さい。API を使用して EMR Serverless を使用する方法の例については AWS SDK for Java、 GitHub リポジトリの Java の例を参照してください。API を使用して EMR Serverless を使用する方法の例については AWS SDK for Python (Boto)、 GitHub リポジトリの Python の例を参照してください。

アプリケーションを作成するには、 を使用しますcreate-application。アプリケーションHIVEとして SPARKまたは を指定する必要がありますtype。このコマンドは、アプリケーションの ARN、名前、および ID を返します。

```
aws emr-serverless create-application \
--name my-application-name \
--type 'application-type' \
--release-label release-version
```

アプリケーションを記述するには、 を使用して get-application を指定しますapplication-id。このコマンドは、アプリケーションの状態と容量に関連する設定を返します。

```
aws emr-serverless get-application \
--application-id application-id
```

すべてのアプリケーションを一覧表示するには、 を呼び出しますlist-applications。このコマンドは と同じプロパティを返しますget-applicationが、すべてのアプリケーションが含まれます。

```
aws emr-serverless list-applications
```

アプリケーションを削除するには、 を呼び出しdelete-applicationて を指定しますapplication-id。

```
aws emr-serverless delete-application \
--application-id application-id
```

アプリケーションの設定

EMR Serverless では、使用するアプリケーションを設定できます。例えば、アプリケーションがスケールアップできる最大容量を設定し、ドライバーとワーカーが応答できるように事前初期化された容量を設定し、アプリケーションレベルでランタイムとモニタリング設定の一般的なセットを指定できます。次のページでは、EMRServerless を使用する際にアプリケーションを設定する方法について説明します。

トピック

アプリケーションの動作について

デフォルトのアプリケーション動作

Auto-start — デフォルトでは、アプリケーションはジョブの送信時に自動起動するように設定されています。この機能はオフにできます。

自動停止 — デフォルトでは、アプリケーションは 15 分間アイドル状態になると自動停止するように設定されています。アプリケーションが STOPPED状態に変わると、設定された事前初期化されたキャパシティが解放されます。アプリケーションが自動停止するまでのアイドル時間を変更することも、この機能をオフにすることもできます。

最大キャパシティ

アプリケーションがスケールアップできる最大容量を設定できます。最大容量は、、メモリ (GB)CPU、およびディスク (GB) で指定できます。

> **Note**
>
> ワーカー数にそのサイズを掛けることで、サポートされるワーカーサイズに比例するように最大容量を設定することをお勧めします。例えば、アプリケーションを 2、16 GB のメモリvCPUs、20 GB のディスクを持つ 50 人のワーカーに制限する場合は、最大容量を 100vCPUs、800 GB のメモリ、1000 GB のディスクに設定します。

サポートされているワーカー設定

次の表は、EMRサーバーレスに指定できるサポートされているワーカー設定とサイズを示しています。ワークロードの必要性に応じて、ドライバーとエグゼキューターにさまざまなサイズを設定できます。

CPU	「メモリ」	デフォルトのエフェメラルストレージ
1 vCPU	最小 2 GB、最大 8 GB、1 GB 刻み	20 GB ~ 200 GB
2 vCPU	最小 4 GB、最大 16 GB、1 GB 刻み	20 GB ~ 200 GB
4 vCPU	最小 8 GB、最大 30 GB、1 GB 刻み	20 GB ~ 200 GB
8 vCPU	最小 16 GB、最大 60 GB、4 GB 刻み	20 GB ~ 200 GB
16 vCPU	最小 32 GB、最大 120 GB、8 GB 刻み	20 GB ~ 200 GB

CPU — 各ワーカーには 1、2、4、8、または 16 を指定できますvCPUs。

メモリ — 各ワーカーには、前の表に記載されている制限内で GB で指定されたメモリがあります。Spark ジョブにはメモリオーバーヘッドがあります。つまり、使用するメモリが指定されたコンテナサイズを超えています。このオーバーヘッドは、 プロパティ`spark.driver.memoryOverhead`と で指定されます`spark.executor.memoryOverhead`。オーバーヘッドのデフォルト値はコンテナメモリの 10% で、最小 384 MB です。ワーカーサイズを選択するときは、このオーバーヘッドを考慮する必要があります。

例えば、ワーカーインスタンスvCPUs に 4 を選択し、事前に初期化されたストレージ容量が 30 GB の場合、Spark ジョブの実行メモリとして約 27 GB の値を設定する必要があります。これにより、事前初期化された容量を最大限に活用できます。使用可能なメモリは 27 GB で、27 GB (2.7 GB) の 10% が合計 29.7 GB になります。

ディスク — 各ワーカーには、最小サイズが 20 GB、最大サイズが 200 GB の一時ストレージディスクを設定できます。ワーカーごとに設定する 20 GB を超える追加ストレージに対してのみ料金が発生します。

事前初期化された容量

EMR Serverless には、ドライバーとワーカーを事前に初期化し、数秒で応答できる状態に保つオプション機能が用意されています。これにより、アプリケーションのワーカーのウォームプールが効果的に作成されます。この機能は、事前初期化されたキャパシティ と呼ばれます。この機能を設定するには、アプリケーションの initialCapacityパラメータを、事前初期化するワーカーの数に設定できます。ワーカー容量が初期化済みの場合、ジョブはすぐに開始されます。これは、反復アプリケーションと時間的制約のあるジョブを実装する場合に最適です。

ジョブを送信するときに、 のワーカーinitialCapacityが利用可能な場合、ジョブはそれらのリソースを使用して実行を開始します。これらのワーカーが他のジョブで既に使用されている場合、またはジョブが から利用可能なリソースよりも多くのリソースを必要とする場合initialCapacity、アプリケーションは追加のワーカーをリクエストし、アプリケーションに設定されているリソースの最大制限まで取得します。ジョブの実行が完了すると、使用したワーカーが解放され、アプリケーションで使用できるリソースの数は に戻りますinitialCapacity。アプリケーションは、ジョブの実行が終了した後でも、 リソースinitialCapacityの を維持します。アプリケーションは、ジョブを実行する必要がなくなったinitialCapacityときに、 を超える余分なリソースを解放します。

事前初期化されたキャパシティーは、アプリケーションの起動時に使用可能で、すぐに使用できます。アプリケーションが停止すると、事前初期化された容量は非アクティブになります。アプリケーションは、リクエストされた事前初期化されたキャパシティが作成され、使用準備が整った場合にのみ STARTED状態に移行します。アプリケーションが STARTED状態である間、EMRServerless は、事前に初期化された容量を、ジョブやインタラクティブワークロードで使用できるように維持します。この機能は、リリースされたコンテナまたは失敗したコンテナの容量を復元します。これにより、 InitialCapacity パラメータが指定するワーカーの数が維持されます。事前に初期化されたキャパシティがないアプリケーションの状態は、すぐに から CREATEDに変わる可能性がありますSTARTED。

アプリケーションが一定期間使用されていない場合は、デフォルトは 15 分で、事前に初期化された容量を解放するようにアプリケーションを設定できます。停止したアプリケーションは、新しいジョブを送信すると自動的に開始されます。アプリケーションの作成時にこれらの自動起動および停止設定を設定することも、アプリケーションが CREATED または STOPPED状態になったときに変更することもできます。

ワーカーごとにInitialCapacityカウントを変更し、CPU、メモリ、ディスクなどのコンピューティング設定を指定できます。部分的な変更を行うことができないため、値を変更するときにす

べてのコンピューティング設定を指定する必要があります。アプリケーションが CREATEDまたは STOPPED状態にある場合にのみ、設定を変更できます。

> **Note**
>
> アプリケーションによるリソースの使用を最適化するには、コンテナサイズを事前に初期化されたキャパシティワーカーのサイズに合わせることをお勧めします。例えば、Spark エグゼキュターサイズを 2CPUs に、メモリを 8 GB に設定し、初期化済みのキャパシティワーカーサイズを 4CPUs で 16 GB のメモリを使用する場合、Spark エグゼキュターはこのジョブに割り当てられたワーカーリソースの半分のみを使用します。

Spark と Hive の事前初期化済み容量のカスタマイズ

特定のビッグデータフレームワークで実行されるワークロード用に、事前に初期化された容量をさらにカスタマイズできます。例えば、ワークロードが Apache Spark で実行されるときに、ドライバーとして開始するワーカーの数とエグゼキュターとして開始するワーカーの数を指定できます。同様に、Apache Hive を使用する場合、Hive ドライバーとして開始するワーカーの数と、Tez タスクを実行するワーカーの数を指定できます。

Apache Hive を実行しているアプリケーションを事前に初期化された容量で設定する

次のAPIリクエストは、Amazon EMRリリース emr-6.6.0 に基づいて Apache Hive を実行するアプリケーションを作成します。アプリケーションは、それぞれ 2 vCPU と 4 GB のメモリを持つ 5 つの初期化済み Hive ドライバーと、それぞれ 4 vCPU と 8 GB のメモリを持つ 50 人の初期化済み Tez タスクワーカーから始まります。Hive クエリがこのアプリケーションで実行されると、まず事前初期化されたワーカーを使用し、すぐに実行を開始します。初期化済みのワーカーがすべてビジー状態であり、より多くの Hive ジョブが送信された場合、アプリケーションは合計 400 vCPU と 1024 GB のメモリにスケールできます。オプションで、 DRIVERまたはTEZ_TASKワーカーの容量を省略できます。

```
aws emr-serverless create-application \
  --type "HIVE" \
  --name my-application-name \
  --release-label emr-6.6.0 \
  --initial-capacity '{
    "DRIVER": {
        "workerCount": 5,
        "workerConfiguration": {
```

```
            "cpu": "2vCPU",
            "memory": "4GB"
        }
    },
    "TEZ_TASK": {
        "workerCount": 50,
        "workerConfiguration": {
            "cpu": "4vCPU",
            "memory": "8GB"
        }
    }
}' \
--maximum-capacity '{
    "cpu": "400vCPU",
    "memory": "1024GB"
}'
```

Apache Spark を実行しているアプリケーションを事前に初期化された容量で設定する

次のAPIリクエストは、Amazon EMRリリース 6.6.0 に基づいて Apache Spark 3.2.0 を実行するアプリケーションを作成します。アプリケーションは、それぞれ 2 vCPU と 4 GB のメモリを持つ 5 つの事前初期化された Spark ドライバーと、それぞれ 4 vCPU と 8 GB のメモリを持つ 50 の事前初期化されたエグゼキューターで始まります。このアプリケーションで Spark ジョブを実行すると、まず事前初期化されたワーカーを使用し、すぐに実行を開始します。初期化済みのワーカーがすべてビジー状態で、より多くの Spark ジョブが送信された場合、アプリケーションは合計 400 vCPU と 1024 GB のメモリにスケールできます。オプションで、 `DRIVER`または の容量を省略できますEXECUTOR。

> **Note**
>
> Spark は、ドライバーとエグゼキューターにリクエストされたメモリに、デフォルト値の 10% で設定可能なメモリオーバーヘッドを追加します。ジョブが事前初期化されたワーカーを使用する場合、初期容量メモリ設定は、ジョブとオーバーヘッドリクエストのメモリよりも大きくする必要があります。

```
aws emr-serverless create-application \
  --type "SPARK" \
  --name my-application-name \
  --release-label emr-6.6.0 \
```

```
  --initial-capacity '{
    "DRIVER": {
        "workerCount": 5,
        "workerConfiguration": {
            "cpu": "2vCPU",
            "memory": "4GB"
        }
    },
    "EXECUTOR": {
        "workerCount": 50,
        "workerConfiguration": {
            "cpu": "4vCPU",
            "memory": "8GB"
        }
    }
  }' \
  --maximum-capacity '{
    "cpu": "400vCPU",
    "memory": "1024GB"
  }'
```

EMR Serverless のデフォルトのアプリケーション設定

同じアプリケーションで送信するすべてのジョブに対して、アプリケーションレベルでランタイム設定とモニタリング設定の共通セットを指定できます。これにより、ジョブごとに同じ設定を送信する必要性に関連する追加のオーバーヘッドが軽減されます。

設定は、次の時点で変更できます。

- ジョブの送信時にアプリケーションレベルの設定を宣言します。
- ジョブの実行中にデフォルト設定を上書きします。

以下のセクションでは、詳細と詳細なコンテキストの例を示します。

アプリケーションレベルでの設定の宣言

アプリケーションで送信するジョブに対して、アプリケーションレベルのログ記録とランタイム設定プロパティを指定できます。

monitoringConfiguration

アプリケーションで送信するジョブのログ設定を指定するには、monitoringConfigurationフィールドを使用します。EMR Serverless のログ記録の詳細については、「」を参照してくださいログの保存。

runtimeConfiguration

などのランタイム設定プロパティを指定するにはspark-defaults、runtimeConfigurationフィールドに設定オブジェクトを指定します。これは、アプリケーションで送信するすべてのジョブのデフォルト設定に影響します。詳細については、「Hive 設定オーバーライドパラメータ」および「Spark 設定オーバーライドパラメータ」を参照してください。

使用可能な設定分類は、特定の EMR Serverless リリースによって異なります。例えば、カスタム Log4j spark-driver-log4j2と の分類spark-executor-log4j2は、リリース 6.8.0 以降でのみ使用できます。アプリケーション固有のプロパティのリストについては、Spark ジョブのプロパティ「」および「」を参照してくださいHive ジョブのプロパティ。

アプリケーションレベルでデータ保護 および Java 17 ランタイムAWS Secrets Manager 用に Apache Log4j2 プロパティ を設定することもできます。

アプリケーションレベルで Secrets Manager シークレットを渡すには、シークレットを使用して EMR Serverless アプリケーションを作成または更新する必要があるユーザーとロールに次のポリシーをアタッチします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SecretsManagerPolicy",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:secretsmanager:your-secret-arn"
        }
    ]
}
```

シークレットのカスタムポリシーの作成の詳細については、 AWS Secrets Manager ユーザーガイドの「 のアクセス許可ポリシーの例 AWS Secrets Manager」を参照してください。

> **Note**
>
> アプリケーションレベルで指定した runtimeConfigurationは、 StartJobRun applicationConfigurationの にマッピングされますAPI。

宣言例

次の例は、 でデフォルト設定を宣言する方法を示していますcreate-application。

```
aws emr-serverless create-application \
    --release-label release-version  \
    --type SPARK \
    --name my-application-name \
    --runtime-configuration '[
        {
            "classification": "spark-defaults",
            "properties": {
                "spark.driver.cores": "4",
                "spark.executor.cores": "2",
                "spark.driver.memory": "8G",
                "spark.executor.memory": "8G",
                "spark.executor.instances": "2",

 "spark.hadoop.javax.jdo.option.ConnectionDriverName":"org.mariadb.jdbc.Driver",
                "spark.hadoop.javax.jdo.option.ConnectionURL":"jdbc:mysql://db-host:db-
port/db-name",
                "spark.hadoop.javax.jdo.option.ConnectionUserName":"connection-user-
name",
                "spark.hadoop.javax.jdo.option.ConnectionPassword":
 "EMR.secret@SecretID"
            }
        },
        {
            "classification": "spark-driver-log4j2",
            "properties": {
                "rootLogger.level":"error",
                "logger.IdentifierForClass.name": "classpathForSettingLogger",
                "logger.IdentifierForClass.level": "info"
```

```
            }
        }
    ]' \
    --monitoring-configuration '{
        "s3MonitoringConfiguration": {
            "logUri": "s3://amzn-s3-demo-logging-bucket/logs/app-level"
        },
        "managedPersistenceMonitoringConfiguration": {
            "enabled": false
        }
    }'
```

ジョブ実行中の設定の上書き

を使用して、アプリケーション設定とモニタリング設定の設定オーバーライドを指定できますStartJobRunAPI。EMR 次に、サーバーレスはアプリケーションレベルとジョブレベルで指定した設定をマージして、ジョブ実行の設定を決定します。

マージが発生したときの粒度レベルは次のとおりです。

- **ApplicationConfiguration** - 分類タイプ。例: spark-defaults。
- **MonitoringConfiguration** - 設定タイプ。例: s3MonitoringConfiguration。

> **Note**
>
> で提供する設定の優先度は、アプリケーションレベルで提供する設定よりもStartJobRun優先されます。

優先順位ランキングの詳細については、Hive 設定オーバーライドパラメータ「」および「」を参照してくださいSpark 設定オーバーライドパラメータ。

ジョブを開始するときに、特定の設定を指定しない場合、アプリケーションから継承されます。ジョブレベルで設定を宣言する場合は、次の操作を実行できます。

- 既存の設定を上書きする - オーバーライド値を使用して、StartJobRunリクエスト内の同じ設定パラメータを指定します。
- 追加設定の追加 - リクエストに、指定する値StartJobRunを含む新しい設定パラメータを追加します。

- 既存の設定を削除する - アプリケーションランタイム設定 を削除するには、削除する設定のキーを指定し、設定{}の空の宣言を渡します。ジョブの実行に必要なパラメータを含む分類を削除することはお勧めしません。例えば、Hive ジョブ に必要なプロパティを削除しようとすると、ジョブは失敗します。

  アプリケーションモニタリング設定 を削除するには、関連する設定タイプに適した方法を使用します。

  - **cloudWatchLoggingConfiguration** - を削除するにはcloudWatchLogging、有効なフラグを として渡しますfalse。
  - **managedPersistenceMonitoringConfiguration** - マネージド永続化設定を削除してデフォルトの有効状態にフォールバックするには、設定{}に空の宣言を渡します。
  - **s3MonitoringConfiguration** - を削除するにはs3MonitoringConfiguration、設定{}の空の宣言を渡します。

オーバーライドの例

次の例は、 でのジョブ送信中に実行できるさまざまなオペレーションを示していますstart-job-run。

```
aws emr-serverless start-job-run \
    --application-id your-application-id \
    --execution-role-arn your-job-role-arn \
    --job-driver '{
        "sparkSubmit": {
            "entryPoint": "s3://us-east-1.elasticmapreduce/emr-containers/samples/
wordcount/scripts/wordcount.py",
            "entryPointArguments": ["s3://amzn-s3-demo-destination-bucket1/
wordcount_output"]
        }
    }' \
    --configuration-overrides '{
        "applicationConfiguration": [
            {
                // Override existing configuration for spark-defaults in the
 application
                "classification": "spark-defaults",
                "properties": {
                    "spark.driver.cores": "2",
                    "spark.executor.cores": "1",
                    "spark.driver.memory": "4G",
```

```
                    "spark.executor.memory": "4G"
                }
            },
            {
                // Add configuration for spark-executor-log4j2
                "classification": "spark-executor-log4j2",
                "properties": {
                    "rootLogger.level": "error",
                    "logger.IdentifierForClass.name": "classpathForSettingLogger",
                    "logger.IdentifierForClass.level": "info"
                }
            },
            {
                // Remove existing configuration for spark-driver-log4j2 from the
application
                "classification": "spark-driver-log4j2",
                "properties": {}
            }
        ],
        "monitoringConfiguration": {
            "managedPersistenceMonitoringConfiguration": {
                // Override existing configuration for managed persistence
                "enabled": true
            },
            "s3MonitoringConfiguration": {
                // Remove configuration of S3 monitoring
            },
            "cloudWatchLoggingConfiguration": {
                // Add configuration for CloudWatch logging
                "enabled": true
            }
        }
    }'
```

ジョブの実行時に、 および で説明されている優先度オーバーライドランキングに基づいて、次の分類Hive 設定オーバーライドパラメータと設定が適用されますSpark 設定オーバーライドパラメータ。

- 分類spark-defaultsは、ジョブレベルで指定されたプロパティで更新されます。この分類では、 に含まれるプロパティのみが考慮StartJobRunされます。
- 分類は、既存の分類リストに追加spark-executor-log4j2されます。
- 分類spark-driver-log4j2は削除されます。

- の設定managedPersistenceMonitoringConfigurationは、ジョブレベルの設定で更新されます。
- の設定s3MonitoringConfigurationは削除されます。
- の設定は、既存のモニタリング設定に追加cloudWatchLoggingConfigurationされます。

EMR Serverless イメージのカスタマイズ

Amazon EMR 6.9.0 以降では、カスタムイメージを使用して、アプリケーションの依存関係とランタイム環境を Amazon EMR Serverless の 1 つのコンテナにパッケージ化できます。これにより、ワークロードの依存関係の管理が簡単になり、パッケージの移植性が向上します。EMR Serverless イメージをカスタマイズすると、次の利点があります。

- ワークロードに最適化されたパッケージをインストールして設定します。これらのパッケージは、Amazon EMR ランタイム環境のパブリックディストリビューションでは広く利用できない場合があります。
- EMR Serverless を、ローカル開発やテストなど、組織内で現在確立されている構築、テスト、デプロイプロセスと統合します。
- 組織内のコンプライアンスとガバナンス要件を満たす、イメージスキャンなどの確立されたセキュリティプロセスを適用します。
- 独自のバージョンの JDKと Python をアプリケーションに使用できます。

EMR Serverless は、独自のイメージを作成するときにベースとして使用できるイメージを提供します。ベースイメージは、イメージが EMR Serverless とやり取りするための必須のジャー、設定、ライブラリを提供します。ベースイメージは Amazon ECR Public Gallery にあります。アプリケーションタイプ (Spark または Hive) とリリースバージョンに一致するイメージを使用します。例えば、Amazon EMRリリース 6.9.0 でアプリケーションを作成する場合は、次のイメージを使用します。

タイプ	イメージ
Spark	`public.ecr.aws/emr-serverless/spark/emr-6.9.0:latest`
[Hive]	`public.ecr.aws/emr-serverless/hive/emr-6.9.0:latest`

前提条件

EMR Serverless カスタムイメージを作成する前に、以下の前提条件を完了してください。

1. EMR Serverless アプリケーションの起動 AWS リージョン に使用するのと同じ に Amazon ECR リポジトリを作成します。Amazon ECRプライベートリポジトリを作成するには、「プライベートリポジトリの作成」を参照してください。
2. Amazon ECRリポジトリへのアクセスをユーザーに許可するには、このリポジトリからのイメージを使用して EMR Serverless アプリケーションを作成または更新するユーザーとロールに次のポリシーを追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ECRRepositoryListGetPolicy",
            "Effect": "Allow",
            "Action": [
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "ecr:DescribeImages"
            ],
            "Resource": "ecr-repository-arn"
        }
    ]
}
```

Amazon ECRアイデンティティベースのポリシーのその他の例については、「Amazon Elastic Container Registry アイデンティティベースのポリシーの例」を参照してください。

ステップ 1: EMR Serverless ベースイメージからカスタムイメージを作成する

まず、任意のベースイメージを使用するFROM命令で始まる Dockerfile を作成します。FROM 指示の後、イメージに加えたい変更を含めることができます。ベースイメージは、 を自動的に USER に設定しますhadoop。この設定には、含めるすべての変更に対するアクセス許可がない場合があります。回避策として、 USERを に設定しroot、イメージを変更してから、 を USERに戻しますhadoop:hadoop。一般的なユースケースのサンプルについては、「」を参照してくださいEMR Serverless でのカスタムイメージの使用。

```
# Dockerfile
FROM public.ecr.aws/emr-serverless/spark/emr-6.9.0:latest

USER root
# MODIFICATIONS GO HERE

# EMRS will run the image as hadoop
USER hadoop:hadoop
```

Dockerfile を取得したら、次のコマンドを使用してイメージを構築します。

```
# build the docker image
docker build . -t aws-account-id.dkr.ecr.region.amazonaws.com/my-
repository[:tag]or[@digest]
```

ステップ 2: イメージをローカルで検証する

EMR Serverless には、カスタムイメージを静的にチェックして、基本的なファイル、環境変数、および正しいイメージ設定を検証できるオフラインツールが用意されています。ツールをインストールして実行する方法については、「Amazon EMR Serverless ImageCLI GitHub」を参照してください。

ツールをインストールしたら、次のコマンドを実行してイメージを検証します。

```
amazon-emr-serverless-image \
validate-image -r emr-6.9.0 -t spark \
-i aws-account-id.dkr.ecr.region.amazonaws.com/my-repository:tag/@digest
```

次のような出力が表示されます。

```
Amazon EMR Serverless - Image CLI
Version: 0.0.1
... Checking if docker cli is installed
... Checking Image Manifest
[INFO] Image ID: 9e2f4359cf5beb466a8a2ed047ab61c9d37786c555655fc122272758f761b41a
[INFO] Created On: 2022-12-02T07:46:42.586249984Z
[INFO] Default User Set to hadoop:hadoop : PASS
[INFO] Working Directory Set to  : PASS
[INFO] Entrypoint Set to /usr/bin/entrypoint.sh : PASS
[INFO] HADOOP_HOME is set with value: /usr/lib/hadoop : PASS
[INFO] HADOOP_LIBEXEC_DIR is set with value: /usr/lib/hadoop/libexec : PASS
```

```
[INFO] HADOOP_USER_HOME is set with value: /home/hadoop : PASS
[INFO] HADOOP_YARN_HOME is set with value: /usr/lib/hadoop-yarn : PASS
[INFO] HIVE_HOME is set with value: /usr/lib/hive : PASS
[INFO] JAVA_HOME is set with value: /etc/alternatives/jre : PASS
[INFO] TEZ_HOME is set with value: /usr/lib/tez : PASS
[INFO] YARN_HOME is set with value: /usr/lib/hadoop-yarn : PASS
[INFO] File Structure Test for hadoop-files in /usr/lib/hadoop: PASS
[INFO] File Structure Test for hadoop-jars in /usr/lib/hadoop/lib: PASS
[INFO] File Structure Test for hadoop-yarn-jars in /usr/lib/hadoop-yarn: PASS
[INFO] File Structure Test for hive-bin-files in /usr/bin: PASS
[INFO] File Structure Test for hive-jars in /usr/lib/hive/lib: PASS
[INFO] File Structure Test for java-bin in /etc/alternatives/jre/bin: PASS
[INFO] File Structure Test for tez-jars in /usr/lib/tez: PASS
----------------------------------------------------------------
Overall Custom Image Validation Succeeded.
----------------------------------------------------------------
```

ステップ 3: Amazon ECRリポジトリにイメージをアップロードする

次のコマンドを使用して、Amazon ECRイメージを Amazon ECRリポジトリにプッシュします。イメージをリポジトリにプッシュするための正しいIAMアクセス許可があることを確認します。詳細については、「Amazon ECRユーザーガイド」の「画像のプッシュ」を参照してください。

```
# login to ECR repo
aws ecr get-login-password --region region | docker login --username AWS --password-
stdin aws-account-id.dkr.ecr.region.amazonaws.com

# push the docker image
docker push aws-account-id.dkr.ecr.region.amazonaws.com/my-repository:tag/@digest
```

ステップ 4: カスタムイメージを使用してアプリケーションを作成または更新する

アプリケーションを起動する方法に従って AWS Management Console タブまたは AWS CLI タブを選択し、次のステップを実行します。

Console

1. https://console.aws.amazon.com/emr で EMR Studio コンソールにサインインします。アプリケーションに移動するか、「アプリケーションの作成」の手順で新しいアプリケーションを作成します。

2. EMR Serverless アプリケーションを作成または更新するときにカスタムイメージを指定するには、アプリケーション設定オプションでカスタム設定を選択します。
3. カスタムイメージ設定 セクションで、このアプリケーションでカスタムイメージを使用するチェックボックスを選択します。
4. Amazon ECRイメージを Image URI フィールドに貼り付けURIます。EMR Serverless は、アプリケーションのすべてのワーカータイプにこのイメージを使用します。または、ワーカータイプごとに異なるカスタムイメージを選択し、異なる Amazon ECRイメージURIsを貼り付けることもできます。

CLI

- `image-configuration` パラメータを使用してアプリケーションを作成します。EMR Serverless は、この設定をすべてのワーカータイプに適用します。

```
aws emr-serverless create-application \
--release-label emr-6.9.0 \
--type SPARK \
--image-configuration '{
    "imageUri": "aws-account-id.dkr.ecr.region.amazonaws.com/my-repository:tag/
@digest"
}'
```

ワーカータイプごとに異なるイメージ設定を持つアプリケーションを作成するには、`worker-type-specifications`パラメータを使用します。

```
aws emr-serverless create-application \
--release-label emr-6.9.0 \
--type SPARK \
--worker-type-specifications '{
    "Driver": {
        "imageConfiguration": {
            "imageUri": "aws-account-id.dkr.ecr.region.amazonaws.com/my-
repository:tag/@digest"
        }
    },
    "Executor" : {
        "imageConfiguration": {
            "imageUri": "aws-account-id.dkr.ecr.region.amazonaws.com/my-
repository:tag/@digest"
        }
```

```
    }
}'
```

アプリケーションを更新するには、 image-configurationパラメータを使用します。EMR Serverless は、この設定をすべてのワーカータイプに適用します。

```
aws emr-serverless update-application \
--application-id application-id \
--image-configuration '{
    "imageUri": "aws-account-id.dkr.ecr.region.amazonaws.com/my-repository:tag/
@digest"
}'
```

ステップ 5: EMR Serverless にカスタムイメージリポジトリへのアクセスを許可する

次のリソースポリシーを Amazon ECRリポジトリに追加して、EMRサーバーレスサービスプリンシパルがこのリポジトリからの get、describe、および downloadリクエストを使用できるようにします。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Emr Serverless Custom Image Support",
      "Effect": "Allow",
      "Principal": {
        "Service": "emr-serverless.amazonaws.com"
      },
      "Action": [
        "ecr:BatchGetImage",
        "ecr:DescribeImages",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Condition":{
        "StringEquals":{
          "aws:SourceArn": "arn:aws:emr-serverless:region:aws-account-id:/
applications/application-id"
        }
      }
```

```
    }
  ]
}
```

セキュリティのベストプラクティスとして、リポジトリポリシーにaws:SourceArn条件キーを追加します。IAM グローバル条件キーaws:SourceArnにより、EMRServerless はアプリケーション にのみリポジトリを使用しますARN。Amazon ECRリポジトリポリシーの詳細については、「プライベートリポジトリの作成」を参照してください。

考慮事項と制約事項

カスタムイメージを使用する場合は、次の点を考慮してください。

- アプリケーションのタイプ (Spark または Hive) とリリースラベル (などemr-6.9.0) に一致する正しいベースイメージを使用します。
- EMR Serverless は、Docker ファイル内の [CMD]または [ENTRYPOINT]の指示を無視します。、、 など[COPY][RUN]、Docker ファイルで一般的な手順を使用します[WORKDIR]。
- カスタムイメージを作成するTEZ_HOMEときはSPARK_HOME、環境変数 JAVA_HOME、HIVE_HOME、 を変更しないでください。
- カスタムイメージのサイズは 5 GB を超えることはできません。
- Amazon EMRベースイメージのバイナリまたはジャーを変更すると、アプリケーションまたはジョブの起動が失敗する可能性があります。
- Amazon ECRリポジトリは、EMRサーバーレスアプリケーションの起動に使用する AWS リージョン ものと同じ にある必要があります。

VPC アクセスの設定

Amazon Redshift クラスターVPC、Amazon RDS データベース、VPCエンドポイントを含む Amazon S3 バケットなど、 内のデータストアに接続するように EMR Serverless アプリケーションを設定できます。EMR Serverless アプリケーションは、 内のデータストアにアウトバウンド接続しますVPC。デフォルトでは、EMRサーバーレスはアプリケーションへのインバウンドアクセスをブロックしてセキュリティを向上させます。

> **Note**
>
> アプリケーションに外部 Hive メタストアデータベースを使用する場合は、VPCアクセスを設定する必要があります。外部 Hive メタストアを設定する方法については、「メタストア設定」を参照してください。

アプリケーションの作成

アプリケーションの作成ページで、カスタム設定を選択し、EMRサーバーレスアプリケーションで使用できる VPC、サブネット、セキュリティグループを指定できます。

VPCs

データストアを含む仮想プライベートクラウド (VPC) の名前を選択します。アプリケーションの作成ページには、選択した VPCsのすべての が一覧表示されます AWS リージョン。

サブネット

データストアVPCを含む 内のサブネットを選択します。アプリケーションの作成ページには、 のデータストアのすべてのサブネットが一覧表示されますVPC。

選択したサブネットはプライベートサブネットである必要があります。つまり、サブネットの関連付けられたルートテーブルにインターネットゲートウェイがあってはなりません。

インターネットへのアウトバウンド接続では、サブネットに NAT Gateway を使用したアウトバウンドルートが必要です。NAT ゲートウェイを設定するには、NAT「ゲートウェイの操作」を参照してください。

Amazon S3 接続の場合、サブネットにはNATゲートウェイまたはVPCエンドポイントが設定されている必要があります。S3 VPCエンドポイントを設定するには、「ゲートウェイエンドポイントの作成」を参照してください。

Amazon DynamoDB などの VPC AWS のサービス 以外の への接続には、VPCエンドポイントまたはNATゲートウェイを設定する必要があります。のVPCエンドポイントを設定するには AWS のサービス、VPC「エンドポイントの操作」を参照してください。

ワーカーは、アウトバウンドトラフィックVPCを介して 内のデータストアに接続できます。デフォルトでは、EMRサーバーレスはセキュリティを向上させるためにワーカーへのインバウンドアクセスをブロックします。

を使用すると AWS Config、EMRServerless はすべてのワーカーのエラスティックネットワークインターフェイス項目レコードを作成します。このリソースに関連するコストを回避するには、`AWS::EC2::NetworkInterface`でオフにすることを検討してください AWS Config。

> **Note**
>
> 複数のアベイラビリティーゾーンで複数のサブネットを選択することをお勧めします。これは、選択したサブネットによって、EMRサーバーレスアプリケーションの起動に使用できるアベイラビリティーゾーンが決定されるためです。各ワーカーは、起動するサブネットで IP アドレスを使用します。指定されたサブネットに、起動するワーカーの数に十分な IP アドレスがあることを確認してください。サブネット計画の詳細については、「」を参照してくださいthe section called “サブネット計画のベストプラクティス”。

セキュリティグループ

データストアと通信できる 1 つ以上のセキュリティグループを選択します。アプリケーションの作成ページには、 内のすべてのセキュリティグループが一覧表示されますVPC。EMR Serverless は、これらのセキュリティグループをVPCサブネットにアタッチされたエラスティックネットワークインターフェイスに関連付けます。

> **Note**
>
> EMR Serverless アプリケーション用に別のセキュリティグループを作成することをお勧めします。これにより、ネットワークルールの分離と管理がより効率的になります。例えば、Amazon Redshift クラスターと通信するには、以下の例に示すように、Redshift と EMR Serverless セキュリティグループ間のトラフィックルールを定義できます。

Example 例 — Amazon Redshift クラスターとの通信

1. EMR Serverless セキュリティグループの 1 つから Amazon Redshift セキュリティグループへのインバウンドトラフィックのルールを追加します。

タイプ	プロトコル	ポート範囲	ソース
すべて TCP	TCP	5439	`emr-serverless-security-group`

2. EMR Serverless セキュリティグループの 1 つからのアウトバウンドトラフィックのルールを追加します。これには以下の 2 つの方法があります。まず、すべてのポートへのアウトバウンドトラフィックを開くことができます。

タイプ	プロトコル	ポート範囲	デスティネーション
すべてのトラフィック	TCP	ALL	0.0.0.0/0

または、アウトバウンドトラフィックを Amazon Redshift クラスターに制限することもできます。これは、アプリケーションが Amazon Redshift クラスターと通信する必要がある場合にのみ役立ちます。

タイプ	プロトコル	ポート範囲	ソース
すべて TCP	TCP	5439	`redshift-security-group`

アプリケーションを設定する

既存の EMR Serverless アプリケーションのネットワーク設定は、アプリケーションの設定ページから変更できます。

ジョブ実行の詳細を表示する

ジョブ実行の詳細ページで、ジョブが特定の実行に使用するサブネットを表示できます。ジョブは、指定されたサブネットから選択された 1 つのサブネットでのみ実行されることに注意してください。

サブネット計画のベストプラクティス

AWS リソースは、Amazon で使用可能な IP アドレスのサブセットであるサブネットに作成されますVPC。例えば、/16 ネットマスクVPCを持つ には最大 65,536 個の使用可能な IP アドレスがあり、サブネットマスクを使用して複数の小さなネットワークに分割できます。例えば、この範囲を 2 つのサブネットに分割し、それぞれに /17 マスクと 32,768 個の使用可能な IP アドレスを使用できます。サブネットはアベイラビリティーゾーン内にあり、ゾーン間でまたがることはできません。

サブネットは、EMRサーバーレスアプリケーションのスケーリング制限を考慮して設計する必要があります。例えば、4 人の vCpu ワーカーをリクエストするアプリケーションがあり、最大 4,000 までスケールできる場合vCpu、アプリケーションには合計 1,000 のネットワークインターフェイスに対して最大 1,000 人のワーカーが必要です。複数のアベイラビリティーゾーンにサブネットを作成することをお勧めします。これにより、EMRサーバーレスは、アベイラビリティーゾーンに障害が発生した場合に、万一、ジョブを再試行したり、別のアベイラビリティーゾーンに事前初期化されたキャパシティーをプロビジョニングしたりできます。したがって、少なくとも 2 つのアベイラビリティーゾーンの各サブネットには、1,000 を超える使用可能な IP アドレスが必要です。

1,000 個のネットワークインターフェイスをプロビジョニングするには、マスクサイズが 22 以下のサブネットが必要です。22 を超えるマスクは要件を満たしません。例えば、サブネットマスクの /23 は 512 IP アドレスを提供し、マスクの /22 は 1024、マスクの /21 は 2048 IP アドレスを提供します。以下は、異なるアベイラビリティーゾーンに割り当てることができる /16 ネットマスクVPCの に /22 マスクを持つ 4 つのサブネットの例です。各サブネットの最初の 4 つの IP アドレスと最後の IP アドレスは によって予約されるため、使用可能な IP アドレスと使用可能な IP アドレスには 5 つの違いがあります AWS。

サブネット ID	サブネットアドレス	サブネットマスク	IP アドレス範囲	使用可能な IP アドレス	使用可能な IP アドレス
1	10.0.0.0	255.255.252.0/22	10.0.0.0 ~ 10.0.3.255	1,024	1,019
2	10.0.4.0	255.255.252.0/22	10.0.4.0 ~ 10.0.7.255	1,024	1,019
3	10.0.8.0	255.255.252.0/22	10.0.4.0 ~ 10.0.7.255	1,024	1,019

サブネット ID	サブネットアドレス	サブネットマスク	IP アドレス範囲	使用可能な IP アドレス	使用可能な IP アドレス
4	10.0.12.0	255.255.252.0/22	10.0.12.0 ~ 10.0.15.255	1,024	1,019

ワークロードがより大きなワーカーサイズに適しているかどうかを評価する必要があります。ワーカーサイズを大きくすると、ネットワークインターフェイスが少なくなります。例えば、アプリケーションスケーリング制限が 4,000 の 16 人のvCpu ワーカーを使用する場合、ネットワークインターフェイスをプロビジョニングするために、最大 250 人のワーカーが必要になり、合計 250 の使用可能な IP アドレス vCpu が必要になります。250 ネットワークインターフェイスをプロビジョニングするには、マスクサイズが 24 以下の複数のアベイラビリティーゾーンにサブネットが必要です。24 を超えるマスクサイズは、250 未満の IP アドレスを提供します。

複数のアプリケーション間でサブネットを共有する場合、各サブネットは、すべてのアプリケーションの集合的なスケーリング制限を念頭に置いて設計する必要があります。例えば、4 人の vCpu ワーカーをリクエストする 3 つのアプリケーションがあり、それぞれが 12,000 個のvCpuアカウントレベルのサービスベースのクォータ vCpu で 4,000 までスケールできる場合、各サブネットには 3,000 個の使用可能な IP アドレスが必要です。VPC 使用する に十分な数の IP アドレスがない場合は、使用可能な IP アドレスの数を増やしてみてください。これを行うには、追加のクラスレスドメイン間ルーティング (CIDR) ブロックを に関連付けますVPC。詳細については、「Amazon VPCユーザーガイド」の「追加のIPv4CIDRブロックを に関連付けるVPC」を参照してください。

オンラインで利用可能な多数のツールのいずれかを使用して、サブネット定義をすばやく生成し、使用可能な IP アドレスの範囲を確認できます。

Amazon EMR Serverless アーキテクチャオプション

Amazon EMR Serverless アプリケーションの命令セットアーキテクチャは、アプリケーションがジョブの実行に使用するプロセッサのタイプを決定します。Amazon EMRには、x86_64 と arm64 の 2 つのアーキテクチャオプションがあります。EMR サーバーレスは、利用可能になると最新世代のインスタンスに自動的に更新されるため、アプリケーションは新しいインスタンスを、追加の労力を必要とせずに使用できます。

トピック

- x86_64 アーキテクチャの使用
- arm64 アーキテクチャの使用 (Graviton)

- Graviton サポートによる新しいアプリケーションの起動
- Graviton を使用するように既存のアプリケーションを設定する
- Graviton を使用する際の考慮事項

x86_64 アーキテクチャの使用

x86_64 アーキテクチャは、x86 64 ビットまたは x64 とも呼ばれます。x86_64 はEMRサーバーレスアプリケーションのデフォルトオプションです。このアーキテクチャは x86 ベースのプロセッサを使用し、ほとんどのサードパーティーのツールやライブラリと互換性があります。

ほとんどのアプリケーションは x86 ハードウェアプラットフォームと互換性があり、デフォルトの x86_64 アーキテクチャで正常に実行できます。ただし、アプリケーションが 64 ビット と互換性がある場合はARM、arm64 に切り替えて Graviton プロセッサを使用してパフォーマンス、コンピューティングパワー、メモリを向上させることができます。x86 アーキテクチャで同じサイズのインスタンスを実行する場合よりも、arm64 アーキテクチャでインスタンスを実行する方がコストがかかります。

arm64 アーキテクチャの使用 (Graviton)

AWS Graviton プロセッサは、64 ビット ARM Neoverse コア AWS を使用して によってカスタム設計され、arm64 アーキテクチャ (Arch64 または 64 ビット とも呼ばれます) を活用します ARM。EMR Serverless で使用できる AWS Graviton プロセッサのラインには、Graviton3 プロセッサと Graviton2 プロセッサが含まれます。これらのプロセッサは、x86_64 アーキテクチャで実行される同等のワークロードと比較して、Spark および Hive ワークロードに対して優れた価格パフォーマンスを提供します。EMR Serverless は、最新世代のプロセッサにアップデートするために、ユーザー側から何も必要とせずに、利用可能な場合、最新世代のプロセッサを自動的に使用します。

Graviton サポートによる新しいアプリケーションの起動

arm64 アーキテクチャを使用するアプリケーションを起動するには、次のいずれかの方法を使用します。

AWS CLI

から Graviton プロセッサを使用してアプリケーションを起動するには AWS CLI、 の `architecture`パラメータ`ARM64`として `create-application` を指定しますAPI。他のパラメータでアプリケーションに適した値を指定します。

```
aws emr-serverless create-application \
 --name my-graviton-app \
 --release-label emr-6.8.0 \
 --type "SPARK" \
 --architecture "ARM64" \
 --region us-west-2
```

EMR Studio

EMR Studio から Graviton プロセッサを使用してアプリケーションを起動するには、アプリケーションを作成または更新するときにアーキテクチャオプションとして arm64 を選択します。

Graviton を使用するように既存のアプリケーションを設定する

SDK、 AWS CLI、または EMR Studio で Graviton (arm64) アーキテクチャを使用するように既存の Amazon EMR Serverless アプリケーションを設定できます。

既存のアプリケーションを x86 から arm64 に変換するには

1. `architecture` パラメータをサポートする AWS CLI/SDK の最新バージョンを使用していることを確認します。
2. 実行中のジョブがないことを確認し、アプリケーションを停止します。

```
aws emr-serverless stop-application \
 --application-id application-id \
 --region us-west-2
```

3. Graviton を使用するようにアプリケーションを更新するには、 の ARM64 `architecture`パラメータに `update-application` を指定しますAPI。

```
aws emr-serverless update-application \
 --application-id application-id \
 --architecture 'ARM64' \
 --region us-west-2
```

4. アプリケーションのCPUアーキテクチャが になったことを確認するにはARM64、 `get-application` を使用しますAPI。

```
aws emr-serverless get-application \
 --application-id application-id \
```

```
 --region us-west-2
```

5. 準備ができたら、アプリケーションを再起動します。

```
aws emr-serverless start-application \
 --application-id application-id \
 --region us-west-2
```

Graviton を使用する際の考慮事項

arm64 for Graviton サポートを使用して EMR Serverless アプリケーションを起動する前に、以下を確認してください。

ライブラリの互換性

アーキテクチャオプションとして Graviton (arm64) を選択する場合は、サードパーティーのパッケージとライブラリが 64 ビットARMアーキテクチャと互換性があることを確認してください。選択したアーキテクチャと互換性のある Python 仮想環境に Python ライブラリをパッケージ化する方法については、「」を参照してくださいServerless での Python EMR ライブラリの使用。

64 ビット を使用するように Spark または Hive ワークロードを設定する方法の詳細についてはARM、 の AWS Graviton 入門リポジトリを参照してください GitHub。このリポジトリには、 ARM ベースの Graviton の使用開始に役立つ重要なリソースが含まれています。

ジョブの同時実行とキューイング

Amazon EMRバージョン 7.0.0 以降では、アプリケーションのジョブ実行キューのタイムアウトと同時実行設定を指定できます。この設定を指定すると、Amazon EMR Serverless はジョブをキューに入れ、アプリケーションの同時実行使用率に基づいて実行を開始します。例えば、ジョブ実行の同時実行が 10 の場合、アプリケーションでは一度に 10 個のジョブのみが実行されます。残りのジョブは、実行中のジョブのいずれかが終了するまでキューに入れられます。キューのタイムアウトが早まると、ジョブはタイムアウトします。詳細については、「ジョブ実行状態」を参照してください。

同時実行とキューイングの主な利点

ジョブの同時実行とキューイングは、多くのジョブ送信が必要な場合に次の利点を提供します。

- これにより、同時実行ジョブを制御して、アプリケーションレベルの容量制限を効率的に使用できます。

- キューには、設定可能なタイムアウト設定で、ジョブ送信の突然のバーストを含めることができます。

同時実行とキューイングの開始方法

次の手順では、同時実行とキューイングを実装するいくつかの方法を示します。

の使用 AWS CLI

1. キュータイムアウトと同時ジョブ実行を使用して Amazon EMR Serverless アプリケーションを作成します。

```
aws emr-serverless create-application \
--release-label emr-7.0.0 \
--type SPARK \
--scheduler-configuration '{"maxConcurrentRuns": 1, "queueTimeoutMinutes": 30}'
```

2. アプリケーションを更新して、ジョブキューのタイムアウトと同時実行を変更します。

```
aws emr-serverless update-application \
--application-id application-id \
--scheduler-configuration '{"maxConcurrentRuns": 5, "queueTimeoutMinutes": 30}'
```

> **Note**
>
> 既存のアプリケーションを更新して、ジョブの同時実行とキューを有効にすることができます。これを行うには、アプリケーションにリリースラベル emr-7.0.0 以降が必要です。

の使用 AWS Management Console

次の手順では、 を使用してジョブの同時実行とキューイングを開始する方法を示します AWS Management Console。

1. EMR Studio に移動し、リリースラベル EMR-7.0.0 以降でアプリケーションを作成することを選択します。
2. アプリケーションセットアップオプション で、カスタム設定を使用する オプションを選択します。

3. その他の設定には、ジョブ実行設定 のセクションがあります。オプション ジョブの同時実行を有効にする を選択して、機能を有効にします。
4. 選択したら、同時ジョブ実行とキュータイムアウトの両方を選択して、それぞれ同時ジョブ実行とキュータイムアウトの数を設定できます。これらの設定に値を入力しない場合、デフォルト値が使用されます。
5. アプリケーションの作成を選択すると、この機能を有効にしてアプリケーションが作成されます。確認するには、ダッシュボードに移動し、アプリケーションを選択し、プロパティタブで機能が有効になっているかどうかを確認します。

設定後、この機能を有効にしてジョブを送信できます。

同時実行とキューイングに関する考慮事項

同時実行とキューイングを実装する場合は、次の点を考慮してください。

- ジョブキューと同時実行は、Amazon EMRリリース 7.0.0 以降でサポートされています。
- STARTED 状態のアプリケーションの同時実行を更新できます。
- の有効な範囲`maxConcurrentRuns`は 1 ~ 1000 で、 `queueTimeoutMinutes` の有効な範囲は 15 ~ 720 です。
- アカウントでは、最大 2000 個のジョブを QUEUED状態にできます。
- 同時実行とキューイングは、バッチジョブとストリーミングジョブに適用されます。インタラクティブジョブには使用できません。詳細については、EMR「Studio を通じて EMR Serverless でインタラクティブワークロードを実行する」を参照してください。

Serverless を使用して S3 Express One Zone にデータEMRを取得する

Amazon EMRリリース 7.2.0 以降では、Amazon S3 Express One Zone ストレージクラスで EMR Serverless を使用して、ジョブとワークロードを実行する際のパフォーマンスを向上させることができます。S3 Express One Zone は、最もレイテンシーの影響を受けやすいアプリケーションに一貫した 1 桁ミリ秒のデータアクセスを提供する、高性能の単一ゾーン Amazon S3 ストレージクラスです。リリース時点で、S3 Express One Zone は、Amazon S3 の中でレイテンシーが最も低く、パフォーマンスの最も高いクラウドオブジェクトストレージを提供しています。

前提条件

- S3 Express One Zone のアクセス許可 – S3 Express One Zone が最初に `GET`、`LIST`、または などのアクションを S3 オブジェクトPUTに対して実行すると、ストレージクラスは`CreateSession`ユーザーに代わって を呼び出します。IAM ポリシーでは、 が `s3express:CreateSession`にアクセス許可を付与する必要があります。S3A コネクタは `CreateSession` を呼び出すことができますAPI。このアクセス許可ポリシーの例については、「S3 Express One Zone の使用を開始する」を参照してください。
- S3A コネクタ — Amazon S3 S3 バケットのデータにアクセスするように Spark を設定するには、Apache Hadoop コネクタを使用する必要があります。S3A。 コネクタを使用するには、すべての S3 が `s3a`スキームURIsを使用していることを確認します。使用していない場合は、`s3` スキームと `s3n` スキーム用にファイルシステム実装を変更してください。

`s3` スキームを変更するには、以下のクラスター設定を指定します。

```
[
  {
    "Classification": "core-site",
    "Properties": {
      "fs.s3.impl": "org.apache.hadoop.fs.s3a.S3AFileSystem",
      "fs.AbstractFileSystem.s3.impl": "org.apache.hadoop.fs.s3a.S3A"
    }
  }
]
```

`s3n` スキームを変更するには、以下のクラスター設定を指定します。

```
[
  {
    "Classification": "core-site",
    "Properties": {
      "fs.s3n.impl": "org.apache.hadoop.fs.s3a.S3AFileSystem",
      "fs.AbstractFileSystem.s3n.impl": "org.apache.hadoop.fs.s3a.S3A"
    }
  }
]
```

S3 Express One Zone の使用を開始する

S3 Express One Zone の使用を開始するには、次の手順に従います。

1. VPC エンドポイント を作成します。エンドポイントをVPCエンドポイント `com.amazonaws.us-west-2.s3express`に追加します。
2. Amazon EMR Serverless の開始方法に従って、Amazon EMRリリースラベル 7.2.0 以降でアプリケーションを作成します。
3. 新しく作成されたVPCエンドポイント、プライベートサブネットグループ、およびセキュリティグループを使用するようにアプリケーションを設定します。
4. ジョブ実行ロールに アクセス`CreateSession`許可を追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Resource": "*",
            "Action": [
                "s3express:CreateSession"
            ]
        }
    ]
}
```

5. ジョブを実行します。S3A スキームを使用して S3 Express One Zone バケットにアクセスする必要があることに注意してください。

```
aws emr-serverless start-job-run \
```

```
--application-id <application-id> \
--execution-role-arn <job-role-arn> \
--name <job-run-name> \
--job-driver '{
 "sparkSubmit": {


 "entryPoint": "s3a://<DOC-EXAMPLE-BUCKET>/scripts/wordcount.py",
 "entryPointArguments":["s3a://<DOC-EXAMPLE-BUCKET>/emr-serverless-spark/output"],
 "sparkSubmitParameters": "--conf spark.executor.cores=4
 --conf spark.executor.memory=8g --conf spark.driver.cores=4
 --conf spark.driver.memory=8g --conf spark.executor.instances=2
 --conf spark.hadoop.fs.s3a.change.detection.mode=none
 --conf spark.hadoop.fs.s3a.endpoint.region={<AWS_REGION>}
 --conf spark.hadoop.fs.s3a.select.enabled=false
 --conf spark.sql.sources.fastS3PartitionDiscovery.enabled=false
 }'
```

ジョブの実行

アプリケーションをプロビジョニングしたら、アプリケーションにジョブを送信できます。このセクションでは、 を使用する方法について説明します。 AWS CLI これらのジョブを実行する 。このセクションでは、EMRServerless で使用できる各タイプのアプリケーションのデフォルト値も識別します。

トピック

ジョブ実行状態

Amazon EMR Serverless ジョブキューにジョブ実行を送信すると、ジョブ実行は SUBMITTED状態になります。ジョブ状態の は、SUBMITTED、FAILED、SUCCESSまたは に達するRUNNINGまで から に渡されますCANCELLING。

ジョブ実行の各状態は以下のとおりです。

状態	説明
送信済み	EMR Serverless にジョブ実行を送信するときの初期ジョブ状態。ジョブはアプリケーションのスケジュールを待機します。EMR Serverles

状態	説明
	s はジョブ実行の優先順位付けとスケジュールを開始します。
キューに登録済み	ジョブ実行は、アプリケーションレベルのジョブ実行の同時実行が完全に占有されると、この状態で待機します。キューイングと同時実行の詳細については、「」を参照してくださいジョブの同時実行とキューイング。
[保留中]	スケジューラは、ジョブ実行を評価して、アプリケーションの実行を優先してスケジュールします。
スケジュール済み	EMR Serverless は、アプリケーションのジョブ実行をスケジュールし、ジョブを実行するリソースを割り当てています。
実行中	EMR Serverless は、ジョブが最初に必要とするリソースを割り当て、ジョブはアプリケーションで実行されています。Spark アプリケーションでは、これは Spark ドライバープロセスが `running` 状態にあることを意味します。
[失敗]	EMR サーバーレスは、ジョブ実行をアプリケーションに送信できなかったか、正常に完了しませんでした。このジョブの失敗の詳細についてはStateDetails 、「」を参照してください。
成功	ジョブの実行が正常に完了しました。
[キャンセル中]	CancelJobRun API がジョブ実行のキャンセルをリクエストしたか、ジョブ実行がタイムアウトしました。EMR Serverless は、アプリケーションのジョブをキャンセルしてリソースを解放しようとしています。

状態	説明
キャンセル	ジョブの実行は正常にキャンセルされ、使用したリソースがリリースされました。

EMR Studio コンソールからのジョブの実行

ジョブ実行を EMR Serverless アプリケーションに送信し、EMRStudio コンソールからジョブを表示できます。EMR Studio コンソールで EMR Serverless アプリケーションを作成または移動するには、「コンソールから開始する」の手順に従います。

ジョブを送信する

「ジョブの送信」ページで、次のようにジョブをEMRサーバーレスアプリケーションに送信できます。

Spark

1. 名前 フィールドに、ジョブ実行の名前を入力します。
2. ランタイムロール フィールドに、EMRサーバーレスアプリケーションがジョブ実行のために引き受けることができるIAMロールの名前を入力します。ランタイムロールの詳細については、「」を参照してくださいAmazon EMR Serverless のジョブランタイムロール。
3. スクリプトの場所 フィールドに、実行するスクリプトまたは の Amazon S3 JAR の場所を入力します。Spark ジョブの場合、スクリプトは Python (`.py`) ファイルまたは JAR (`.jar`) ファイルです。
4. スクリプトの場所が JAR ファイルの場合は、ジョブのエントリポイントであるクラス名を Main class フィールドに入力します。
5. (オプション) 残りのフィールドに値を入力します。

   - スクリプト引数 — メインJARスクリプトまたは Python スクリプトに渡す引数を入力します。コードはこれらのパラメータを読み取ります。配列内の各引数をカンマで区切ります。
   - Spark プロパティ — Spark プロパティセクションを展開し、このフィールドに Spark 設定パラメータを入力します。

> **Note**
>
> Spark ドライバーとエグゼキューターのサイズを指定する場合は、メモリのオーバーヘッドを考慮する必要があります。プロパティ `spark.driver.memoryOverhead`および でメモリオーバーヘッド値を指定しますspark.executor.memoryOverhead。メモリオーバーヘッドのデフォルト値はコンテナメモリの 10% で、最低 384 MB です。エグゼキューターメモリとメモリオーバーヘッドは、一緒にワーカーメモリを超えることはできません。例えば、30 GB ワーカーspark.executor.memoryの最大は 27 GB である必要があります。

- ジョブ設定 - このフィールドにジョブ設定を指定します。これらのジョブ設定を使用して、アプリケーションのデフォルト設定を上書きできます。
- 追加設定 — を有効または無効にする AWS Glue Data Catalog をメタストアとして作成し、アプリケーションログ設定を変更します。メタストア設定の詳細については、「」を参照してくださいメタストア設定。アプリケーションの口グ記録オプションの詳細については、「」を参照してくださいログの保存。
- タグ — アプリケーションにカスタムタグを割り当てます。

6. [Submit job] (ジョブの送信) を選択します。

Hive

1. 名前 フィールドに、ジョブ実行の名前を入力します。
2. ランタイムロール フィールドに、EMRサーバーレスアプリケーションがジョブ実行のために引き受けることができるIAMロールの名前を入力します。
3. スクリプトの場所 フィールドに、実行するスクリプトまたは の Amazon S3 JAR の場所を入力します。Hive ジョブの場合、スクリプトは Hive (`.sql`) ファイルである必要があります。
4. (オプション) 残りのフィールドに値を入力します。

   - 初期化スクリプトの場所 – Hive スクリプトが実行される前にテーブルを初期化するスクリプトの場所を入力します。
   - Hive プロパティ – Hive プロパティセクションを展開し、このフィールドに Hive 設定パラメータを入力します。
   - ジョブ設定 — 任意のジョブ設定を指定します。これらのジョブ設定を使用して、アプリケーションのデフォルト設定を上書きできます。Hive ジョブの場合、

hive.exec.scratchdirおよび hive.metastore.warehouse.dirはhive-site設定で必須のプロパティです。

```
{
    "applicationConfiguration": [
        {
            "classification": "hive-site",
            "configurations": [],
            "properties": {
                "hive.exec.scratchdir": "s3://DOC-EXAMPLE_BUCKET/hive/
scratch",
                "hive.metastore.warehouse.dir": "s3://DOC-EXAMPLE_BUCKET/hive/
warehouse"
            }
        }
    ],
    "monitoringConfiguration": {}
}
```

- 追加設定 — を有効または無効にする AWS Glue Data Catalog をメタストアとして作成し、アプリケーションログ設定を変更します。メタストア設定の詳細については、「」を参照してくださいメタストア設定。アプリケーションのログ記録オプションの詳細については、「」を参照してくださいログの保存。
- タグ — アプリケーションにカスタムタグを割り当てます。

5. [Submit job] (ジョブの送信) を選択します。

ジョブ実行を表示する

アプリケーションの 詳細 ページの ジョブ実行 タブから、ジョブ実行を表示し、ジョブ実行に対して次のアクションを実行できます。

ジョブのキャンセル — RUNNING状態のジョブ実行をキャンセルするには、このオプションを選択します。ジョブ実行の移行の詳細については、「」を参照してくださいジョブ実行状態。

ジョブのクローン — 以前のジョブ実行のクローンを作成して再送信するには、このオプションを選択します。

からジョブを実行する AWS CLI

で個々のジョブを作成、説明、削除できます AWS CLI。すべてのジョブを一覧表示して、それらを一目で確認することもできます。

新しいジョブを送信するには、 を使用しますstart-job-run。実行するアプリケーションの ID と、ジョブ固有のプロパティを指定します。Spark の例については、「」を参照してくださいSpark ジョブ。Hive の例については、「」を参照してくださいHive ジョブ。このコマンドはapplication-id、、ARN、および新しい を返しますjob-id。

各ジョブの実行には、タイムアウト期間が設定されています。ジョブの実行がこの期間を超えると、EMRServerless は自動的にジョブをキャンセルします。デフォルトのタイムアウトは 12 時間です。ジョブの実行を開始するときに、このタイムアウト設定をジョブ要件を満たす値に設定できます。executionTimeoutMinutes プロパティを使用して 値を設定します。

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --execution-timeout-minutes 15 \
  --job-driver '{
    "hive": {
        "query": "s3://amzn-s3-demo-bucket/scripts/create_table.sql",
        "parameters": "--hiveconf hive.exec.scratchdir=s3://amzn-s3-demo-bucket/
hive/scratch --hiveconf hive.metastore.warehouse.dir=s3://amzn-s3-demo-bucket/hive/
warehouse"
    }
   }' \
  --configuration-overrides '{
    "applicationConfiguration": [{
        "classification": "hive-site",
        "properties": {
            "hive.client.cores": "2",
            "hive.client.memory": "4GIB"
        }
    }]
}'
```

ジョブを記述するには、 を使用しますget-job-run。このコマンドは、ジョブ固有の設定と新しいジョブの設定容量を返します。

```
aws emr-serverless get-job-run \
```

```
--job-run-id job-id \
--application-id application-id
```

ジョブを一覧表示するには、 を使用しますlist-job-runs。このコマンドは、ジョブタイプ、状態、その他の高レベル属性を含むプロパティの省略セットを返します。すべてのジョブを表示しない場合は、表示するジョブの最大数を最大 50 個まで指定できます。次の例では、最後の 2 つのジョブ実行を表示するように指定します。

```
aws emr-serverless list-job-runs \
--max-results 2 \
--application-id application-id
```

ジョブをキャンセルするには、 を使用しますcancel-job-run。キャンセルするジョブjob-idの application-idと を指定します。

```
aws emr-serverless cancel-job-run \
--job-run-id job-id \
--application-id application-id
```

からジョブを実行する方法の詳細については AWS CLI、EMR「サーバーレスAPIリファレンス」を参照してください。

シャッフル最適化ディスクの使用

Amazon EMRリリース 7.1.0 以降では、Apache Spark または Hive ジョブを実行するときにシャッフル最適化ディスクを使用して、I/O 集約型ワークロードのパフォーマンスを向上させることができます。標準ディスクと比較して、シャッフル最適化ディスクは、シャッフル操作中のデータ移動を高速化し、レイテンシーを短縮するために、より高い IOPS (1 秒あたりの I/O 操作) を提供します。シャッフル最適化ディスクを使用すると、ワーカーごとに最大 2 TB のディスクサイズをアタッチできるため、ワークロード要件に適した容量を設定できます。

主な利点

シャッフル最適化ディスクには、次の利点があります。

- ハイIOPSパフォーマンス — シャッフル最適化ディスクは標準ディスクIOPSよりも高いディスクを提供するため、Spark および Hive ジョブやその他のシャッフル集約型のワークロード中に、より効率的で迅速なデータシャッフルが可能になります。

- より大きなディスクサイズ – シャッフル最適化ディスクは、ワーカーごとに 20GB から 2TB のディスクサイズをサポートしているため、ワークロードに基づいて適切な容量を選択できます。

使用開始

ワークフローでシャッフル最適化ディスクを使用するには、次のステップを参照してください。

Spark

1. 次のコマンドを使用して、EMRサーバーレスリリース 7.1.0 アプリケーションを作成します。

```
aws emr-serverless create-application \
  --type "SPARK" \
  --name my-application-name \
  --release-label emr-7.1.0 \
  --region <AWS_REGION>
```

2. パラメータを含めたりspark.emr-serverless.driver.disk.type、シャッフル最適化ディスクで実行したりspark.emr-serverless.executor.disk.typeするように Spark ジョブを設定します。ユースケースに応じて、一方または両方のパラメータを使用できます。

```
aws emr-serverless start-job-run \
    --application-id application-id \
    --execution-role-arn job-role-arn \
    --job-driver '{
        "sparkSubmit": {
            "entryPoint": "/usr/lib/spark/examples/jars/spark-examples.jar",
            "entryPointArguments": ["1"],
            "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi
            --conf spark.executor.cores=4
            --conf spark.executor.memory=20g
            --conf spark.driver.cores=4
            --conf spark.driver.memory=8g
            --conf spark.executor.instances=1
            --conf spark.emr-serverless.executor.disk.type=shuffle_optimized"
        }
    }'
```

詳細については、「Spark ジョブのプロパティ」を参照してください。

Hive

1. 次のコマンドを使用して、EMRサーバーレスリリース 7.1.0 アプリケーションを作成します。

```
aws emr-serverless create-application \
  --type "HIVE" \
  --name my-application-name \
  --release-label emr-7.1.0 \
  --region <AWS_REGION>
```

2. パラメータを含めたりhive.driver.disk.type、シャッフル最適化ディスクで実行したりhive.tez.disk.typeするように Hive ジョブを設定します。ユースケースに応じて、一方または両方のパラメータを使用できます。

```
aws emr-serverless start-job-run \
    --application-id application-id \
    --execution-role-arn job-role-arn \
    --job-driver '{
        "hive": {
            "query": "s3://<DOC-EXAMPLE-BUCKET>/emr-serverless-hive/query/hive-
query.ql",
            "parameters": "--hiveconf hive.log.explain.output=false"
        }
    }' \
    --configuration-overrides '{
        "applicationConfiguration": [{
            "classification": "hive-site",
            "properties": {
                "hive.exec.scratchdir": "s3://<DOC-EXAMPLE-BUCKET>/emr-
serverless-hive/hive/scratch",
                "hive.metastore.warehouse.dir": "s3://<DOC-EXAMPLE-BUCKET>/emr-
serverless-hive/hive/warehouse",
                "hive.driver.cores": "2",
                "hive.driver.memory": "4g",
                "hive.tez.container.size": "4096",
                "hive.tez.cpu.vcores": "1",
                "hive.driver.disk.type": "shuffle_optimized",
                "hive.tez.disk.type": "shuffle_optimized"
            }
        }]
    }'
```

詳細については、Hive ジョブのプロパティ を参照してください。

事前に初期化された容量でアプリケーションを設定する

Amazon EMRリリース 7.1.0 に基づいてアプリケーションを作成するには、次の例を参照してください。これらのアプリケーションには、次のプロパティがあります。

- 事前に初期化された 5 つの Spark ドライバー。それぞれに 2 v CPU、4 GB のメモリ、50 GB のシャッフル最適化ディスクが搭載されています。
- 事前に初期化されたエグゼキュター 50 個。それぞれに 4 vCPU、8 GB のメモリ、500 GB のシャッフル最適化ディスクが付属しています。

このアプリケーションが Spark ジョブを実行すると、まず事前に初期化されたワーカーを消費し、次にオンデマンドワーカーを最大容量の 400 vCPU と 1024 GB のメモリまでスケーリングします。オプションで、 DRIVERまたは の容量を省略できますEXECUTOR。

Spark

```
aws emr-serverless create-application \
  --type "SPARK" \
  --name <my-application-name> \
  --release-label emr-7.1.0 \
  --initial-capacity '{
    "DRIVER": {
        "workerCount": 5,
        "workerConfiguration": {
            "cpu": "2vCPU",
            "memory": "4GB",
            "disk": "50GB",
            "diskType": "SHUFFLE_OPTIMIZED"
        }
    },
    "EXECUTOR": {
        "workerCount": 50,
        "workerConfiguration": {
            "cpu": "4vCPU",
            "memory": "8GB",
            "disk": "500GB",
            "diskType": "SHUFFLE_OPTIMIZED"
        }
```

```
    }
  }' \
  --maximum-capacity '{
    "cpu": "400vCPU",
    "memory": "1024GB"
  }'
```

Hive

```
aws emr-serverless create-application \
  --type "HIVE" \
  --name <my-application-name> \
  --release-label emr-7.1.0 \
  --initial-capacity '{
    "DRIVER": {
        "workerCount": 5,
        "workerConfiguration": {
            "cpu": "2vCPU",
            "memory": "4GB",
            "disk": "50GB",
            "diskType": "SHUFFLE_OPTIMIZED"
        }
    },
    "EXECUTOR": {
        "workerCount": 50,
        "workerConfiguration": {
            "cpu": "4vCPU",
            "memory": "8GB",
            "disk": "500GB",
            "diskType": "SHUFFLE_OPTIMIZED"
        }
    }
  }' \
  --maximum-capacity '{
    "cpu": "400vCPU",
    "memory": "1024GB"
  }'
```

ストリーミングジョブ

EMR Serverless のストリーミングジョブは、ストリーミングデータをほぼリアルタイムで分析および処理できるジョブモードです。これらの長時間実行されるジョブは、ストリーミングデータをポー

リングし、データが到着すると継続的に結果を処理します。ストリーミングジョブは、ほぼリアルタイムの分析、不正検出、レコメンデーションエンジンなど、リアルタイムのデータ処理を必要とするタスクに最適です。EMR サーバーレスストリーミングジョブは、組み込みジョブの耐障害性、リアルタイムモニタリング、拡張ログ管理、ストリーミングコネクタとの統合などの最適化を提供します。

ストリーミングジョブのユースケースは次のとおりです。

- ほぼリアルタイムの分析 – Amazon EMR Serverless のストリーミングジョブでは、ストリーミングデータをほぼリアルタイムで処理できるため、ログデータ、センサーデータ、クリックストリームデータなどの継続的なデータストリームに対してリアルタイム分析を実行してインサイトを取得し、最新情報に基づいてタイムリーな意思決定を行うことができます。
- 不正検出 — データストリームを分析し、疑わしいパターンや異常が発生したときに特定すると、ストリーミングジョブを使用して、金融取引、クレジットカードオペレーション、オンラインアクティビティでほぼリアルタイムの不正検出を実行できます。
- レコメンデーションエンジン – ストリーミングジョブは、ユーザーアクティビティデータを処理してレコメンデーションモデルを更新できます。これにより、行動や好みに基づいてパーソナライズされたリアルタイムのレコメンデーションの可能性が広がります。
- ソーシャルメディア分析 – ストリーミングジョブは、ツイート、コメント、投稿などのソーシャルメディアデータを処理できるため、組織は傾向をモニタリングし、感情分析を行い、ブランドの評価をほぼリアルタイムで管理できます。
- モノのインターネット (IoT) 分析 – ストリーミングジョブは、IoT デバイス、センサー、コネクテッドマシンからの高速ストリームを処理および分析できるため、異常検出、予測メンテナンス、その他の IoT 分析のユースケースを実行できます。
- クリックストリーム分析 – ストリーミングジョブは、ウェブサイトまたはモバイルアプリケーションからのクリックストリームデータを処理および分析できます。このようなデータを使用する企業は、分析を実行して、ユーザー行動の詳細を確認し、ユーザーエクスペリエンスをパーソナライズし、マーケティングキャンペーンを最適化できます。
- ログのモニタリングと分析 — ストリーミングジョブは、サーバー、アプリケーション、ネットワークデバイスからのログデータを処理することもできます。これにより、異常の検出、トラブルシューティング、システムの状態とパフォーマンスが得られます。

主な利点

EMR Serverless のストリーミングジョブは、ジョブの耐障害性 を自動的に提供します。これは、次の要素の組み合わせです。

- 自動再試行 – EMRサーバーレスは、失敗したジョブを自動的に再試行します。ユーザーからの手動入力は必要ありません。
- アベイラビリティーゾーン (AZ) の回復性 – 元の AZ で問題が発生した場合、EMRサーバーレスはストリーミングジョブを正常な AZ に自動的に切り替えます。
- ログ管理：
  - ログのローテーション — ディスクストレージの管理を効率化するために、EMRサーバーレスは長時間のストリーミングジョブのログを定期的にローテーションします。これにより、すべてのディスク領域を消費する可能性のあるログの蓄積を防ぐことができます。
  - ログ圧縮 — は、マネージド永続化でログファイルを効率的に管理および最適化するのに役立ちます。圧縮により、マネージド型Spark 履歴サーバーを使用する場合のデバッグエクスペリエンスも向上します。

サポートされているデータソースとデータシンク

EMR サーバーレスは、多数の入力データソースと出力データシンクで動作します。

- サポートされている入力データソース – Amazon Kinesis Data Streams、Amazon Managed Streaming for Apache Kafka、およびセルフマネージド Apache Kafka クラスター。デフォルトでは、Amazon EMRリリース 7.1.0 以降には Amazon Kinesis Data Streams コネクタ が含まれているため、追加のパッケージを構築またはダウンロードする必要はありません。
- サポートされている出力データシンク – AWS Glue Data Catalog Amazon S3、Amazon Redshift、My SQL、PostgreSQL Oracle、Oracle、Microsoft、Apache IcebergSQL、Delta Lake、および Apache Hudi。

考慮事項と制約事項

ストリーミングジョブを使用する場合は、以下の考慮事項と制限事項に注意してください。

- ストリーミングジョブは、Amazon EMRリリース 7.1.0 以降の でサポートされています。
- EMR サーバーレスはストリーミングジョブを長時間実行することを期待しているため、実行タイムアウトを設定してジョブのランタイムを制限することはできません。
- ストリーミングジョブは、構造化されたストリーミングフレームワーク の上に構築された Spark エンジンとのみ互換性があります。
- EMR サーバーレスはストリーミングジョブを無期限に再試行するため、最大試行回数をカスタマイズすることはできません。1 時間あたりの時間枠で失敗した試行回数がしきい値セットを超えた

場合、スラッシュ防止が自動的に含まれ、ジョブの再試行が停止します。デフォルトのしきい値は、1 時間で 5 回の失敗です。このしきい値は、1 ~ 10 回の試行で設定できます。詳細については、「ジョブの耐障害性」を参照してください。

- ストリーミングジョブにはランタイムの状態と進行状況を保存するためのチェックポイントがあるため、EMRサーバーレスは最新のチェックポイントからストリーミングジョブを再開できます。詳細については、Apache Spark ドキュメントの「チェックポイントによる障害からの回復」を参照してください。

ストリーミングジョブの開始方法

ストリーミングジョブを開始する方法については、次の手順を参照してください。

1. 「Amazon EMR Serverless の開始方法」に従ってアプリケーションを作成します。アプリケーションは Amazon EMRリリース 7.1.0 以降を実行する必要があります。
2. アプリケーションの準備ができたら、 modeパラメータを に設定STREAMINGして、次のようなストリーミングジョブを送信します。 AWS CLI 例から始めることができます。

```
aws emr-serverless start-job-run \
--application-id <APPPLICATION_ID> \
--execution-role-arn <JOB_EXECUTION_ROLE> \
--mode 'STREAMING' \
--job-driver '{
    "sparkSubmit": {
        "entryPoint": "s3://<streaming script>",
        "entryPointArguments": ["s3://<DOC-EXAMPLE-BUCKET-OUTPUT>/output"],
        "sparkSubmitParameters": "--conf spark.executor.cores=4
            --conf spark.executor.memory=16g
            --conf spark.driver.cores=4
            --conf spark.driver.memory=16g
            --conf spark.executor.instances=3"
    }
}'
```

サポートされているストリーミングコネクタ

ストリーミングコネクタは、ストリーミングソースからのデータの読み取りを容易にし、ストリーミングシンクにデータを書き込むこともできます。

サポートされているストリーミングコネクタは次のとおりです。

Amazon Kinesis Data Streams コネクタ

Apache Spark 用の Amazon Kinesis Data Streams コネクタを使用すると、Amazon Kinesis Data Streams との間でデータを使用し、データを書き込むストリーミングアプリケーションとパイプラインを構築できます。コネクタは、シャードあたり最大 2MB/秒の専用読み取りスループットレートで、拡張ファンアウト消費をサポートします。デフォルトでは、Amazon EMR Serverless 7.1.0 以降にはコネクタが含まれているため、追加のパッケージを構築またはダウンロードする必要はありません。コネクタの詳細については、「」の spark-sql-kinesis-connector ページ GitHubを参照してください。

以下は、Kinesis Data Streams コネクタの依存関係を使用してジョブ実行を開始する方法の例です。

```
aws emr-serverless start-job-run \
--application-id <APPLICATION_ID> \
--execution-role-arn <JOB_EXECUTION_ROLE> \
--mode 'STREAMING' \
--job-driver '{
    "sparkSubmit": {
        "entryPoint": "s3://<Kinesis-streaming-script>",
        "entryPointArguments": ["s3://<DOC-EXAMPLE-BUCKET-OUTPUT>/output"],
        "sparkSubmitParameters": "--conf spark.executor.cores=4
                --conf spark.executor.memory=16g
                --conf spark.driver.cores=4
                --conf spark.driver.memory=16g
                --conf spark.executor.instances=3
                --jars /usr/share/aws/kinesis/spark-sql-kinesis/lib/spark-streaming-
sql-kinesis-connector.jar"
    }
}'
```

Kinesis Data Streams に接続するには、EMRサーバーレスアプリケーションVPCにアクセスを設定し、VPCエンドポイントを使用してプライベートアクセスを許可する必要があります。または、NATゲートウェイを使用してパブリックアクセスを取得する必要があります。詳細については、VPC「アクセスの設定」を参照してください。また、ジョブランタイムロールに、必要なデータストリームにアクセスするために必要な読み取りおよび書き込みアクセス許可があることを確認する必要があります。ジョブランタイムロールの設定方法の詳細については、「Amazon EMR Serverless のジョブランタイムロール」を参照してください。必要なすべてのアクセス許可の完全なリストについては、spark-sql-kinesis-connector 「」のページ GitHubを参照してください。

Apache Kafka コネクタ

Spark 構造化ストリーミング用の Apache Kafka コネクタは、Spark コミュニティのオープンソースコネクタであり、Maven リポジトリで利用できます。このコネクタは、Spark 構造化ストリーミングアプリケーションがセルフマネージド Apache Kafka と Amazon Managed Streaming for Apache Kafka との間でデータを読み書きすることを容易にします。コネクタの詳細については、Apache Spark ドキュメントの「構造化ストリーミング + Kafka 統合ガイド」を参照してください。

次の例は、Kafka コネクタをジョブ実行リクエストに含める方法を示しています。

```
aws emr-serverless start-job-run \
--application-id <APPLICATION_ID> \
--execution-role-arn <JOB_EXECUTION_ROLE> \
--mode 'STREAMING' \
--job-driver '{
    "sparkSubmit": {
        "entryPoint": "s3://<Kafka-streaming-script>",
        "entryPointArguments": ["s3://<DOC-EXAMPLE-BUCKET-OUTPUT>/output"],
        "sparkSubmitParameters": "--conf spark.executor.cores=4
                --conf spark.executor.memory=16g
                --conf spark.driver.cores=4
                --conf spark.driver.memory=16g
                --conf spark.executor.instances=3
                --packages org.apache.spark:spark-sql-
kafka-0-10_2.12:<KAFKA_CONNECTOR_VERSION>"
    }
}'
```

Apache Kafka コネクタのバージョンは、EMRサーバーレスリリースバージョンと対応する Spark バージョンによって異なります。正しい Kafka バージョンを見つけるには、「構造化ストリーミング + Kafka 統合ガイド」を参照してください。

IAM 認証で Amazon Managed Streaming for Apache Kafka を使用するには、別の依存関係を含めて、Kafka コネクタが MSKで Amazon に接続できるようにする必要がありますIAM。詳細については、「」のaws-msk-iam-auth 「 リポジトリ GitHub」を参照してください。また、ジョブランタイムロールに必要なIAMアクセス許可があることを確認する必要があります。次の例は、IAM認証でコネクタを使用する方法を示しています。

```
aws emr-serverless start-job-run \
--application-id <APPLICATION_ID> \
--execution-role-arn <JOB_EXECUTION_ROLE> \
--mode 'STREAMING' \
```

```
--job-driver '{
    "sparkSubmit": {
        "entryPoint": "s3://<Kafka-streaming-script>",
        "entryPointArguments": ["s3://<DOC-EXAMPLE-BUCKET-OUTPUT>/output"],
        "sparkSubmitParameters": "--conf spark.executor.cores=4
                --conf spark.executor.memory=16g
                --conf spark.driver.cores=4
                --conf spark.driver.memory=16g
                --conf spark.executor.instances=3
                --packages org.apache.spark:spark-sql-
kafka-0-10_2.12:<KAFKA_CONNECTOR_VERSION>,software.amazon.msk:aws-msk-iam-
auth:<MSK_IAM_LIB_VERSION>"
    }
}'
```

Amazon の Kafka コネクタとIAM認証ライブラリを使用するにはMSK、VPCサーバーEMRレスアプリケーションにアクセスを設定する必要があります。サブネットにはインターネットアクセスが必要で、Maven の依存関係にアクセスするにはNATゲートウェイを使用する必要があります。詳細については、VPC「アクセスの設定」を参照してください。Kafka クラスターにアクセスするには、サブネットにネットワーク接続が必要です。これは、Kafka クラスターがセルフマネージド型であるか、Amazon Managed Streaming for Apache Kafka を使用しているかに関係なく当てはまります。

ストリーミングジョブのログ管理

ログのローテーション

ストリーミングジョブは、Spark アプリケーションログとイベントログのログローテーションをサポートします。ログローテーションにより、長時間のストリーミングジョブで、使用可能なすべてのディスク領域を占める可能性のある大きなログファイルが生成されなくなります。ログローテーションは、ディスクストレージを節約し、ディスク容量不足によるジョブの失敗を防ぐのに役立ちます。詳細については、「ログのローテーション」を参照してください。

ログ圧縮

ストリーミングジョブは、マネージドログが利用可能な場合に Spark イベントログのログ圧縮もサポートします。マネージドログ記録の詳細については、「マネージドストレージを使用したログ記録」を参照してください。ストリーミングジョブは長時間実行でき、イベントデータ量は時間の経過とともに蓄積され、ログファイルのサイズが大幅に増加する可能性があります。Spark History Server は、これらのイベントを読み取り、Spark アプリケーション UI のメモリにロードします。このプロセスにより、特に Amazon S3 に保存されているイベントログが非常に大きい場合、レイテンシーとコストが高くなる可能性があります。

ログ圧縮はイベントログのサイズを小さくするため、Spark History Server はいつでも 1 GB を超えるイベントログをロードする必要はありません。詳細については、Apache Spark ドキュメントの「モニタリングと計測」を参照してください。

Spark ジョブ

`type` パラメータを に設定して、アプリケーションで Spark ジョブを実行できますSPARK。ジョブは、Amazon EMRリリースバージョンと互換性のある Spark バージョンと互換性がある必要があります。例えば、Amazon EMRリリース 6.6.0 でジョブを実行する場合、ジョブは Apache Spark 3.2.0 と互換性がある必要があります。各リリースのアプリケーションバージョンの詳細については、「」を参照してくださいAmazon EMR Serverless リリースバージョン。

Spark ジョブパラメータ

を使用して Spark ジョブ`StartJobRun`APIを実行する場合は、次のパラメータを指定できます。

必須パラメータ

- Spark ジョブランタイムロール
- Spark ジョブドライバーパラメータ
- Spark 設定オーバーライドパラメータ
- Spark 動的リソース割り当ての最適化

Spark ジョブランタイムロール

を使用して**`executionRoleArn`**、アプリケーションが Spark ジョブの実行に使用するIAMロール ARNの を指定します。このロールには、次のアクセス許可が含まれている必要があります。

- データが存在する S3 バケットまたはその他のデータソースからの読み取り
- PySpark スクリプトまたはJARファイルが存在する S3 バケットまたはプレフィックスから読み取る
- 最終出力を書き込む予定の S3 バケットに書き込む
- が`S3MonitoringConfiguration`指定する S3 バケットまたはプレフィックスにログを書き込む
- KMS キーを使用して S3 バケット内のデータを暗号化する場合のKMSキーへのアクセス
- Spark を使用している場合の AWS Glue Data Catalog へのアクセスSQL

Spark ジョブが他のデータソースとの間でデータの読み取りまたは書き込みを行う場合は、このIAMロールで適切なアクセス許可を指定します。これらのアクセス許可をIAMロールに提供しないと、ジョブが失敗する可能性があります。詳細については、「Amazon EMR Serverless のジョブランタイムロール」および「ログの保存」を参照してください。

Spark ジョブドライバーパラメータ

jobDriver を使用して、ジョブに入力を提供します。ジョブドライバーパラメータは、実行するジョブタイプに対して 1 つの値のみを受け入れます。Spark ジョブの場合、パラメータ値は ですsparkSubmit。このジョブタイプを使用して、Spark 送信を介して Scala、Java PySpark、SparkR 、およびその他のサポートされているジョブを実行できます。Spark ジョブには次のパラメータがあります。

- **sparkSubmitParameters** – これらは、ジョブに送信する追加の Spark パラメータです。このパラメータを使用して、 --conf または --class引数で定義されているような、ドライバーメモリやエグゼキューターの数などのデフォルトの Spark プロパティを上書きします。
- **entryPointArguments** – これは、メインファイルJARまたは Python ファイルに渡したい引数の配列です。これらのパラメータの読み取りは、エントリーポイントコードを使用して処理する必要があります。配列内の各引数をカンマで区切ります。
- **entryPoint** – これは、Amazon S3 で実行するメインファイルJARまたは Python ファイルへの参照です。Scala または Java を実行している場合はJAR、 --class引数SparkSubmitParameterを使用して でメインエントリクラスを指定します。

詳細については、spark-submit を使用したアプリケーションの起動を参照してください。

Spark 設定オーバーライドパラメータ

を使用して**configurationOverrides**、モニタリングレベルおよびアプリケーションレベルの設定プロパティを上書きします。このパラメータは、次の 2 つのフィールドを持つJSONオブジェクトを受け入れます。

- **monitoringConfiguration** - このフィールドを使用して、EMRサーバーレスジョブに Spark ジョブのログを保存する Amazon S3 URL (s3MonitoringConfiguration) を指定します。このバケットは、アプリケーションをホスト AWS アカウント する と同じ と、ジョブが実行され AWS リージョン ている と同じ で作成していることを確認してください。
- **applicationConfiguration** – アプリケーションのデフォルト設定を上書きするには、このフィールドに設定オブジェクトを指定できます。簡略構文を使用して設定を提供するか、JSON ファイル内の設定オブジェクトを参照できます。設定オブジェクトは、分類、プロパティ、オプ

ションの入れ子になっている設定で構成されます。プロパティは、そのファイル内で上書きする設定で構成されます。1 つのJSONオブジェクトで複数のアプリケーションに複数の分類を指定できます。

> **Note**
>
> 使用可能な設定分類は、特定の EMR Serverless リリースによって異なります。例えば、カスタム Log4j `spark-driver-log4j2`と の分類`spark-executor-log4j2`は、リリース 6.8.0 以降でのみ使用できます。

アプリケーションオーバーライドと Spark 送信パラメータで同じ設定を使用する場合、Spark 送信パラメータが優先されます。設定は、次のように優先順位が高くなります。

- EMR Serverless が を作成するときに提供する設定`SparkSession`。
- `--conf` 引数`sparkSubmitParameters`で の一部として指定する設定。
- アプリケーションの一部として提供する設定は、ジョブの開始時に上書きされます。
- アプリケーションの作成`runtimeConfiguration`時に の一部として提供する設定。
- Amazon がリリースEMRに使用する最適化設定。
- アプリケーションのデフォルトのオープンソース構成。

アプリケーションレベルでの設定の宣言と、ジョブ実行時の設定の上書きの詳細については、「」を参照してくださいEMR Serverless のデフォルトのアプリケーション設定。

Spark 動的リソース割り当ての最適化

Serverless EMR のリソース使用量を最適化`dynamicAllocationOptimization`するには、 を使用します。このプロパティを Spark 設定分類`true`で に設定すると、EMRServerless が エグゼキュターリソースの割り当てを最適化して、Spark が エグゼキュターをリクエストおよびキャンセルするレートとEMR、Serverless がワーカーを作成および解放するレートをより適切に調整できることを示します。これにより、EMRServerless はステージ間でワーカーをより最適に再利用できるため、同じパフォーマンスを維持しながら複数のステージでジョブを実行する場合のコストが削減されます。

このプロパティは、すべての Amazon EMRリリースバージョンで使用できます。

以下は、 を使用した設定分類の例ですdynamicAllocationOptimization。

```
[
  {
    "Classification": "spark",
    "Properties": {
      "dynamicAllocationOptimization": "true"
    }
  }
]
```

動的配分最適化を使用している場合は、次の点を考慮してください。

- この最適化は、動的リソース割り当てを有効にした Spark ジョブで使用できます。
- 最高のコスト効率を実現するには、ワークロードに基づいてジョブレベル設定`spark.dynamicAllocation.maxExecutors`または[アプリケーションレベルの最大容量]設定を使用して、ワーカーの上限スケーリングを設定することをお勧めします。
- 単純なジョブではコスト改善が見られない場合があります。例えば、ジョブが小さなデータセットで実行されている場合や、1 つのステージで実行が終了した場合、Spark はより多くの工グゼキュターや複数のスケーリングイベントを必要としない場合があります。
- 大きなステージ、小さなステージ、そして再び大きなステージのシーケンスを持つジョブでは、ジョブランタイムが後退する可能性があります。EMR Serverless がリソースをより効率的に使用すると、より大きなステージで利用可能なワーカーが少なくなり、ランタイムが長くなる可能性があります。

Spark ジョブのプロパティ

次の表に、オプションの Spark プロパティとそのデフォルト値を示します。これは、Spark ジョブを送信するときに上書きできます。

キー	説明	デフォルト値
`spark.archives`	Spark が各工グゼキューターの作業ディレクトリに抽出するアーカイブのカンマ区切りリスト。サポートされているファイルタイプには`.jar`、、`.tar.gz`、`.tgz`および が含まれます`.zip`。抽出する	NULL

キー	説明	デフォルト値
	ディレクトリ名を指定するには、抽出するファイル名の#後に を追加します。例えば、file.zip#directory と指定します。	
spark.authenticate	Spark の内部接続の認証を有効にするオプション。	TRUE
spark.driver.cores	ドライバーが使用するコアの数。	4
spark.driver.extraJavaOptions	Spark ドライバーの追加 Java オプション。	NULL
spark.driver.memory	ドライバーが使用するメモリの量。	14G
spark.dynamicAllocation.enabled	動的リソース割り当てを有効にするオプション。このオプションは、ワークロードに基づいて、アプリケーションに登録されたエグゼキューターの数をスケールアップまたはスケールダウンします。	TRUE
spark.dynamicAllocation.executorIdleTimeout	Spark がエグゼキューターを削除するまで、エグゼキューターがアイドル状態を維持できる時間の長さ。これは、動的割り当てをオンにした場合にのみ適用されます。	60 年代
spark.dynamicAllocation.initialExecutors	動的割り当てをオンにした場合に実行するエグゼキューターの初期数。	3

キー	説明	デフォルト値
spark.dynamicAllocation.maxExecutors	動的割り当てを有効にする場合のエグゼキュター数の上限。	6.10.0 以降では、 infinity 6.9.0 以前では、 100
spark.dynamicAllocation.minExecutors	動的割り当てを有効にする場合のエグゼキュター数の下限。	0
spark.emr-serverless.allocation.batch.size	エグゼキュター割り当ての各サイクルでリクエストするコンテナの数。各割り当てサイクルの間に 1 秒のギャップがあります。	20
spark.emr-serverless.driver.disk	Spark ドライバーディスク。	20G
spark.emr-serverless.driverEnv. *[KEY]*	Spark ドライバーに環境変数を追加するオプション。	NULL
spark.emr-serverless.executor.disk	Spark エグゼキュターディスク。	20G
spark.emr-serverless.memoryOverheadFactor	ドライバーとエグゼキュターのコンテナメモリに追加するメモリオーバーヘッドを設定します。	0.1
spark.emr-serverless.driver.disk.type	Spark ドライバーにアタッチされたディスクタイプ。	標準
spark.emr-serverless.executor.disk.type	Spark エグゼキュターにアタッチされたディスクタイプ。	標準
spark.executor.cores	各エグゼキュターが使用するコアの数。	4

キー	説明	デフォルト値
spark.executor.extraJavaOptions	Spark エグゼキュターの追加 Java オプション。	NULL
spark.executor.instances	割り当てる Spark エグゼキュターコンテナの数。	3
spark.executor.memory	各エグゼキュターが使用するメモリの量。	14G
spark.executorEnv.*[KEY]*	Spark エグゼキュターに環境変数を追加するオプション。	NULL
spark.files	各エグゼキューターの作業ディレクトリに移動するファイルのカンマ区切りリスト。これらのファイルのファイルパスには、 を使用してエグゼキューターでアクセスできますSparkFiles.get(*fileName*)。	NULL
spark.hadoop.hive.metastore.client.factory.class	Hive メタストア実装クラス。	NULL
spark.jars	ドライバーとエグゼキューターのランタイムクラスパスに追加する追加のジャー。	NULL
spark.network.crypto.enabled	AESベースのRPC暗号化を有効にするオプション。これには、Spark 2.2.0 で追加された認証プロトコルが含まれます。	FALSE

キー	説明	デフォルト値
spark.sql.warehouse.dir	マネージドデータベースとテーブルのデフォルトの場所。	$PWD/spark-warehouse
spark.submit.pyFiles	for PYTHONPATH Python アプリに配置する .zip、.egg、または .py ファイルのカンマ区切りリスト。	NULL

次の表に、デフォルトの Spark 送信パラメータを示します。

キー	説明	デフォルト値
archives	Spark が各エグゼキューターの作業ディレクトリに抽出するアーカイブのカンマ区切りリスト。	NULL
class	アプリケーションのメインクラス (Java および Scala アプリケーション用) 。	NULL
conf	任意の Spark 設定プロパティ。	NULL
driver-cores	ドライバーが使用するコアの数。	4
driver-memory	ドライバーが使用するメモリの量。	14G
executor-cores	各エグゼキュターが使用するコアの数。	4
executor-memory	エグゼキュターが使用するメモリの量。	14G

キー	説明	デフォルト値
files	各エグゼキューターの作業ディレクトリに配置するファイルのカンマ区切りリスト。これらのファイルのファイルパスには、 を使用してエグゼキューターでアクセスできますSparkFiles.get(*fileName*)。	NULL
jars	ドライバーとエグゼキュターのクラスパスに含めるジャーのカンマ区切りリスト。	NULL
num-executors	起動するエグゼキュターの数。	3
py-files	for PYTHONPATH Python アプリケーションに配置する .zip、.egg、または .pyファイルのカンマ区切りリスト。	NULL
verbose	追加のデバッグ出力を有効にするオプション。	NULL

Spark の例

次の例は、 StartJobRun API を使用して Python スクリプトを実行する方法を示しています。この例を使用するチュートリアルについては end-to-end、「」を参照してくださいAmazon EMR Serverless の開始方法。 PySpark ジョブを実行し、Python 依存関係を追加する方法のその他の例は、EMRServerless Samples GitHub リポジトリにあります。

```
aws emr-serverless start-job-run \
    --application-id application-id \
    --execution-role-arn job-role-arn \
    --job-driver '{
```

```
        "sparkSubmit": {
            "entryPoint": "s3://us-east-1.elasticmapreduce/emr-containers/samples/
wordcount/scripts/wordcount.py",
            "entryPointArguments": ["s3://amzn-s3-demo-destination-bucket/
wordcount_output"],
            "sparkSubmitParameters": "--conf spark.executor.cores=1 --conf
 spark.executor.memory=4g --conf spark.driver.cores=1 --conf spark.driver.memory=4g --
conf spark.executor.instances=1"
        }
    }'
```

次の例は、 StartJobRun API を使用して Spark を実行する方法を示していますJAR。

```
aws emr-serverless start-job-run \
    --application-id application-id \
    --execution-role-arn job-role-arn \
    --job-driver '{
        "sparkSubmit": {
            "entryPoint": "/usr/lib/spark/examples/jars/spark-examples.jar",
            "entryPointArguments": ["1"],
            "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi --conf
 spark.executor.cores=4 --conf spark.executor.memory=20g --conf spark.driver.cores=4 --
conf spark.driver.memory=8g --conf spark.executor.instances=1"
        }
    }'
```

Hive ジョブ

type パラメータを に設定して、アプリケーションで Hive ジョブを実行できますHIVE。ジョブは、Amazon EMRリリースバージョンと互換性のある Hive バージョンと互換性がある必要があります。例えば、Amazon EMRリリース 6.6.0 のアプリケーションでジョブを実行する場合、ジョブは Apache Hive 3.1.2 と互換性がある必要があります。各リリースのアプリケーションバージョンの詳細については、「」を参照してくださいAmazon EMR Serverless リリースバージョン。

Hive ジョブパラメータ

を使用して Hive ジョブStartJobRunAPIを実行する場合は、次のパラメータを指定する必要があります。

必須パラメータ

- Hive ジョブランタイムロール
- Hive ジョブドライバーパラメータ
- Hive 設定オーバーライドパラメータ

Hive ジョブランタイムロール

を使用して**executionRoleArn**、アプリケーションが Hive ジョブの実行に使用するIAMロール ARNの を指定します。このロールには、次のアクセス許可が含まれている必要があります。

- データが存在する S3 バケットまたはその他のデータソースからの読み取り
- Hive クエリファイルと init クエリファイルが存在する S3 バケットまたはプレフィックスから読み取る
- Hive Scratch ディレクトリと Hive Metastore ウェアハウスディレクトリが存在する S3 バケットへの読み取りと書き込み
- 最終出力を書き込む予定の S3 バケットに書き込む
- が`S3MonitoringConfiguration`指定する S3 バケットまたはプレフィックスにログを書き込む
- KMS キーを使用して S3 バケット内のデータを暗号化する場合のKMSキーへのアクセス
- AWS Glue データカタログへのアクセス

Hive ジョブが他のデータソースとの間でデータを読み取りまたは書き込みする場合は、このIAM ロールで適切なアクセス許可を指定します。これらのアクセス許可をIAMロールに提供しないと、ジョブが失敗する可能性があります。詳細については、「Amazon EMR Serverless のジョブランタイムロール」を参照してください。

Hive ジョブドライバーパラメータ

jobDriver を使用して、ジョブに入力を提供します。ジョブドライバーパラメータは、実行するジョブタイプに対して 1 つの値のみを受け入れます。をジョブタイプ`hive`として指定すると、EMRServerless は Hive クエリを `jobDriver`パラメータに渡します。Hive ジョブには次のパラメータがあります。

- **query** – これは、Amazon S3 で実行する Hive クエリファイルへの参照です。
- **parameters** – 上書きする追加の Hive 設定プロパティです。プロパティを上書きするには、 としてこのパラメータに渡します`--hiveconf` *`property=value`*。変数を上書きするには、 としてこのパラメータに渡します`--hivevar` *`key=value`*。

- **initQueryFile** – これは init Hive クエリファイルです。Hive はクエリの前にこのファイルを実行し、それを使用してテーブルを初期化できます。

Hive 設定オーバーライドパラメータ

を使用して**configurationOverrides**、モニタリングレベルおよびアプリケーションレベルの設定プロパティを上書きします。このパラメータは、次の 2 つのフィールドを持つJSONオブジェクトを受け入れます。

- **monitoringConfiguration** – このフィールドを使用して、EMRサーバーレスジョブに Hive ジョブの口グを保存する Amazon S3 URL (s3MonitoringConfiguration) を指定します。このバケットは、アプリケーションをホスト AWS アカウント する と同じ と、ジョブが実行され AWS リージョン ている と同じ で作成してください。
- **applicationConfiguration** – このフィールドに設定オブジェクトを指定して、アプリケーションのデフォルト設定を上書きできます。短縮構文を使用して設定を指定するか、JSONファイル内の設定オブジェクトを参照できます。設定オブジェクトは、分類、プロパティ、オプションの入れ子になっている設定で構成されます。プロパティは、そのファイル内で上書きする設定で構成されます。1 つのJSONオブジェクトで複数のアプリケーションに複数の分類を指定できます。

> **Note**
>
> 使用可能な設定分類は、特定の EMR Serverless リリースによって異なります。例えば、カスタム Log4j `spark-driver-log4j2`と の分類`spark-executor-log4j2`は、リリース 6.8.0 以降でのみ使用できます。

アプリケーションオーバーライドと Hive パラメータで同じ設定を渡すと、Hive パラメータが優先されます。次のリストでは、設定を最優先度から最優先度にランク付けします。

- で Hive パラメータの一部として提供する設定`--hiveconf` *`property=value`*。
- アプリケーションの一部として提供する設定は、ジョブの開始時に上書きされます。
- アプリケーションの作成`runtimeConfiguration`時に の一部として提供する設定。
- Amazon がリリースEMRに割り当てる最適化設定。
- アプリケーションのデフォルトのオープンソース設定。

アプリケーションレベルでの設定の宣言と、ジョブ実行中の設定の上書きの詳細については、「」を参照してくださいEMR Serverless のデフォルトのアプリケーション設定。

Hive ジョブのプロパティ

次の表に、Hive ジョブを送信するときに設定する必要がある必須プロパティを示します。

設定	説明
`hive.exec.scratchdir`	Hive ジョブの実行中に EMR Serverless が一時ファイルを作成する Amazon S3 の場所。
`hive.metastore.warehouse.dir`	Hive のマネージドテーブルのデータベースの Amazon S3 の場所。

次の表に、オプションの Hive プロパティと、Hive ジョブを送信するときに上書きできるデフォルト値を示します。

設定	説明	デフォルト値
`fs.s3.customAWSCredentialsProvider`	使用する AWS 認証情報プロバイダー。	com.amazonaws.auth.DefaultAWSCredentialsProviderChain
`fs.s3a.aws.credentials.provider`	S3A ファイルシステムで使用する AWS 認証情報プロバイダー。	com.amazonaws.auth.DefaultAWSCredentialsProviderChain
`hive.auto.convert.join`	入力ファイルサイズに基づいて、共通結合を Mapjoins に自動変換するオプション。	`TRUE`
`hive.auto.convert.join.noconditionaltask`	Hive が入力ファイルサイズに基づいて共通結合を mapjoin に変換するときに最適化を有効にするオプション。	`TRUE`

設定	説明	デフォルト値
hive.auto.convert.join.noconditionaltask.size	結合は、このサイズ未満の Mapjoin に直接変換されます。	最適な値は Tez タスクメモリに基づいて計算されます
hive.cbo.enable	Calcite フレームワークでコストベースの最適化を有効にするオプション。	TRUE
hive.cli.tez.session.async	Hive クエリのコンパイル中にバックグラウンド Tez セッションを開始するオプション。に設定するとfalse、Hive クエリのコンパイル後に Tez AM が起動します。	TRUE
hive.compute.query.using.stats	Hive をアクティブ化して、メタストアに保存された統計を使用して特定のクエリに回答するオプション。基本的な統計については、 hive.stats.autogather を に設定しますTRUE。クエリのより高度なコレクションについては、 を実行しますanalyze table queries。	TRUE
hive.default.fileformat	CREATE TABLE ステートメントのデフォルトのファイル形式。CREATE TABLE コマンドSTORED AS [FORMAT]でを指定した場合、これを明示的に上書きできます。	TEXTFILE
hive.driver.cores	Hive ドライバープロセスに使用するコアの数。	2

設定	説明	デフォルト値
hive.driver.disk	Hive ドライバーのディスクサイズ。	20G
hive.driver.disk.type	Hive ドライバーのディスクタイプ。	標準
hive.tez.disk.type	Tez ワーカーのディスクサイズ。	標準
hive.driver.memory	Hive ドライバープロセスごとに使用するメモリの量。Hive アプリケーションマスターCLI と Tez アプリケーションマスターは、このメモリをヘッドルームの 20% と等しく共有します。	6G
hive.emr-serverless.launch.env.[*KEY*]	Hive ドライバー、Tez AM、Tez タスクなど、すべての Hive 固有のプロセスで*KEY*環境変数を設定するオプション。	
hive.exec.dynamic.partition	DML/ で動的パーティションを有効にするオプションDDL。	TRUE

設定	説明	デフォルト値
hive.exec.dynamic.partition.mode	厳格モードを使用するか、非厳格モードを使用するかを指定するオプション。厳格モードでは、すべてのパーティションを誤って上書きする場合に備えて、少なくとも 1 つの静的パーティションを指定する必要があります。非厳密なモードでは、すべてのパーティションを動的にすることができます。	strict
hive.exec.max.dynamic.partitions	Hive が作成する動的パーティションの最大数。	1,000
hive.exec.max.dynamic.partitions.pernode	Hive が各マッパーノードとレデューサーノードに作成する動的パーティションの最大数。	100
hive.exec.orc.split.strategy	、BI、ETLまたは のいずれかの値が必要ですHYBRID。これはユーザーレベルの設定ではありません。 は、クエリの実行よりも分割生成に費やす時間を減らすようにBI指定します。 は分割生成により多くの時間を費やすようにETL指定します。 は、ヒューリスティックに基づいて上記の戦略の選択HYBRIDを指定します。	HYBRID

設定	説明	デフォルト値
hive.exec.reducers.bytes.per.reducer	リデューサーあたりのサイズ。デフォルトは 256 MB です。入力サイズが 1G の場合、ジョブは 4 つのリデューサーを使用します。	256000000
hive.exec.reducers.max	レデューサーの最大数。	256
hive.exec.stagingdir	Hive がテーブルの場所内と、hive.exec.scratchdir プロパティで指定されたスクラッチディレクトリの場所内に作成する一時ファイルを保存するディレクトリの名前。	.hive-staging
hive.fetch.task.conversion	、NONE、MINIMALまたはのいずれかの値が必要ですMORE。Hive は、選択クエリを 1 つのFETCHタスクに変換できます。これにより、レイテンシーが最小限に抑えられます。	MORE
hive.groupby.position.alias	Hive がGROUP BYステートメントで列位置エイリアスを使用するようにするオプション。	FALSE
hive.input.format	デフォルトの入力形式。で問題が発生した場合HiveInputFormat は、 を に設定しますCombineHiveInputFormat 。	org.apache.hadoop.hive.ql.io.CombineHiveInputFormat

設定	説明	デフォルト値
hive.log.explain.output	Hive ログ内のクエリの拡張出力の説明を有効にするオプション。	FALSE
hive.log.level	Hive ログ記録レベル。	INFO
hive.mapred.reduce.tasks.speculative.execution	レデューサーの投機的起動を有効にするオプション。Amazon EMR 6.10.x 以前でのみサポートされています。	TRUE
hive.max-task-containers	同時コンテナの最大数。設定されたマッパーメモリにこの値を乗算して、計算とタスクのプリエンプションで使用する使用可能なメモリを決定します。	1,000
hive.merge.mapfiles	マップのみのジョブの最後に小さなファイルがマージされるオプション。	TRUE
hive.merge.size.per.task	ジョブの最後にマージされたファイルのサイズ。	256000000
hive.merge.tezfiles	Tez の最後に小さなファイルのマージを有効にするオプションDAG。	FALSE
hive.metastore.client.factory.class	IMetaStoreClient インターフェイスを実装するオブジェクトを生成するファクトリクラスの名前。	com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory

設定	説明	デフォルト値
hive.metastore.glue.catalogid	AWS Glue Data Catalog がメタストアとして機能し、ジョブとは異なる AWS アカウント で実行される AWS アカウント 場合、ジョブが実行されている の ID。	NULL
hive.metastore.uris	メタストアクライアントURIがリモートメタストアへの接続に使用するドリフト。	NULL
hive.optimize.ppd	述語プッシュダウンを有効にするオプション。	TRUE
hive.optimize.ppd.storage	ストレージハンドラーへの述語プッシュダウンを有効にするオプション。	TRUE
hive.orderby.position.alias	Hive がORDER BYステートメントで列位置エイリアスを使用するようにするオプション。	TRUE
hive.prewarm.enabled	Tez のコンテナ事前ウォームを有効にするオプション。	FALSE
hive.prewarm.numcontainers	Tez の事前ウォームするコンテナの数。	10
hive.stats.autogather	INSERT OVERWRITE コマンド中に Hive が基本統計を自動的に収集できるようにするオプション。	TRUE

設定	説明	デフォルト値
hive.stats.fetch.column.stats	メタストアからの列統計の取得をオフにするオプション。列の数が多い場合、列統計のフェッチは高価になる可能性があります。	FALSE
hive.stats.gather.num.threads	partialscan および noscan analyze コマンドがパーティションテーブルに使用するスレッドの数。これは、 StatsProvidingRecordReader （ など) を実装するファイル形式にのみ適用されますORC。	10
hive.strict.checks.cartesian.product	厳密なデカルト結合チェックを有効にするオプション。これらのチェックでは、デカルト製品 (クロス結合) は許可されません。	FALSE
hive.strict.checks.type.safety	厳密なタイプの安全チェックを有効にし、 string と bigintの両方との比較をオフにするオプションdouble。	TRUE
hive.support.quoted.identifiers	NONE または の値が必要ですCOLUMN。 NONE は、識別子で有効な英数字とアンダースコア文字のみを意味します。COLUMN は、列名に任意の文字を含めることができることを示します。	COLUMN

設定	説明	デフォルト値
hive.tez.auto.reducer.parallelism	Tez 自動削減並列処理機能を有効にするオプション。Hive は引き続きデータサイズを推定し、並列処理の推定を設定します。Tez はソース頂点の出力サイズをサンプリングし、必要に応じて実行時に見積りを調整します。	TRUE
hive.tez.container.size	Tez タスクプロセスごとに使用するメモリの量。	6144
hive.tez.cpu.vcores	Tez タスクごとに使用するコアの数。	2
hive.tez.disk.size	各タスクコンテナのディスクサイズ。	20G
hive.tez.input.format	Tez AM での分割生成の入力形式。	org.apache.hadoop.hive.ql.io.HiveInputFormat
hive.tez.min.partition.factor	自動リデューサー並列処理をオンにするときに Tez が指定するレデューサーの下限。	0.25
hive.vectorized.execution.enabled	ベクトル化されたクエリ実行モードを有効にするオプション。	TRUE
hive.vectorized.execution.reduce.enabled	クエリ実行のレデュース側のベクトル化モードを有効にするオプション。	TRUE
javax.jdo.option.ConnectionDriverName	JDBC メタストアのドライバークラス名。	org.apache.derby.jdbc.EmbeddedDriver

設定	説明	デフォルト値
javax.jdo.option.ConnectionPassword	メタストアデータベースに関連付けられたパスワード。	NULL
javax.jdo.option.ConnectionURL	JDBC メタストアJDBCの接続文字列。	jdbc:derby:;databaseName=metastore_db;create=true
javax.jdo.option.ConnectionUserName	メタストアデータベースに関連付けられているユーザー名。	NULL
mapreduce.input.fileinputformat.split.maxsize	入力形式が の場合の分割計算中の分割の最大サイズ org.apache.hadoop.hive.ql.io.CombineHiveInputFormat 。値 0 (ゼロ) は、制限がないことを示します。	0
tez.am.dag.cleanup.on.completion	DAG 完了時にシャッフルデータのクリーンアップを有効にするオプション。	TRUE
tez.am.emr-serverless.launch.env.[*KEY*]	Tez AM プロセスで*KEY*環境変数を設定するオプション。Tez AM の場合、この値はhive.emr-serverless.launch.env.[*KEY*]値を上書きします。	
tez.am.log.level	EMR Serverless が Tez アプリケーションマスターに渡すルートログ記録レベル。	INFO

設定	説明	デフォルト値
tez.am.sleep.time.before.exit.millis	EMR サーバーレスは、AM シャットダウンリクエストの後に、この期間の後にATSイベントをプッシュする必要があります。	0
tez.am.speculation.enabled	遅いタスクの投機的な起動を引き起こすオプション。これにより、一部のタスクの実行が遅くなっている場合、マシンが不良または遅い場合にジョブのレイテンシーを減らすことができます。Amazon EMR 6.10.x 以前でのみサポートされています。	FALSE
tez.am.task.max.failed.attempts	タスクが失敗する前に、特定のタスクで失敗する可能性がある最大試行回数。この数は、手動で終了した試行数をカウントしません。	3
tez.am.vertex.cleanup.height	すべての依存頂点が完了すると、Tez AM が頂点シャッフルデータを削除する距離。この機能は、値が 0 の場合にオフになります。Amazon EMR バージョン 6.8.0 以降では、この機能がサポートされています。	0
tez.client.asynchronous-stop	Hive ドライバーを終了する前に EMR Serverless がATSイベントをプッシュするオプション。	FALSE

設定	説明	デフォルト値
tez.grouping.max-size	グループ化された分割の最大サイズ制限 (バイト単位) 。この制限により、過度に大きな分割が防止されます。	1073741824
tez.grouping.min-size	グループ化された分割の下限サイズ (バイト単位) 。この制限により、小さすぎる分割が防止されます。	16777216
tez.runtime.io.sort.mb	Tez が出力をソートするときのソフトバッファのサイズはソートされます。	最適な値は Tez タスクメモリに基づいて計算されます
tez.runtime.unordered.output.buffer.size-mb	Tez がディスクに直接書き込まない場合に使用するバッファのサイズ。	最適な値は Tez タスクメモリに基づいて計算されます
tez.shuffle-vertex-manager.max-src-fraction	EMR Serverless が現在の頂点のすべてのタスクをスケジュールする前に完了する必要があるソースタスクの割合 (ScatterGather 接続の場合) 。現在の頂点でスケジューリングできるタスクの数は、 min-fraction と の間で線形にスケールされますmax-fraction 。これはデフォルト値または tez.shuffle-vertex-manager.min-src-fraction のいずれか大きい方です。	0.75

設定	説明	デフォルト値
tez.shuffle-vertex-manager.min-src-fraction	EMR Serverless が現在の頂点のタスクをスケジュールする前に完了する必要があるソースタスクの割合 (ScatterGather 接続の場合) 。	0.25
tez.task.emr-serverless.launch.env.[*KEY*]	Tez タスクプロセスで*KEY*環境変数を設定するオプション。Tez タスクの場合、この値はhive.emr-serverless.launch.env.[*KEY*]値を上書きします。	
tez.task.log.level	EMR Serverless が Tez タスクに渡すルートログ記録レベル。	INFO
tez.yarn.ats.event.flush.timeout.millis	シャットダウンする前にイベントがフラッシュされるまで AM が待機する最大時間。	300000

Hive ジョブの例

次のコード例は、 で Hive StartJobRun クエリを実行する方法を示していますAPI。

```
aws emr-serverless start-job-run \
    --application-id application-id \
    --execution-role-arn job-role-arn \
    --job-driver '{
        "hive": {
            "query": "s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-
query.ql",
            "parameters": "--hiveconf hive.log.explain.output=false"
        }
    }' \
    --configuration-overrides '{
        "applicationConfiguration": [{
```

```
            "classification": "hive-site",
            "properties": {
                "hive.exec.scratchdir": "s3://amzn-s3-demo-bucket/emr-serverless-hive/
hive/scratch",
                "hive.metastore.warehouse.dir": "s3://amzn-s3-demo-bucket/emr-
serverless-hive/hive/warehouse",
                "hive.driver.cores": "2",
                "hive.driver.memory": "4g",
                "hive.tez.container.size": "4096",
                "hive.tez.cpu.vcores": "1"
            }
        }]
    }'
```

EMR サーバーレスサンプル GitHub リポジトリで Hive ジョブを実行する方法のその他の例があります。

EMR サーバーレスジョブの耐障害性

EMR サーバーレスリリース 7.1.0 以降にはジョブの回復性のサポートが含まれているため、失敗したジョブは手動で入力しなくても自動的に再試行されます。ジョブの耐障害性に関するもう 1 つの利点は、AZ で問題が発生した場合に、EMRサーバーレスがジョブ実行を別のアベイラビリティーゾーン (AZ) に移動することです。

ジョブのジョブレジリエンシーを有効にするには、ジョブの再試行ポリシーを設定します。再試行ポリシーにより、EMRサーバーレスは、ジョブが任意の時点で失敗した場合にジョブを自動的に再起動します。再試行ポリシーはバッチジョブとストリーミングジョブの両方でサポートされるため、ユースケースに応じてジョブの耐障害性をカスタマイズできます。次の表は、バッチジョブとストリーミングジョブにおけるジョブの耐障害性と動作の違いを比較したものです。

	バッチジョブ	ストリーミングジョブ
デフォルトの動作	ジョブを再実行しません。	アプリケーションがジョブの実行中にチェックポイントを作成するため、常にジョブの実行を再試行します。
再試行ポイント	バッチジョブにはチェックポイントがないため、EMRサー	ストリーミングジョブはチェックポイントをサポートするため、Amazon S3 の

	バッチジョブ	ストリーミングジョブ
	バーレスは常にジョブを最初から再実行します。	チェックポイントの場所にランタイム状態と進行状況を保存するようにストリーミングクエリを設定できます。EMRサーバーレスはチェックポイントからジョブ実行を再開します。詳細については、Apache Spark ドキュメントの「チェックポイントによる障害からの回復」を参照してください。
再試行の最大回数	最大 10 回の再試行を許可します。	ストリーミングジョブにはスラッシュ防止コントロールが組み込まれているため、1 時間後にジョブが失敗し続けると、アプリケーションは再試行を停止します。1 時間以内のデフォルトの再試行回数は 5 回です。この再試行回数は 1 ～ 10 回に設定できます。最大試行回数をカスタマイズすることはできません。値が 1 の場合、再試行がないことを示します。

EMR Serverless がジョブを再実行しようとすると、ジョブに試行番号のインデックスも付けられるため、試行全体のジョブのライフサイクルを追跡できます。

EMR Serverless APIオペレーションまたは AWS CLI は、ジョブの耐障害性を変更したり、ジョブの耐障害性に関連する情報を表示したりします。詳細については、EMR「サーバーレスAPIガイド」を参照してください。

デフォルトでは、EMRサーバーレスはバッチジョブを再実行しません。バッチジョブの再試行を有効にするには、バッチジョブの実行を開始するときに `maxAttempts`パラメータを設定しま

す。maxAttempts パラメータはバッチジョブにのみ適用されます。デフォルトは 1 です。つまり、ジョブを再実行しません。使用できる値は 1 ~ 10 です。

次の例は、ジョブ実行を開始するときに最大 10 回の試行を指定する方法を示しています。

```
aws emr-serverless start-job-run
 --application-id <APPLICATION_ID> \
 --execution-role-arn <JOB_EXECUTION_ROLE> \
 --mode 'BATCH' \
 --retry-policy '{
    "maxAttempts": 10
 }' \
 --job-driver '{
    "sparkSubmit": {
         "entryPoint": "/usr/lib/spark/examples/jars/spark-examples-does-not-
exist.jar",
         "entryPointArguments": ["1"],
         "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi"
     }
}'
```

EMR サーバーレスは、失敗したストリーミングジョブを無期限に再試行します。回復不可能な障害が繰り返し発生するためにスラッシュを防ぐには、 を使用して、ストリーミングジョブの再試行のスラッシュ防止コントロールmaxFailedAttemptsPerHourを設定します。このパラメータを使用すると、EMRサーバーレスが再試行を停止するまでの 1 時間で許可される最大試行失敗回数を指定できます。デフォルトは 5 です。使用できる値は 1 ~ 10 です。

```
aws emr-serverless start-job-run
 --application-id <APPPLICATION_ID> \
 --execution-role-arn <JOB_EXECUTION_ROLE> \
 --mode 'STREAMING' \
 --retry-policy '{
    "maxFailedAttemptsPerHour": 7
 }' \
 --job-driver '{
    "sparkSubmit": {
         "entryPoint": "/usr/lib/spark/examples/jars/spark-examples-does-not-
exist.jar",
         "entryPointArguments": ["1"],
         "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi"
     }
}'
```

他のジョブ実行APIオペレーションを使用して、ジョブに関する情報を取得することもできます。例えば、 attemptパラメータを GetJobRunオペレーションで使用して、特定のジョブの試行に関する詳細を取得できます。attempt パラメータを含めない場合、オペレーションは最新の試行に関する情報を返します。

```
aws emr-serverless get-job-run \
    --job-run-id job-run-id \
    --application-id application-id \
    --attempt 1
```

ListJobRunAttempts オペレーションは、ジョブ実行に関連するすべての試行に関する情報を返します。

```
aws emr-serverless list-job-run-attempts \
  --application-id application-id \
  --job-run-id job-run-id
```

GetDashboardForJobRun オペレーションは、ジョブ実行UIsのアプリケーションにアクセスするためにURL使用できる を作成して返します。attempt パラメータを使用すると、特定の試行URLに対して を取得できます。attempt パラメータを含めない場合、オペレーションは最新の試行に関する情報を返します。

```
aws emr-serverless get-dashboard-for-job-run \
    --application-id application-id \
    --job-run-id job-run-id \
    --attempt 1
```

再試行ポリシーを使用してジョブをモニタリングする

ジョブの耐障害性のサポートでは、新しいイベント EMR Serverless ジョブ実行再試行 も追加されます。EMR Serverless は、ジョブの再試行ごとにこのイベントを発行します。この通知を使用して、ジョブの再試行を追跡できます。イベントの詳細については、「Amazon EventBridge イベント」を参照してください。

再試行ポリシーによるログ記録

EMR サーバーレスがジョブを再試行するたびに、試行によって独自のログセットが生成されます。EMR Serverless がこれらのログを上書き CloudWatch せずに Amazon S3 と Amazon に正常に

配信できるように、EMRServerless は S3 ログパスと CloudWatch ログストリーム名の形式にプレフィックスを追加して、ジョブの試行数を含めます。

形式の例を次に示します。

```
'/applications/<applicationId>/jobs/<jobId>/attempts/<attemptNumber>/'.
```

この形式により、EMRServerless はジョブの試行ごとにすべてのログを Amazon S3 および 内の独自の指定された場所に発行します CloudWatch。詳細については、「ログの保存」を参照してください。

> **Note**
>
> EMR サーバーレスは、すべてのストリーミングジョブと再試行が有効になっているバッチジョブでのみ、このプレフィックス形式を使用します。

メタストア設定

Hive メタストアは、スキーマ、パーティション名、データ型など、テーブルに関する構造情報を保存する一元的な場所です。EMR Serverless を使用すると、このテーブルメタデータをジョブにアクセスできるメタストアに保持できます。

Hive メタストアには 2 つのオプションがあります。

- AWS Glue データカタログ
- 外部 Apache Hive メタストア

AWS Glue Data Catalog をメタストアとして使用する

AWS Glue Data Catalog をメタストアとして使用するように Spark ジョブと Hive ジョブを設定できます。この設定は、永続的なメタストア、または異なるアプリケーション、サービス、または によって共有されるメタストアが必要な場合にお勧めします AWS アカウント。データカタログの詳細については、AWS 「Glue データカタログ の入力」を参照してください。 AWS Glue の料金については、AWS 「Glue の料金」を参照してください。

AWS Glue Data Catalog を使用するようにEMRサーバーレスジョブを、 AWS アカウント アプリケーションと同じ または別の で設定できます AWS アカウント。

AWS Glue データカタログを設定する

データカタログを設定するには、使用するEMRサーバーレスアプリケーションのタイプを選択します。

Spark

EMR Studio を使用して EMR Serverless Spark アプリケーションでジョブを実行する場合、AWS Glue Data Catalog はデフォルトのメタストアです。

SDKs または を使用する場合 AWS CLI、ジョブ実行のsparkSubmitパラメータcom.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactoryで spark.hadoop.hive.metastore.client.factory.class 設定を に設定することができます。次の例は、 を使用して Data Catalog を設定する方法を示しています AWS CLI。

```
aws emr-serverless start-job-run \
    --application-id application-id \
    --execution-role-arn job-role-arn \
    --job-driver '{
        "sparkSubmit": {
            "entryPoint": "s3://amzn-s3-demo-bucket/code/pyspark/
extreme_weather.py",
            "sparkSubmitParameters": "--conf
 spark.hadoop.hive.metastore.client.factory.class=com.amazonaws.glue.catalog.metastore.AWSGl
 --conf spark.driver.cores=1 --conf spark.driver.memory=3g --conf
 spark.executor.cores=4 --conf spark.executor.memory=3g"
        }
    }'
```

または、Spark コードSparkSessionで新しい を作成するときにこの設定を設定することもできます。

```
from pyspark.sql import SparkSession

spark = (
    SparkSession.builder.appName("SparkSQL")
    .config(
        "spark.hadoop.hive.metastore.client.factory.class",
        "com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory",
    )
    .enableHiveSupport()
    .getOrCreate()
```

```
)

# we can query tables with SparkSQL
spark.sql("SHOW TABLES").show()

# we can also them with native Spark
print(spark.catalog.listTables())
```

Hive

EMR Serverless Hive アプリケーションの場合、Data Catalog はデフォルトのメタストアです。つまり、EMRServerless Hive アプリケーションでジョブを実行すると、Hive はアプリケーション AWS アカウント と同じ Data Catalog にメタストア情報を記録します。Data Catalog をメタストアとして使用するには、仮想プライベートクラウド (VPC) は必要ありません。

Hive メタストアテーブルにアクセスするには、 AWS Glue のアクセス許可の設定 で説明されている必要な AWS Glue IAM ポリシーを追加します。

EMR Serverless および AWS Glue Data Catalog のクロスアカウントアクセスを設定する

EMR Serverless のクロスアカウントアクセスを設定するには、まず次の にサインインする必要があります AWS アカウント。

- AccountA – EMR Serverless アプリケーションを作成 AWS アカウント している 。
- AccountB – EMR Serverless ジョブを実行する AWS Glue データカタログ AWS アカウント を含む 。

1. の管理者またはその他の承認された ID が、 のデータカタログにリソースポリシーをAccountBアタッチしていることを確認しますAccountB。このポリシーは、AccountBカタログ内のリソースに対してオペレーションを実行するためのAccountA特定のクロスアカウントアクセス許可を付与します。

```
{
  "Version" : "2012-10-17",
  "Statement" : [ {
    "Effect" : "Allow",
    "Principal": {
        "AWS": [
```

```
            "arn:aws:iam::accountA:role/job-runtime-role-A"
        ]},
    "Action" : [
        "glue:GetDatabase",
        "glue:CreateDatabase",
        "glue:GetDataBases",
        "glue:CreateTable",
        "glue:GetTable",
        "glue:UpdateTable",
        "glue:DeleteTable",
        "glue:GetTables",
        "glue:GetPartition",
        "glue:GetPartitions",
        "glue:CreatePartition",
        "glue:BatchCreatePartition",
        "glue:GetUserDefinedFunctions"
    ],
    "Resource": ["arn:aws:glue:region:AccountB:catalog"]
  } ]
}
```

2. ロールが の Data Catalog リソースにアクセスできるAccountAように、 のEMRサーバーレスジョブランタイムロールにIAMポリシーを追加しますAccountB。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:GetDatabase",
        "glue:CreateDatabase",
        "glue:GetDataBases",
        "glue:CreateTable",
        "glue:GetTable",
        "glue:UpdateTable",
        "glue:DeleteTable",
        "glue:GetTables",
        "glue:GetPartition",
        "glue:GetPartitions",
        "glue:CreatePartition",
        "glue:BatchCreatePartition",
        "glue:GetUserDefinedFunctions"
```

```
      ],
      "Resource": ["arn:aws:glue:region:AccountB:catalog"]
    }
  ]
}
```

3. ジョブの実行を開始します。このステップは、 AccountAの EMR Serverless アプリケーションタイプによって若干異なります。

   Spark

   次の例に示すように、hive-site分類に spark.hadoop.hive.metastore.glue.catalogidプロパティを設定します。置換 *AccountB -catalog-id* でデータカタログの ID を使用しますAccountB。

   ```
   aws emr-serverless start-job-run \
   --application-id "application-id" \
   --execution-role-arn "job-role-arn" \
   --job-driver '{
       "sparkSubmit": {
           "query": "s3://amzn-s3-demo-bucket/hive/scripts/create_table.sql",
           "parameters": "--hiveconf hive.exec.scratchdir=s3://amzn-s3-demo-bucket/
   hive/scratch --hiveconf hive.metastore.warehouse.dir=s3://amzn-s3-demo-bucket/
   hive/warehouse"
       }
     }' \
   --configuration-overrides '{
       "applicationConfiguration": [{
           "classification": "hive-site",
           "properties": {
               "spark.hadoop.hive.metastore.glue.catalogid": "AccountB-catalog-id"
           }
       }]
   }'
   ```

   Hive

   次の例に示すように、 hive.metastore.glue.catalogidプロパティをhive-site分類に設定します。置換 *AccountB -catalog-id* でデータカタログの ID を使用しますAccountB。

   ```
   aws emr-serverless start-job-run \
   ```

```
--application-id "application-id" \
--execution-role-arn "job-role-arn" \
--job-driver '{
    "hive": {
    "query": "s3://amzn-s3-demo-bucket/hive/scripts/create_table.sql",
    "parameters": "--hiveconf hive.exec.scratchdir=s3://amzn-s3-demo-bucket/
hive/scratch --hiveconf hive.metastore.warehouse.dir=s3://amzn-s3-demo-bucket/
hive/warehouse"
    }
}' \
--configuration-overrides '{
    "applicationConfiguration": [{
        "classification": "hive-site",
        "properties": {
            "hive.metastore.glue.catalogid": "AccountB-catalog-id"
        }
    }]
}'
```

AWS Glue データカタログを使用する際の考慮事項

Hive スクリプトADD JARで JARs に補助を追加できます。その他の考慮事項については、AWS「Glue Data Catalog を使用する際の考慮事項」を参照してください。

外部 Hive メタストアの使用

EMR Serverless Spark ジョブと Hive ジョブを設定して、Amazon Aurora や Amazon RDS for My などの外部 Hive メタストアに接続できますSQL。このセクションでは、Amazon RDS Hive メタストアをセットアップし、 を設定しVPC、外部メタストアを使用するように EMR Serverless ジョブを設定する方法について説明します。

外部 Hive メタストアを作成する

1. の手順に従って、プライベートサブネットを使用して Amazon Virtual Private Cloud (Amazon VPC) を作成しますVPC。
2. 新しい Amazon サブネットVPCとプライベートサブネットを使用して EMR Serverless アプリケーションを作成します。EMR Serverless アプリケーションを で設定するとVPC、最初に指定したサブネットごとにElastic Network Interface がプロビジョニングされます。次に、指定されたセキュリティグループをそのネットワークインターフェイスにアタッチします。これにより、

アプリケーションにアクセスコントロールが付与されます。のセットアップ方法の詳細についてはVPC、「」を参照してくださいVPC アクセスの設定。

3. Amazon のプライベートサブネットに MySQL または Aurora PostgreSQL データベースを作成しますVPC。Amazon RDS データベースの作成方法については、「Amazon RDS DB インスタンスの作成」を参照してください。
4. Amazon RDS DB インスタンスの変更 の手順に従って、MySQL または Aurora データベースのセキュリティグループを変更して、EMRサーバーレスセキュリティグループからのJDBC接続を許可します。EMR Serverless RDS セキュリティグループの 1 つからセキュリティグループへのインバウンドトラフィックのルールを追加します。

タイプ	プロトコル	ポート範囲	ソース
すべて TCP	TCP	3306	`emr-serverless-security-group`

Spark オプションの設定

の使用 JDBC

Amazon RDS for MySQL または Amazon Aurora MySQL インスタンスに基づいて Hive メタストアに接続するように EMR Serverless Spark アプリケーションを設定するには、JDBC接続を使用します。ジョブ実行の`spark-submit`パラメータ`--jars`に `mariadb-connector-java.jar`を渡します。

```
aws emr-serverless start-job-run \
  --application-id "application-id" \
  --execution-role-arn "job-role-arn" \
  --job-driver '{
        "sparkSubmit": {
            "entryPoint": "s3://amzn-s3-demo-bucket/scripts/spark-jdbc.py",
            "sparkSubmitParameters": "--jars s3://amzn-s3-demo-bucket/mariadb-
connector-java.jar
            --conf
 spark.hadoop.javax.jdo.option.ConnectionDriverName=org.mariadb.jdbc.Driver
            --conf spark.hadoop.javax.jdo.option.ConnectionUserName=<connection-user-
name>
```

```
            --conf spark.hadoop.javax.jdo.option.ConnectionPassword=<connection-
password>
            --conf spark.hadoop.javax.jdo.option.ConnectionURL=<JDBC-Connection-
string>
            --conf spark.driver.cores=2
            --conf spark.executor.memory=10G
            --conf spark.driver.memory=6G
            --conf spark.executor.cores=4"
        }
    }' \
    --configuration-overrides '{
        "monitoringConfiguration": {
        "s3MonitoringConfiguration": {
            "logUri": "s3://amzn-s3-demo-bucket/spark/logs/"
        }
    }
}'
```

次のコード例は、Amazon の Hive メタストアとやり取りする Spark エントリポイントスクリプトですRDS。

```
from os.path import expanduser, join, abspath
from pyspark.sql import SparkSession
from pyspark.sql import Row
# warehouse_location points to the default location for managed databases and tables
warehouse_location = abspath('spark-warehouse')
spark = SparkSession \
    .builder \
    .config("spark.sql.warehouse.dir", warehouse_location) \
    .enableHiveSupport() \
    .getOrCreate()
spark.sql("SHOW DATABASES").show()
spark.sql("CREATE EXTERNAL TABLE `sampledb`.`sparknyctaxi`(`dispatching_base_num`
 string, `pickup_datetime` string, `dropoff_datetime` string, `pulocationid` bigint,
 `dolocationid` bigint, `sr_flag` bigint) STORED AS PARQUET LOCATION 's3://<s3 prefix>/
nyctaxi_parquet/'")
spark.sql("SELECT count(*) FROM sampledb.sparknyctaxi").show()
spark.stop()
```

thrift サービスの使用

Amazon RDS for MySQL または Amazon Aurora MySQL インスタンスに基づいて Hive メタストアに接続するように EMR Serverless Hive アプリケーションを設定できます。これを行うには、

既存の Amazon EMRクラスターのマスターノードで thrift サーバーを実行します。このオプションは、EMRサーバーレスジョブ設定を簡素化するために使用するスリフトサーバーを備えた Amazon EMRクラスターが既にある場合に最適です。

```
aws emr-serverless start-job-run \
  --application-id "application-id" \
  --execution-role-arn "job-role-arn" \
  --job-driver '{
        "sparkSubmit": {
            "entryPoint": "s3://amzn-s3-demo-bucket/thriftscript.py",
            "sparkSubmitParameters": "--jars s3://amzn-s3-demo-bucket/mariadb-
connector-java.jar
            --conf spark.driver.cores=2
            --conf spark.executor.memory=10G
            --conf spark.driver.memory=6G
            --conf spark.executor.cores=4"
        }
    }' \
    --configuration-overrides '{
        "monitoringConfiguration": {
        "s3MonitoringConfiguration": {
            "logUri": "s3://amzn-s3-demo-bucket/spark/logs/"
        }
    }
}'
```

次のコード例は、ドリフトプロトコルを使用して Hive メタストアに接続するエントリポイントスクリプト (thriftscript.py) です。hive.metastore.uris プロパティは、外部 Hive メタストアから読み取るように設定する必要があります。

```
from os.path import expanduser, join, abspath
from pyspark.sql import SparkSession
from pyspark.sql import Row
# warehouse_location points to the default location for managed databases and tables
warehouse_location = abspath('spark-warehouse')
spark = SparkSession \
    .builder \
    .config("spark.sql.warehouse.dir", warehouse_location) \
    .config("hive.metastore.uris","thrift://thrift-server-host:thift-server-port") \
    .enableHiveSupport() \
    .getOrCreate()
spark.sql("SHOW DATABASES").show()
```

```
spark.sql("CREATE EXTERNAL TABLE sampledb.`sparknyctaxi`( `dispatching_base_num`
 string, `pickup_datetime` string, `dropoff_datetime` string, `pulocationid` bigint,
 `dolocationid` bigint, `sr_flag` bigint) STORED AS PARQUET LOCATION 's3://<s3 prefix>/
nyctaxi_parquet/'")
spark.sql("SELECT * FROM sampledb.sparknyctaxi").show()
spark.stop()
```

Hive オプションを設定する

の使用 JDBC

Amazon RDS MySQL インスタンスまたは Amazon Aurora インスタンスのいずれかで外部 Hive データベースの場所を指定する場合は、デフォルトのメタストア設定を上書きできます。

> **Note**
>
> Hive では、メタストアテーブルへの複数の書き込みを同時に実行できます。2 つのジョブ間でメタストア情報を共有する場合は、同じメタストアテーブルの異なるパーティションに書き込まない限り、同じメタストアテーブルに同時に書き込まないようにしてください。

`hive-site` 分類で次の設定を設定して、外部 Hive メタストアをアクティブ化します。

```
{
    "classification": "hive-site",
    "properties": {
        "hive.metastore.client.factory.class":
 "org.apache.hadoop.hive.ql.metadata.SessionHiveMetaStoreClientFactory",
        "javax.jdo.option.ConnectionDriverName": "org.mariadb.jdbc.Driver",
        "javax.jdo.option.ConnectionURL": "jdbc:mysql://db-host:db-port/db-name",
        "javax.jdo.option.ConnectionUserName": "username",
        "javax.jdo.option.ConnectionPassword": "password"
    }
}
```

スリフトサーバーの使用

Amazon RDS for MySQL または Amazon Aurora M に基づいて Hive メタストアに接続するように EMR Serverless Hive アプリケーションを設定できますySQLinstance。これを行うには、既存の Amazon EMRクラスターのメインノードで thrift サーバーを実行します。このオプションは、スリフ

トサーバーを実行する Amazon EMRクラスターが既にあり、EMRサーバーレスジョブ設定を使用する場合に最適です。

EMR Serverless がリモートスリフトメタストアにアクセスできるように、hive-site分類で次の設定を設定します。外部 Hive メタストアから読み取るように hive.metastore.urisプロパティを設定する必要があります。

```
{
    "classification": "hive-site",
    "properties": {
        "hive.metastore.client.factory.class":
 "org.apache.hadoop.hive.ql.metadata.SessionHiveMetaStoreClientFactory",
        "hive.metastore.uris": "thrift://thrift-server-host:thirft-server-port"
    }
}
```

外部メタストアを使用する場合の考慮事項

- MariaDB と互換性のあるデータベースをメタストアJDBCとして設定できます。これらのデータベースの例は、MariaDB 、My SQL、Amazon Aurora RDS用です。
- メタストアは自動初期化されません。メタストアが Hive バージョンのスキーマで初期化されていない場合は、Hive スキーマツール を使用します。
- EMR Serverless は Kerberos 認証をサポートしていません。EMR Serverless Spark ジョブまたは Hive ジョブでは、Kerberos 認証で thrift メタストアサーバーを使用することはできません。

別の の S3 データへのアクセス AWS EMR Serverless からの アカウント

1 つの から Amazon EMR Serverless ジョブを実行できます。 AWS アカウントを作成し、別の に属する Amazon S3 バケットのデータにアクセスするように設定する AWS アカウント。このページでは、EMRサーバーレスから S3 へのクロスアカウントアクセスを設定する方法について説明します。

EMR Serverless で実行されるジョブは、S3 バケットポリシーまたは引き受けたロールを使用して、別の から Amazon S3 のデータにアクセスできます。 AWS アカウント。

前提条件

Amazon EMR Serverless のクロスアカウントアクセスを設定するには、2 つにサインインしているときにタスクを完了する必要があります。 AWS accounts:

- **AccountA** – これは AWS Amazon EMR Serverless アプリケーションを作成した アカウント。クロスアカウントアクセスを設定する前に、このアカウントで次の準備が整っている必要があります。
  - ジョブを実行する Amazon EMR Serverless アプリケーション。
  - アプリケーションでジョブを実行するために必要なアクセス許可を持つジョブ実行ロール。詳細については、「Amazon EMR Serverless のジョブランタイムロール」を参照してください。
- **AccountB** - これは AWS Amazon EMR Serverless ジョブがアクセスする S3 バケットを含む アカウント。

S3 バケットポリシーを使用してクロスアカウント S3 データにアクセスする

で S3 バケットにアクセスするには account B 送信元 account Aで、次のポリシーを の S3 バケットにアタッチします。account B.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "Example permissions 1",
         "Effect": "Allow",
         "Principal": {
            "AWS": "arn:aws:iam::AccountA:root"
         },
         "Action": [
            "s3:ListBucket"
         ],
         "Resource": [
            "arn:aws:s3:::bucket_name_in_AccountB"
         ]
      },
      {
         "Sid": "Example permissions 2",
         "Effect": "Allow",
```

```
            "Principal": {
               "AWS": "arn:aws:iam::AccountA:root"
            },
            "Action": [
               "s3:PutObject",
               "s3:GetObject",
               "s3:DeleteObject"
            ],
            "Resource": [
               "arn:aws:s3:::bucket_name_in_AccountB/*"
            ]
         }
      ]
}
```

S3 バケットポリシーによる S3 クロスアカウントアクセスの詳細については、Amazon Simple Storage Service ユーザーガイドの「例 2: バケット所有者がクロスアカウントバケットのアクセス許可を付与する」を参照してください。

引き受けたロールを使用してクロスアカウント S3 データにアクセスする

Amazon EMR Serverless のクロスアカウントアクセスを設定するもう 1 つの方法は、 からの `AssumeRole`アクションを使用することです。 AWS Security Token Service (AWS STS). AWS STS は、ユーザーの権限が制限された一時的な認証情報をリクエストできるグローバルウェブサービスです。で作成した一時的なセキュリティ認証情報を使用して、EMRサーバーレスと Amazon S3 をAPI 呼び出すことができます`AssumeRole`。

次の手順は、引き受けたロールを使用して EMR Serverless からクロスアカウント S3 データにアクセスする方法を示しています。

1. Amazon S3 バケットを作成する *`cross-account-bucket`*の `AccountB`。詳細については、「Amazon Simple Storage Service ユーザーガイド」の「バケットの作成」を参照してください。DynamoDB へのクロスアカウントアクセスが必要な場合は、`AccountB` で DynamoDB テーブルを作成することもできます。詳細については、「Amazon DynamoDB デベロッパーガイド」の「DynamoDB テーブルの作成」を参照してください。 DynamoDB
2. `AccountB` にアクセスできる`Cross-Account-Role-B`IAMロールを に作成する *`cross-account-bucket`*.
   a. にサインインする AWS Management Console でIAMコンソールを開きますhttps://console.aws.amazon.com/iam/。

b. [ロール] を選択し、新しいロール Cross-Account-Role-B を作成します。IAM ロールの作成方法の詳細については、「 ユーザーガイド」のIAM「ロールの作成IAM」を参照してください。

c. Cross-Account-Role-B へのアクセス許可を指定する IAMポリシーを作成する *cross-account-bucket* 次のポリシーステートメントが示すように、S3 バケット。次に、IAMポリシーを にアタッチしますCross-Account-Role-B。詳細については、「 ユーザーガイド」のIAM「ポリシーの作成IAM」を参照してください。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::cross-account-bucket",
                "arn:aws:s3:::cross-account-bucket/*"
            ]
        }
    ]
}
```

DynamoDB アクセスが必要な場合は、クロスアカウント DynamoDB テーブルへのアクセス許可を指定する IAMポリシーを作成します。次に、IAMポリシーを にアタッチしますCross-Account-Role-B。詳細については、「 ユーザーガイド」の「Amazon DynamoDB : 特定のテーブルへのアクセスを許可するIAM」を参照してください。

以下は、DynamoDB テーブル へのアクセスを許可するポリシーですCrossAccountTable。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:*",
            "Resource": "arn:aws:dynamodb:MyRegion:AccountB:table/CrossAccountTable"
        }
    ]
}
```

3. Cross-Account-Role-B ロールの信頼関係を編集します。

   a. ロールの信頼関係を設定するには、ステップ 2 で作成したロールのコンソールで信頼関係タブを選択します。 IAM Cross-Account-Role-B

   b. [信頼関係の編集] を選択します。

   c. 次のポリシードキュメントを追加します。これにより、 Job-Execution-Role-Aで Cross-Account-Role-Bロールを引き受けAccountAることができます。

```
{
"Version": "2012-10-17",
  "Statement": [
    {
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::AccountA:role/Job-Execution-Role-A"
        },
        "Action": "sts:AssumeRole"
    }
  ]
}
```

4. Job-Execution-Role-A AccountA での付与 AWS STS AssumeRole を引き受ける アクセス許可Cross-Account-Role-B。

   a. のIAMコンソールで AWS アカウント AccountAで、 を選択しますJob-Execution-Role-A。

   b. 次のポリシーステートメントを Job-Execution-Role-A に追加して、Cross-Account-Role-B ロールに対して AssumeRole アクションを許可します。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::AccountB:role/Cross-Account-Role-B"
        }
    ]
}
```

引き受けたロールの例

単一の引き受けたロールを使用して、アカウント内のすべての S3 リソースにアクセスすることも、Amazon EMR 6.11 以降では、異なるクロスアカウント S3 バケットにアクセスするときに引き受ける複数のIAMロールを設定することもできます。

トピック

1 つの引き受けたロールで S3 リソースにアクセスする

> **Note**
>
> 単一の引き受けたロールを使用するようにジョブを設定すると、ジョブ全体のすべての S3 リソースは、`entryPoint`スクリプトを含むそのロールを使用します。

単一の引き受けたロールを使用してアカウント B のすべての S3 リソースにアクセスする場合は、次の設定を指定します。

1. EMRFS 設定を `fs.s3.customAWSCredentialsProvider`に指定しますspark.hadoop.fs.s3.customAWSCredentialsProvider=com.amazonaws.emr.AssumeRoleA
2. Spark では、 `spark.emr-serverless.driverEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN`と を使用してドライバーと エグゼキューターの環境変数`spark.executorEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN`を指定します。
3. Hive の場合は、`hive.emr-serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN`、`tez.am.emr-serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN`、および を使用して、Hive ドライバー、Tez アプリケーションマスター、および Tez タスクコンテナの環境変数`tez.task.emr-serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN`を指定します。

次の例は、引き受けたロールを使用して、クロスアカウントアクセスでEMRサーバーレスジョブの実行を開始する方法を示しています。

Spark

次の例は、引き受けたロールを使用して、S3 へのクロスアカウントアクセスで EMR Serverless Spark ジョブの実行を開始する方法を示しています。

```
aws emr-serverless start-job-run \
    --application-id application-id \
    --execution-role-arn job-role-arn \
    --job-driver '{
        "sparkSubmit": {
            "entryPoint": "entrypoint_location",
            "entryPointArguments": [":argument_1:", ":argument_2:"],
            "sparkSubmitParameters": "--conf spark.executor.cores=4 --conf
 spark.executor.memory=20g --conf spark.driver.cores=4 --conf spark.driver.memory=8g
 --conf spark.executor.instances=1"
        }
    }' \
     --configuration-overrides '{
        "applicationConfiguration": [{
            "classification": "spark-defaults",
            "properties": {
                "spark.hadoop.fs.s3.customAWSCredentialsProvider":
 "spark.hadoop.fs.s3.customAWSCredentialsProvider=com.amazonaws.emr.AssumeRoleAWSCredentials
                "spark.emr-serverless.driverEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN":
 "arn:aws:iam::AccountB:role/Cross-Account-Role-B",
                "spark.executorEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN":
 "arn:aws:iam::AccountB:role/Cross-Account-Role-B"
            }
        }]
    }'
```

Hive

次の例は、引き受けたロールを使用して、S3 へのクロスアカウントアクセスで EMR Serverless Hive ジョブの実行を開始する方法を示しています。

```
aws emr-serverless start-job-run \
    --application-id application-id \
    --execution-role-arn job-role-arn \
    --job-driver '{
        "hive": {
            "query": "query_location",
            "parameters": "hive_parameters"
```

```
        }
    }' \
    --configuration-overrides '{
        "applicationConfiguration": [{
            "classification": "hive-site",
            "properties": {
                "fs.s3.customAWSCredentialsProvider":
 "com.amazonaws.emr.serverless.credentialsprovider.AssumeRoleAWSCredentialsProvider",
                "hive.emr-serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN":
 "arn:aws:iam::AccountB:role/Cross-Account-Role-B",
                "tez.am.emr-serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN":
 "arn:aws:iam::AccountB:role/Cross-Account-Role-B",
                "tez.task.emr-
serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN":
 "arn:aws:iam::AccountB:role/Cross-Account-Role-B"
            }
        }]
    }'
```

複数の引き受けたロールを持つ S3 リソースにアクセスする

EMR Serverless リリース 6.11.0 以降では、異なるクロスアカウントバケットにアクセスするときに引き受ける複数のIAMロールを設定できます。アカウント B で引き受けたロールが異なる異なる S3 リソースにアクセスする場合は、ジョブの実行を開始するときに次の設定を使用します。

1. EMRFS 設定を `fs.s3.customAWSCredentialsProvider`に指定しますcom.amazonaws.emr.serverless.credentialsprovider.BucketLevelAssumeRoleCredent
2. S3 バケット名から引き受けるアカウント B のIAMロールへのマッピングfs.s3.bucketLevelAssumeRoleMappingを定義するEMRFS設定を指定します。値は の形式である必要がありますbucket1->role1;bucket2->role2。

例えば、 arn:aws:iam::*AccountB*:role/Cross-Account-Role-B-1を使用してバケット にアクセスしbucket1、 arn:aws:iam::*AccountB*:role/Cross-Account-Role-B-2を使用してバケット にアクセスできますbucket2。次の例は、複数の引き受けたロールを介してクロスアカウントアクセスでEMRサーバーレスジョブの実行を開始する方法を示しています。

Spark

次の例は、複数の引き受けたロールを使用して EMR Serverless Spark ジョブ実行を作成する方法を示しています。

```
aws emr-serverless start-job-run \
    --application-id application-id \
    --execution-role-arn job-role-arn \
    --job-driver '{
        "sparkSubmit": {
            "entryPoint": "entrypoint_location",
            "entryPointArguments": [":argument_1:", ":argument_2:"],
            "sparkSubmitParameters": "--conf spark.executor.cores=4 --conf
 spark.executor.memory=20g --conf spark.driver.cores=4 --conf spark.driver.memory=8g
 --conf spark.executor.instances=1"
        }
    }' \
     --configuration-overrides '{
        "applicationConfiguration": [{
            "classification": "spark-defaults",
            "properties": {
                "spark.hadoop.fs.s3.customAWSCredentialsProvider":
 "com.amazonaws.emr.serverless.credentialsprovider.BucketLevelAssumeRoleCredentialsProvider"
                "spark.hadoop.fs.s3.bucketLevelAssumeRoleMapping":
 "bucket1->arn:aws:iam::AccountB:role/Cross-Account-Role-B-1;bucket2-
>arn:aws:iam::AccountB:role/Cross-Account-Role-B-2"
            }
        }]
    }'
```

Hive

次の例は、複数の引き受けたロールを使用して EMR Serverless Hive ジョブ実行を作成する方法を示しています。

```
aws emr-serverless start-job-run \
    --application-id application-id \
    --execution-role-arn job-role-arn \
    --job-driver '{
        "hive": {
            "query": "query_location",
            "parameters": "hive_parameters"
        }
    }' \
    --configuration-overrides '{
        "applicationConfiguration": [{
            "classification": "hive-site",
            "properties": {
```

```
                "fs.s3.customAWSCredentialsProvider":
 "com.amazonaws.emr.serverless.credentialsprovider.AssumeRoleAWSCredentialsProvider",
                "fs.s3.bucketLevelAssumeRoleMapping": "bucket1-
>arn:aws:iam::AccountB:role/Cross-Account-Role-B-1;bucket2-
>arn:aws:iam::AccountB:role/Cross-Account-Role-B-2"
            }
        }]
    }'
```

EMR Serverless でのエラーのトラブルシューティング

以下の情報は、Amazon EMR Serverless の使用時に発生する可能性がある一般的な問題の診断と修正に役立ちます。

トピック

- エラー: 最大許容容量の制限を超えました。
- エラー: 設定された最大容量を超えました。後で再試行してください。
- エラー: S3 アクセスが拒否されました。必要な S3 リソースのジョブランタイムロールの S3 アクセス許可を確認してください。
- エラー： ModuleNotFoundError: <module> という名前のモジュールがありません。EMR Serverless で Python ライブラリを使用する方法については、ユーザーガイドを参照してください。
- エラー: 実行ロール <ロール名> が存在しないか、必要な信頼関係で設定されていないため、継承できませんでした。

エラー: 最大許容容量の制限を超えました。

このエラーは、アプリケーションが設定された最大容量制限を超えたため、EMRサーバーレスがジョブを送信できなかったことを示します。アプリケーションの最大容量制限を引き上げます。

エラー: 設定された最大容量を超えました。後で再試行してください。

このエラーは、アプリケーションが設定された最大容量制限を超えたため、EMRサーバーレスが新しいジョブを開始できなかったことを示します。アプリケーションの最大容量制限を引き上げます。

エラー: S3 アクセスが拒否されました。必要な S3 リソースのジョブランタイムロールの S3 アクセス許可を確認してください。

このエラーは、ジョブが S3 リソースにアクセスできないことを示します。ジョブランタイムロールに、ジョブが使用する必要がある S3 リソースへのアクセス許可があることを確認します。ランタイムロールの詳細については、「」を参照してくださいAmazon EMR Serverless のジョブランタイムロール。

エラー： ModuleNotFoundError: <module> という名前のモジュールがありません。EMR Serverless で Python ライブラリを使用する方法については、ユーザーガイドを参照してください。

このエラーは、Python モジュールが Spark ジョブで使用できないことを示します。依存する Python ライブラリがジョブで使用できることを確認します。Python ライブラリをパッケージ化する方法の詳細については、「」を参照してくださいServerless での Python EMR ライブラリの使用。

エラー: 実行ロール <ロール名> が存在しないか、必要な信頼関係で設定されていないため、継承できませんでした。

このエラーは、ジョブに指定したジョブランタイムロールが存在しないか、ロールに EMR Serverless アクセス許可の信頼関係がないことを示します。IAM ロールが存在することを確認し、ロールの信頼ポリシーが正しく設定されていることを確認するには、「」の手順を参照してくださいAmazon EMR Serverless のジョブランタイムロール。

EMR Studio を使用して EMR Serverless でインタラクティブワークロードを実行する

概要

インタラクティブアプリケーションは、インタラクティブ機能が有効になっているEMRサーバーレスアプリケーションです。Amazon EMR Serverless インタラクティブアプリケーションを使用すると、Amazon EMR Studio で管理されている Jupyter Notebook を使用してインタラクティブワークロードを実行できます。これにより、データエンジニア、データサイエンティスト、データアナリストは EMR Studio を使用して、Amazon S3 や Amazon DynamoDB などのデータストア内のデータセットでインタラクティブな分析を実行できます。

EMR Serverless のインタラクティブアプリケーションのユースケースは次のとおりです。

- データエンジニアは EMR Studio のIDEエクスペリエンスを使用してETLスクリプトを作成します。このスクリプトは、オンプレミスからデータを取り込み、分析のためにデータを変換し、Amazon S3 にデータを保存します。
- データサイエンティストはノートブックを使用してデータセットを探索し、機械学習 (ML) モデルをトレーニングしてデータセット内の異常を検出します。
- データアナリストはデータセットを調べ、ビジネスダッシュボードなどのアプリケーションを更新するための日次レポートを生成するスクリプトを作成します。

前提条件

EMR Serverless でインタラクティブワークロードを使用するには、次の要件を満たす必要があります。

- EMR サーバーレスインタラクティブアプリケーションは、Amazon EMR 6.14.0 以降でサポートされています。
- インタラクティブアプリケーションにアクセスし、送信するワークロードを実行し、EMRStudio からインタラクティブノートブックを実行するには、特定のアクセス許可とロールが必要です。詳細については、「インタラクティブワークロードに必要なアクセス許可」を参照してください。

インタラクティブワークロードに必要なアクセス許可

EMR Serverless へのアクセスに必要な基本的なアクセス許可に加えて、ID またはロールに追加のアクセス許可を設定する必要があります。 IAM

インタラクティブアプリケーションにアクセスするには

EMR Studio のユーザーおよび Workspace アクセス許可を設定します。詳細については、「Amazon EMR管理ガイド」のEMR「Studio ユーザーアクセス許可の設定」を参照してください。

EMR Serverless で送信するワークロードを実行するには

ジョブランタイムロールを設定します。詳細については、「ジョブランタイムロールを作成する」を参照してください。

EMR Studio からインタラクティブノートブックを実行するには

Studio ユーザーのIAMポリシーに次のアクセス許可を追加します。

- **`emr-serverless:AccessInteractiveEndpoints`** - として指定したインタラクティブアプリケーションにアクセスして接続するアクセス許可を付与しますResource。このアクセス許可は、EMRStudio Workspace からEMRサーバーレスアプリケーションにアタッチするために必要です。
- **`iam:PassRole`** - アプリケーションにアタッチするときに使用する予定のIAM実行ロールにアクセスするアクセス許可を付与します。EMR Studio Workspace からEMRサーバーレスアプリケーションにアタッチするには、適切なPassRoleアクセス許可が必要です。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EMRServerlessInteractiveAccess",
            "Effect": "Allow",
            "Action": "emr-serverless:AccessInteractiveEndpoints",
            "Resource": "arn:aws:emr-serverless:Region:account:/applications/*"
        },
        {
            "Sid": "EMRServerlessRuntimeRoleAccess",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "interactive-execution-role-ARN",
            "Condition": {
```

```
                "StringLike": {
                    "iam:PassedToService": "emr-serverless.amazonaws.com"
                }
            }
        }
    ]
}
```

インタラクティブアプリケーションの設定

以下の大まかなステップを使用して、 の Amazon EMR Studio からインタラクティブ機能を備えたEMRサーバーレスアプリケーションを作成します。 AWS Management Console.

1. の手順に従ってアプリケーションAmazon EMR Serverless の開始方法を作成します。
2. 次に、EMRStudio からワークスペースを起動し、コンピューティングオプションとしてEMRサーバーレスアプリケーションにアタッチします。詳細については、Serverless 入門ドキュメントのステップ 2 の「インタラクティブワークロード」タブを参照してください。 EMR

アプリケーションを Studio Workspace にアタッチすると、まだ実行されていない場合、アプリケーションが自動的にトリガーされます。アプリケーションを Workspace にアタッチする前に、アプリケーションを事前に起動して準備しておくこともできます。

インタラクティブアプリケーションの考慮事項

- EMR サーバーレスインタラクティブアプリケーションは、Amazon EMR 6.14.0 以降でサポートされています。
- EMR Studio は、EMRサーバーレスインタラクティブアプリケーションと統合されている唯一のクライアントです。サーバーEMRレスインタラクティブアプリケーションでは、Workspace コラボレーション、SQLExplorer、およびノートブックのプログラムによる実行の EMR Studio 機能はサポートされていません。
- インタラクティブアプリケーションは Spark エンジンでのみサポートされています。
- インタラクティブアプリケーションは Python 3 PySpark と Spark Scala カーネルをサポートします。
- 1 つのインタラクティブアプリケーションで最大 25 個のノートブックを同時に実行できます。
- インタラクティブアプリケーションでセルフホスト型 Jupyter Notebook をサポートするエンドポイントやAPIインターフェイスはありません。

- 起動エクスペリエンスを最適化するには、ドライバーとエグゼキューターの事前に初期化された容量を設定し、アプリケーションを事前に起動することをお勧めします。アプリケーションを起動する前に、Workspace にアタッチする準備が整っていることを確認します。

```
aws emr-serverless start-application \
--application-id your-application-id
```

- デフォルトでは、 autoStopConfigはアプリケーションに対して有効になっています。これにより、アイドル時間が 30 分経過するとアプリケーションがシャットダウンされます。この設定は、create-applicationまたは update-applicationリクエストの一部として変更できます。
- インタラクティブアプリケーションを使用する場合は、ノートブックを実行するカーネル、ドライバー、エグゼキューターの事前統合容量を設定することをお勧めします。Spark インタラクティブセッションごとに 1 つのカーネルと 1 つのドライバーが必要なため、EMRServerless は、事前に初期化されたドライバーごとに、事前に初期化されたカーネルワーカーを維持します。デフォルトでは、EMRServerless は、ドライバーに事前初期化された容量を指定しなくても、アプリケーション全体で 1 つのカーネルワーカーの事前初期化された容量を維持します。各カーネルワーカーは 4 vCPU と 16 GB のメモリを使用します。現在の料金情報については、「Amazon のEMR 料金」ページを参照してください。
- には十分な vCPU サービスクォータが必要です AWS アカウント インタラクティブワークロードを実行する 。Lake Formation 対応ワークロードを実行しない場合は、少なくとも 24 v をお勧めしますCPU。その場合は、少なくとも 28 v をお勧めしますCPU。
- EMR サーバーレスは、60 分以上アイドル状態になっている場合、ノートブックからカーネルを自動的に終了します。EMR Serverless は、ノートブックセッション中に最後に完了したアクティビティからのカーネルアイドル時間を計算します。現在、カーネルアイドルタイムアウト設定を変更することはできません。
- インタラクティブワークロードで Lake Formation を有効にするには、サーバーレスアプリケーション を作成するときに、 runtime-configuration オブジェクトの spark-defaults 分類spark.emr-serverless.lakeformation.enabledtrueで 設定を に設定します。 EMR EMR Serverless で Lake Formation を有効にする方法の詳細については、「Amazon で Lake Formation を有効にするEMR」を参照してください。

Apache Livy エンドポイントを介してEMRサーバーレスでインタラクティブワークロードを実行する

Amazon EMRリリース 6.14.0 以降では、EMRサーバーレスアプリケーションの作成中に Apache Livy エンドポイントを作成して有効にし、セルフホストノートブックまたはカスタムクライアントを使用してインタラクティブワークロードを実行できます。Apache Livy エンドポイントには、次の利点があります。

- Jupyter Notebook を介して Apache Livy エンドポイントに安全に接続し、Apache Livy のRESTインターフェイスを使用して Apache Spark ワークロードを管理できます。
- Apache Spark ワークロードのデータを使用するインタラクティブなウェブアプリケーションには、Apache Livy RESTAPIオペレーションを使用します。

前提条件

Serverless で Apache Livy EMR エンドポイントを使用するには、次の要件を満たす必要があります。

- 「Amazon EMR Serverless の開始方法」のステップを完了します。
- Apache Livy エンドポイントを介してインタラクティブワークロードを実行するには、特定のアクセス許可とロールが必要です。詳細については、「インタラクティブワークロードに必要なアクセス許可」を参照してください。

必要なアクセス許可

EMR Serverless へのアクセスに必要なアクセス許可に加えて、Apache Livy エンドポイントにアクセスしてアプリケーションを実行するには、IAMロールに次のアクセス許可も追加する必要があります。

- `emr-serverless:AccessLivyEndpoints` – として指定した Livy 対応アプリケーションにアクセスして接続するアクセス許可を付与しますResource。Apache Livy エンドポイントから利用可能なRESTAPIオペレーションを実行するには、このアクセス許可が必要です。
- `iam:PassRole` – Apache Livy セッションの作成中にIAM実行ロールにアクセスするアクセス許可を付与します。EMR サーバーレスはこのロールを使用してワークロードを実行します。

- `emr-serverless:GetDashboardForJobRun` – は、Spark Live UI とドライバーのログリンクを生成するアクセス許可を付与し、Apache Livy セッションの結果の一部としてログへのアクセスを提供します。

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Sid": "EMRServerlessInteractiveAccess",
            "Effect": "Allow",
            "Action": "emr-serverless:AccessLivyEndpoints",
            "Resource": "arn:aws:emr-serverless:<AWS_REGION>:account:/applications/*"
        },
        {
            "Sid": "EMRServerlessRuntimeRoleAccess",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "execution-role-ARN",
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": "emr-serverless.amazonaws.com"
                }
            }
        },
        {
            "Sid": "EMRServerlessDashboardAccess",
            "Effect": "Allow",
            "Action": "emr-serverless:GetDashboardForJobRun",
            "Resource": "arn:aws:emr-serverless:<AWS_REGION>:account:/applications/*"
        }
    ]
}
```

使用開始

1. Apache Livy 対応アプリケーションを作成するには、次のコマンドを実行します。

```
 aws emr-serverless create-application \
--name my-application-name \
--type 'application-type' \
 --release-label <Amazon EMR-release-version>
--interactive-configuration '{"livyEndpointEnabled": true}'
```

2. EMR Serverless がアプリケーションを作成したら、アプリケーションを起動して Apache Livy エンドポイントを使用可能にします。

```
aws emr-serverless start-application \
--application-id application-id
```

次のコマンドを使用して、アプリケーションのステータスを確認します。ステータスが になるとSTARTED、Apache Livy エンドポイントにアクセスできます。

```
aws emr-serverless get-application \
--region <AWS_REGION> --application-id >application_id>
```

3. エンドポイントURLにアクセスするには、以下を使用します。

```
https://_<application-id>_.livy.emr-serverless-
services._<AWS_REGION>_.amazonaws.com
```

エンドポイントの準備ができたら、ユースケースに基づいてワークロードを送信できます。エンドポイントへのすべてのリクエストにSIGv4プロトコルで署名し、認証ヘッダーを渡す必要があります。ワークロードを実行するには、次の方法を使用できます。

- HTTP client – カスタムHTTPクライアントを使用して Apache Livy エンドポイントAPIオペレーションを送信する必要があります。
- Sparkmagic カーネル — sparkmagic カーネルをローカルで実行し、Jupyter ノートブックでインタラクティブクエリを送信する必要があります。

HTTP クライアント

Apache Livy セッションを作成するには、リクエスト本文の confパラメータemr-serverless.session.executionRoleArnで を送信する必要があります。次の例は、サンプルPOST /sessionsリクエストです。

```
{
    "kind": "pyspark",
    "heartbeatTimeoutInSecond": 60,
    "conf": {
        "emr-serverless.session.executionRoleArn": "<executionRoleArn>"
    }
```

```
}
```

次の表に、使用可能なすべての Apache Livy APIオペレーションを示します。

API オペレーション	説明
GET /セッション	アクティブなすべてのインタラクティブセッションのリストを返します。
POST /セッション	spark または pyspark を使用して新しいインタラクティブセッションを作成します。
GET /セッション/<*sessionId* >	セッション情報を返します。
GET /セッション/<*sessionId* >/ステート	セッションの状態を返します。
DELETE /セッション/<*sessionId* >	セッションを停止して削除します。
GET /セッション/<*sessionId* >/ステートメント	セッション内のすべてのステートメントを返します。
POST /セッション/<*sessionId* >/ステートメント	セッションでステートメントを実行します。
GET /セッション/<*sessionId* >/ステートメント/<*statementId* >	セッション内の指定されたステートメントの詳細を返します。
POST /セッション/<*sessionId* >/ステートメント/<*statementId* >/キャンセル	このセッションで指定されたステートメントをキャンセルします。

Apache Livy エンドポイントへのリクエストの送信

HTTP クライアントから Apache Livy エンドポイントにリクエストを直接送信することもできます。これにより、ノートブックの外部でユースケースのコードをリモートで実行できます。

エンドポイントへのリクエストの送信を開始する前に、次のライブラリがインストールされていることを確認してください。

```
pip3 install botocore awscrt requests
```

エンドポイントにHTTPリクエストを直接送信する Python スクリプトの例を次に示します。

```
from botocore import crt
import requests
from botocore.awsrequest import AWSRequest
from botocore.credentials import Credentials
import botocore.session
import json, pprint, textwrap

endpoint = 'https://<application_id>.livy.emr-serverless-
services-<AWS_REGION>.amazonaws.com'
headers = {'Content-Type': 'application/json'}

session = botocore.session.Session()
signer = crt.auth.CrtS3SigV4Auth(session.get_credentials(), 'emr-serverless',
 '<AWS_REGION>')


### Create session request

data = {'kind': 'pyspark', 'heartbeatTimeoutInSecond': 60, 'conf': { 'emr-
serverless.session.executionRoleArn': 'arn:aws:iam::123456789012:role/role1'}}

request = AWSRequest(method='POST', url=endpoint + "/sessions", data=json.dumps(data),
 headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r = requests.post(prepped.url, headers=prepped.headers, data=json.dumps(data))

pprint.pprint(r.json())


### List Sessions Request

request = AWSRequest(method='GET', url=endpoint + "/sessions", headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)
```

```
prepped = request.prepare()

r2 = requests.get(prepped.url, headers=prepped.headers)
pprint.pprint(r2.json())


### Get session state

session_url = endpoint + r.headers['location']

request = AWSRequest(method='GET', url=session_url, headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r3 = requests.get(prepped.url, headers=prepped.headers)

pprint.pprint(r3.json())


### Submit Statement

data = {
      'code': "1 + 1"
}

statements_url = endpoint + r.headers['location'] + "/statements"

request = AWSRequest(method='POST', url=statements_url, data=json.dumps(data),
 headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r4 = requests.post(prepped.url, headers=prepped.headers, data=json.dumps(data))

pprint.pprint(r4.json())
```

```
### Check statements results

specific_statement_url = endpoint + r4.headers['location']

request = AWSRequest(method='GET', url=specific_statement_url, headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r5 = requests.get(prepped.url, headers=prepped.headers)

pprint.pprint(r5.json())


### Delete session


session_url = endpoint + r.headers['location']

request = AWSRequest(method='DELETE', url=session_url, headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r6 = requests.delete(prepped.url, headers=prepped.headers)

pprint.pprint(r6.json())
```

Sparkmagic カーネル

sparkmagic をインストールする前に、 AWS sparkmagic をインストールするインスタンスの 認証情報

1. インストール手順に従って sparkmagic をインストールします。最初の 4 つのステップのみを実行する必要があることに注意してください。

2. sparkmagic カーネルはカスタム認証をサポートしているため、認証を sparkmagic カーネルと統合して、すべてのリクエストSIGv4に署名できます。
3. EMR Serverless カスタム認証ツールをインストールします。

```
pip install emr-serverless-customauth
```

4. 次に、Sparkmagic 設定 json ファイルURLでカスタム認証機能と Apache Livy エンドポイントへのパスを指定します。次のコマンドを使用して設定ファイルを開きます。

```
vim ~/.sparkmagic/config.json
```

次に、サンプル config.json ファイルを示します。

```
{
"kernel_python_credentials" : {
    "username": "",
    "password": "",
    "url": "https://<application-id>.livy.emr-serverless-
services.<AWS_REGION>.amazonaws.com",
    "auth": "Custom_Auth"
  },

  "kernel_scala_credentials" : {
    "username": "",
    "password": "",
    "url": "https://<application-id>.livy.emr-serverless-
services.<AWS_REGION>.amazonaws.com",
    "auth": "Custom_Auth"
  },
  "authenticators": {
    "None": "sparkmagic.auth.customauth.Authenticator",
    "Basic_Access": "sparkmagic.auth.basic.Basic",
    "Custom_Auth":
 "emr_serverless_customauth.customauthenticator.EMRServerlessCustomSigV4Signer"
  },
  "livy_session_startup_timeout_seconds": 600,
  "ignore_ssl_errors": false
}
```

5. Jupyter ラボを起動します。最後のステップで設定したカスタム認証を使用する必要があります。

6. その後、次のノートブックコマンドとコードを実行して開始できます。

```
%%info //Returns the information about the current sessions.
```

```
%%configure -f //Configure information specific to a session. We supply
 executionRoleArn in this example. Change it for your use case.
{
    "driverMemory": "4g",
    "conf": {
          "emr-serverless.session.executionRoleArn":
 "arn:aws:iam::123456789012:role/JobExecutionRole"
    }
}
```

```
<your code>//Run your code to start the session
```

内部的には、各命令は、設定された Apache Livy エンドポイント を介して各 Apache Livy APIオペレーションを呼び出しますURL。その後、ユースケースに応じて指示書を作成できます。

考慮事項

Apache Livy エンドポイントを介してインタラクティブワークロードを実行する場合は、次の考慮事項を考慮してください。

- EMR Serverless は、発信者プリンシパルを使用してセッションレベルの分離を維持します。セッションを作成する発信者プリンシパルは、そのセッションにアクセスできる唯一のプリンシパルです。より詳細な分離のために、認証情報を引き受けるときにソース ID を設定できます。この場合、EMRサーバーレスは発信者プリンシパルとソース ID の両方に基づいてセッションレベルの分離を適用します。ソース ID の詳細については、「引き受けたロールで実行されたアクションのモニタリングと制御」を参照してください。
- Apache Livy エンドポイントは、EMRサーバーレスリリース 6.14.0 以降でサポートされています。
- Apache Livy エンドポイントは、Apache Spark エンジンでのみサポートされています。
- Apache Livy エンドポイントは Scala Spark と をサポートしています PySpark。
- デフォルトでは、 `autoStopConfig`はアプリケーションで有効になっています。つまり、アプリケーションはアイドル状態から 15 分後にシャットダウンします。この設定は、 `create-application`または `update-application`リクエストの一部として変更できます。

- 1 つの Apache Livy エンドポイント対応アプリケーションで最大 25 の同時セッションを実行できます。
- 最適な起動エクスペリエンスを得るには、ドライバーとエグゼキュター用に事前に初期化された容量を設定することをお勧めします。
- Apache Livy エンドポイントに接続する前に、アプリケーションを手動で起動する必要があります。
- には十分な vCPU サービスクォータが必要です AWS アカウント Apache Livy エンドポイントを使用してインタラクティブワークロードを実行するには、 を使用します。少なくとも 24 v をお勧めしますCPU。
- デフォルトの Apache Livy セッションタイムアウトは 1 時間です。ステートメントを 1 時間実行しない場合、Apache Livy はセッションを削除し、ドライバーとエグゼキュターを解放します。この設定は変更できません。
- アクティブなセッションのみが Apache Livy エンドポイントとやり取りできます。セッションが終了、キャンセル、または終了すると、Apache Livy エンドポイントからセッションにアクセスできなくなります。

ログ記録とモニタリング

モニタリングは、EMRサーバーレスアプリケーションとジョブの信頼性、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、EMRサーバーレス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。

トピック

ログの保存

EMR Serverless でのジョブの進行状況をモニタリングし、ジョブの失敗をトラブルシューティングするには、EMRServerless がアプリケーションログを保存して提供する方法を選択できます。ジョブ実行を送信するときは、ログ記録オプション CloudWatch としてマネージドストレージ、Amazon S3、Amazon を指定できます。

では CloudWatch、使用するログタイプとログの場所を指定するか、デフォルトのタイプと場所を受け入れることができます。 CloudWatch ログの詳細については、「」を参照してくださいthe section called “Amazon CloudWatch”。マネージドストレージと S3 ログ記録では、次の表に、マネージドストレージ 、Amazon S3 バケット 、またはその両方を選択した場合に期待できるログの場所と UI の可用性を示します。 Amazon S3

オプション	イベントログ	コンテナログ	アプリケーション UI
マネージドストレージ	マネージドストレージに保存	マネージドストレージに保存	サポート

オプション	イベントログ	コンテナログ	アプリケーション UI
マネージドストレージと S3 バケットの両方	両方の場所に保存	S3 バケットに保存	サポート
Amazon S3 バケット	S3 バケットに保存	S3 バケットに保存	サポートされていません[1]

[1] マネージドストレージオプションを選択したままにすることをお勧めします。それ以外の場合は、組み込みアプリケーション を使用できませんUIs。

マネージドストレージを使用した EMR Serverless のログ記録

デフォルトでは、EMRServerless はアプリケーションログを Amazon EMRマネージドストレージに最大 30 日間安全に保存します。

> **Note**
>
> デフォルトのオプションをオフにすると、Amazon EMRはユーザーに代わってジョブのトラブルシューティングを行うことができません。

EMR Studio からこのオプションをオフにするには、ジョブの送信ページの「追加設定」セクションの「ログを 30 日間保持 AWS する許可」チェックボックスをオフにします。

からこのオプションをオフにするには AWS CLI、ジョブ実行を送信するときに `managedPersistenceMonitoringConfiguration`設定を使用します。

```
{
    "monitoringConfiguration": {
        "managedPersistenceMonitoringConfiguration": {
            "enabled": false
        }
    }
}
```

Amazon S3 バケットを使用した EMR Serverless のログ記録 Amazon S3

ジョブが Amazon S3 にログデータを送信する前に、ジョブランタイムロールのアクセス許可ポリシーに次のアクセス許可を含める必要があります。を口グ記録バケットの名前*amzn-s3-demo-logging-bucket*に置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-logging-bucket/*"
            ]
        }
    ]
}
```

からの口グを保存するように Amazon S3 バケットを設定するには AWS CLI、ジョブの実行を開始するときに `s3MonitoringConfiguration`設定を使用します。これを行うには、 設定`--configuration-overrides`で以下を指定します。

```
{
    "monitoringConfiguration": {
        "s3MonitoringConfiguration": {
            "logUri": "s3://amzn-s3-demo-logging-bucket/logs/"
        }
    }
}
```

再試行が有効になっていないバッチジョブの場合、EMRServerless はログを次のパスに送信します。

```
'/applications/<applicationId>/jobs/<jobId>'
```

EMR Serverless リリース 7.1.0 以降では、ストリーミングジョブとバッチジョブの再試行がサポートされています。再試行を有効にしてジョブを実行すると、EMRServerless はログパスプレフィックスに試行番号を自動的に追加するため、ログをより適切に区別して追跡できます。

```
'/applications/<applicationId>/jobs/<jobId>/attempts/<attemptNumber>/'
```

Amazon での EMR Serverless のログ記録 CloudWatch

EMR Serverless アプリケーションにジョブを送信するときは、アプリケーションログを保存するオプションとして Amazon CloudWatch を選択できます。これにより、 CloudWatch Logs Insights や Live Tail などの CloudWatch ログ分析機能を使用できます。ログ CloudWatch を から他のシステムにストリーミングして、 OpenSearch さらに分析することもできます。

EMR Serverless は、ドライバーログのリアルタイムログ記録を提供します。 CloudWatch ライブテール機能またはテールコマンドを使用して CloudWatch CLI、ログをリアルタイムで表示できます。

デフォルトでは、EMRServerless の CloudWatch ログ記録は無効になっています。有効にするには、 の設定を参照してくださいAWS CLI。

> **Note**
>
> Amazon はログをリアルタイムで CloudWatch 発行するため、ワーカーからより多くのリソースがかかります。ワーカーのキャパシティーを低くすると、ジョブの実行時間への影響が増加する可能性があります。 CloudWatch ログ記録を有効にする場合は、ワーカー容量を増やすことをお勧めします。また、1 秒あたりのトランザクション (TPS) レートが に対して低すぎると、ログ発行がスロットリングされる可能性がありますPutLogEvents。CloudWatch スロットリング設定は、EMRサーバーレスを含むすべてのサービスに対してグローバルです。詳細については、 AWS re:post の CloudWatch 「ログのスロットリングを判断する方法」を参照してください。

でのログ記録に必要なアクセス許可 CloudWatch

ジョブが Amazon にログデータを送信する前に CloudWatch、ジョブランタイムロールのアクセス許可ポリシーに次のアクセス許可を含める必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:AWS #####:111122223333:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
               "logs:PutLogEvents",
               "logs:CreateLogGroup",
               "logs:CreateLogStream",
               "logs:DescribeLogStreams"
            ],
            "Resource": [
               "arn:aws:logs:AWS #####:111122223333:log-group:my-log-group-name:*"
            ]
        }
    ]
}
```

AWS CLI

から EMR Serverless のログを保存する CloudWatch ように Amazon を設定するには AWS CLI、ジョブの実行を開始するときに `cloudWatchLoggingConfiguration`設定を使用します。これを行うには、次の設定オーバーライドを指定します。必要に応じて、ロググループ名、ログストリームプレフィックス名、ログタイプ、および暗号化キー を指定することもできますARN。

オプション値を指定しない場合、 は、デフォルトのログストリーム を使用して`/aws/emr-serverless`、ログをデフォルトのロググループ に CloudWatch 発行します`/applications/`*`applicationId`*`/jobs/`*`jobId`*`/`*`worker-type`*。

EMR Serverless リリース 7.1.0 以降では、ストリーミングジョブとバッチジョブの再試行がサポートされています。ジョブの再試行を有効にした場合、EMRServerless はログパスプレフィックスに試行番号を自動的に追加するため、ログをより適切に区別して追跡できます。

```
'/applications/<applicationId>/jobs/<jobId>/attempts/<attemptNumber>/worker-type'
```

以下は、EMRServerless のデフォルト設定で Amazon CloudWatch ログ記録を有効にするために必要な最小設定を示しています。

```
{
    "monitoringConfiguration": {
        "cloudWatchLoggingConfiguration": {
            "enabled": true
         }
     }
}
```

次の例は、EMRServerless の Amazon CloudWatch ログ記録を有効にするときに指定できる必須およびオプションのすべての設定を示しています。サポートされているlogTypes値もこの例の下に表示されます。

```
{
    "monitoringConfiguration": {
        "cloudWatchLoggingConfiguration": {
            "enabled": true, // Required
            "logGroupName": "Example_logGroup", // Optional
            "logStreamNamePrefix": "Example_logStream", // Optional
            "encryptionKeyArn": "key-arn", // Optional
            "logTypes": {
                "SPARK_DRIVER": ["stdout", "stderr"] //List of values
             }
         }
     }
}
```

デフォルトでは、EMRServerless はドライバーの stdout と stderr ログのみを に発行します CloudWatch。他のログが必要な場合は、 logTypesフィールドを使用してコンテナロールと対応するログタイプを指定できます。

次のリストは、logTypes設定で指定できるサポートされているワーカータイプを示しています。

Spark

- SPARK_DRIVER : ["STDERR", "STDOUT"]
- SPARK_EXECUTOR : ["STDERR", "STDOUT"]

[Hive]

- HIVE_DRIVER : ["STDERR", "STDOUT", "HIVE_LOG", "TEZ_AM"]
- TEZ_TASK : ["STDERR", "STDOUT", "SYSTEM_LOGS"]

ログのローテーション

Amazon EMR Serverless は、Spark アプリケーションログとイベントログをローテーションできます。ログローテーションは、長時間実行されているジョブが、ディスク領域をすべて占有できる大きなログファイルを生成する問題に役立ちます。ログをローテーションすると、ディスクストレージを節約し、ディスクに空き領域がなくなるため、ジョブの失敗を減らすことができます。

ログローテーションはデフォルトで有効になっており、Spark ジョブでのみ使用できます。

Spark イベントログ

> **Note**
>
> Spark イベントログローテーションは、すべての Amazon EMRリリースラベルで使用できます。

EMR Serverless は、単一のイベントログファイルを生成する代わりに、イベントログを定期的にローテーションし、古いイベントログファイルを削除します。ログをローテーションしても、S3 バケットにアップロードされたログには影響しません。

Spark アプリケーションログ

> **Note**
>
> Spark アプリケーションログローテーションは、すべての Amazon EMRリリースラベルで使用できます。

EMR Serverless は、 `stdout` や `stderr` ファイルなどのドライバーやエグゼキュターのSpark アプリケーションログもローテーションします。Spark History Server と Live UI リンクを使用して Studio のログリンクを選択すると、最新のログファイルにアクセスできます。ログファイルは、最新のログの切り捨てられたバージョンです。古いローテーションログを表示するには、ログを保存するときに Amazon S3 の場所を指定する必要があります。詳細については、「Amazon S3 バケットを使用した EMR Serverless のログ記録Amazon S3」を参照してください。

最新のログファイルは、次の場所にあります。EMR Serverless は 15 秒ごとにファイルを更新します。これらのファイルの範囲は 0 MB から 128 MB です。

```
<example-S3-logUri>/applications/<application-id>/jobs/<job-id>/SPARK_DRIVER/stderr.gz
```

次の場所には、古いローテーションされたファイルが含まれています。各ファイルは 128 MB です。

```
<example-S3-logUri>/applications/<application-id>/jobs/<job-id>/SPARK_DRIVER/archived/
stderr_<index>.gz
```

Spark エグゼキューターにも同じ動作が適用されます。この変更は S3 ログ記録にのみ適用されます。ログローテーションでは、Amazon にアップロードされたログストリームに変更はありません CloudWatch。

EMR サーバーレスリリース 7.1.0 以降では、ストリーミングジョブとバッチジョブの再試行がサポートされています。ジョブで再試行を有効にした場合、EMRServerless はそのようなジョブのログパスにプレフィックスを追加し、ログをより適切に追跡して区別できるようにします。このパスには、ローテーションされたすべてのログが含まれます。

```
'/applications/<applicationId>/jobs/<jobId>/attempts/<attemptNumber>/'.
```

ログの暗号化

マネージドストレージによるEMRサーバーレスログの暗号化

マネージドストレージのログを独自のKMSキーで暗号化するには、ジョブ実行を送信するときに`managedPersistenceMonitoringConfiguration`設定を使用します。

```
{
    "monitoringConfiguration": {
        "managedPersistenceMonitoringConfiguration" : {
            "encryptionKeyArn": "key-arn"
        }
    }
}
```

Amazon S3 バケットによる EMR Serverless ログの暗号化 Amazon S3

Amazon S3 バケットのログを独自のKMSキーで暗号化するには、ジョブ実行を送信するときに`s3MonitoringConfiguration`設定を使用します。

```
{
    "monitoringConfiguration": {
        "s3MonitoringConfiguration": {
            "logUri": "s3://amzn-s3-demo-logging-bucket/logs/",
            "encryptionKeyArn": "key-arn"
        }
    }
}
```

Amazon による EMR Serverless ログの暗号化 CloudWatch

Amazon のログを独自のKMSキー CloudWatch で暗号化するには、ジョブ実行を送信するときに `cloudWatchLoggingConfiguration`設定を使用します。

```
{
    "monitoringConfiguration": {
        "cloudWatchLoggingConfiguration": {
            "enabled": true,
            "encryptionKeyArn": "key-arn"
         }
     }
}
```

ログ暗号化に必要なアクセス許可

このセクションの内容

- 必要なユーザーアクセス許可
- Amazon S3 とマネージドストレージの暗号化キーアクセス許可
- Amazon の暗号化キーのアクセス許可 CloudWatch

必要なユーザーアクセス許可

ジョブを送信するか、ログまたはアプリケーションを表示するユーザーは、 キーを使用するアクセス許可を持っているUIs必要があります。アクセス許可は、KMSキーポリシーまたはユーザー、グループ、またはロールのIAMポリシーのいずれかで指定できます。ジョブを送信するユーザーにKMSキーアクセス許可がない場合、EMRServerless はジョブ実行の送信を拒否します。

キーポリシーの例

次のキーポリシーは、 kms:GenerateDataKeyおよび へのアクセス許可を提供しますkms:Decrypt。

```
{
    "Effect": "Allow",
    "Principal":{
       "AWS": "arn:aws:iam::111122223333:user/user-name"
     },
     "Action": [
       "kms:GenerateDataKey",
       "kms:Decrypt"
      ],
     "Resource": "*"
 }
```

IAMポリシーの例

次のIAMポリシーは、 kms:GenerateDataKeyおよび へのアクセス許可を提供しますkms:Decrypt。

```
{
 "Version": "2012-10-17",
  "Statement": {
     "Effect": "Allow",
     "Action": [
           "kms:GenerateDataKey",
           "kms:Decrypt"
      ],
     "Resource": "key-arn"
   }
}
```

Spark または Tez UI を起動するには、ユーザー、グループ、またはロールにemr-serverless:GetDashboardForJobRunAPI、次のように にアクセスするためのアクセス許可を付与する必要があります。

```
{
 "Version": "2012-10-17",
  "Statement": {
     "Effect": "Allow",
     "Action": [
```

```
            "emr-serverless:GetDashboardForJobRun"
        ]
    }
}
```

Amazon S3 とマネージドストレージの暗号化キーアクセス許可

マネージドストレージまたは S3 バケットで独自の暗号化キーを使用してログを暗号化する場合は、次のようにKMSキーアクセス許可を設定する必要があります。

`emr-serverless.amazonaws.com` プリンシパルは、 KMS キーのポリシーで次のアクセス許可を持っている必要があります。

```
{
    "Effect": "Allow",
    "Principal":{
       "Service": "emr-serverless.amazonaws.com"
     },
     "Action": [
       "kms:Decrypt",
       "kms:GenerateDataKey"
      ],
     "Resource": "*"
     "Condition": {
       "StringLike": {
         "aws:SourceArn": "arn:aws:emr-serverless:region:aws-account-id:/
applications/application-id"
       }
     }
 }
```

セキュリティのベストプラクティスとして、KMSキーポリシーに`aws:SourceArn`条件キーを追加することをお勧めします。IAM グローバル条件キーは、EMRServerless がKMSキーをアプリケーション にのみ使用できるようにする`aws:SourceArn`のに役立ちますARN。

ジョブランタイムロールのIAMポリシーには、次のアクセス許可が必要です。

```
{
 "Version": "2012-10-17",
  "Statement": {
     "Effect": "Allow",
```

```
     "Action": [
           "kms:GenerateDataKey",
           "kms:Decrypt"
      ],
     "Resource": "key-arn"
   }
}
```

Amazon の暗号化キーのアクセス許可 CloudWatch

KMS キーをARNロググループに関連付けるには、ジョブランタイムロールに次のIAMポリシーを使用します。

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "logs:AssociateKmsKey"
        ],
        "Resource": [
            "arn:aws:logs:AWS #####:111122223333:log-group:my-log-group-name:*"
        ]
    }
}
```

Amazon にアクセスKMS許可を付与するようにKMSキーポリシーを設定します CloudWatch。

```
{
    "Version": "2012-10-17",
    "Id": "key-default-1",
    "Statement":
     {
        "Effect": "Allow",
        "Principal": {
            "Service": "logs.AWS #####.amazonaws.com"
        },
        "Action": [
            "kms:Decrypt",
            "kms:GenerateDataKey",
        ],
        "Resource": "*",
```

```
        "Condition": {
            "ArnLike": {
                "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:AWS ####
#:111122223333:*"
            }
        }
    }
}
```

Amazon EMR Serverless の Apache Log4j2 プロパティを設定する

このページでは、 で EMR Serverless ジョブのカスタム Apache Log4j 2.x プロパティを設定する方法について説明しますStartJobRun。アプリケーションレベルで Log4j 分類を設定する場合は、「」を参照してくださいEMR Serverless のデフォルトのアプリケーション設定。

Amazon EMR Serverless の Spark Log4j2 プロパティを設定する

Amazon EMRリリース 6.8.0 以降では、Apache Log4j 2.x プロパティをカスタマイズして、きめ細かなログ設定を指定できます。これにより、EMRServerless での Spark ジョブのトラブルシューティングが簡単になります。これらのプロパティを設定するには、 spark-driver-log4j2および spark-executor-log4j2分類を使用します。

トピック

- Spark の Log4j2 分類
- Spark の Log4j2 設定例
- サンプル Spark ジョブの Log4j2
- Spark の Log4j2 に関する考慮事項

Spark の Log4j2 分類

Spark ログ設定をカスタマイズするには、 で次の分類を使用しますapplicationConfiguration。Log4j 2.x プロパティを設定するには、次の を使用しますproperties。

spark-driver-log4j2

この分類は、ドライバーの log4j2.properties ファイル内の値を設定します。

spark-executor-log4j2

この分類は、エグゼキュターの log4j2.properties ファイル内の値を設定します。

Spark の Log4j2 設定例

次の例は、 を使用して Spark ジョブを送信applicationConfigurationし、Spark ドライバーとエグゼキュターの Log4j2 設定をカスタマイズする方法を示しています。

ジョブを送信するときではなく、アプリケーションレベルで Log4j 分類を設定するには、「」を参照してくださいEMR Serverless のデフォルトのアプリケーション設定。

```
aws emr-serverless start-job-run \
    --application-id application-id \
    --execution-role-arn job-role-arn \
    --job-driver '{
        "sparkSubmit": {
            "entryPoint": "/usr/lib/spark/examples/jars/spark-examples.jar",
            "entryPointArguments": ["1"],
            "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi --conf
 spark.executor.cores=4 --conf spark.executor.memory=20g --conf spark.driver.cores=4 --
conf spark.driver.memory=8g --conf spark.executor.instances=1"
        }
    }'
    --configuration-overrides '{
        "applicationConfiguration": [
             {
                "classification": "spark-driver-log4j2",
                "properties": {
                    "rootLogger.level":"error", // will only display Spark error logs
                    "logger.IdentifierForClass.name": "classpath for setting logger",
                    "logger.IdentifierForClass.level": "info"

                }
            },
            {
                "classification": "spark-executor-log4j2",
                "properties": {
                    "rootLogger.level":"error", // will only display Spark error logs
                    "logger.IdentifierForClass.name": "classpath for setting logger",
                    "logger.IdentifierForClass.level": "info"
                }
```

```
            }
        ]
    }'
```

サンプル Spark ジョブの Log4j2

次のコードサンプルは、アプリケーションのカスタム Log4j2 設定を初期化するときに Spark アプリケーションを作成する方法を示しています。

Python

Example - Python で Spark ジョブに Log4j2 を使用する

```
import os
import sys

from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession

app_name = "PySparkApp"
if __name__ == "__main__":
    spark = SparkSession\
        .builder\
        .appName(app_name)\
        .getOrCreate()

    spark.sparkContext._conf.getAll()
    sc = spark.sparkContext
    log4jLogger = sc._jvm.org.apache.log4j
    LOGGER = log4jLogger.LogManager.getLogger(app_name)

    LOGGER.info("pyspark script logger info")
    LOGGER.warn("pyspark script logger warn")
    LOGGER.error("pyspark script logger error")

    // your code here

    spark.stop()
```

Spark ジョブの実行時にドライバーの Log4j2 をカスタマイズするには、次の設定を使用します。

```
{
```

```
   "classification": "spark-driver-log4j2",
      "properties": {
          "rootLogger.level":"error", // only display Spark error logs
          "logger.PySparkApp.level": "info",
          "logger.PySparkApp.name": "PySparkApp"
      }
}
```

Scala

Example - Scala での Spark ジョブでの Log4j2 の使用

```
import org.apache.log4j.Logger
import org.apache.spark.sql.SparkSession

object ExampleClass {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession
    .builder
    .appName(this.getClass.getName)
    .getOrCreate()

    val logger = Logger.getLogger(this.getClass);
    logger.info("script logging info logs")
    logger.warn("script logging warn logs")
    logger.error("script logging error logs")

// your code here
    spark.stop()
  }
}
```

Spark ジョブの実行時にドライバーの Log4j2 をカスタマイズするには、次の設定を使用します。

```
{
   "classification": "spark-driver-log4j2",
      "properties": {
          "rootLogger.level":"error", // only display Spark error logs
          "logger.ExampleClass.level": "info",
          "logger.ExampleClass.name": "ExampleClass"
      }
}
```

Spark の Log4j2 に関する考慮事項

Spark プロセスでは、次の Log4j2.x プロパティは設定できません。

- `rootLogger.appenderRef.stdout.ref`
- `appender.console.type`
- `appender.console.name`
- `appender.console.target`
- `appender.console.layout.type`
- `appender.console.layout.pattern`

設定できる Log4j2.x プロパティの詳細については、 の `log4j2.properties.template` ファイルを参照してください GitHub。

EMR サーバーレスのモニタリング

このセクションでは、Amazon EMR Serverless アプリケーションとジョブをモニタリングする方法について説明します。

トピック

- EMR Serverless アプリケーションとジョブのモニタリング
- Amazon Managed Service for Prometheus で Spark メトリクスをモニタリングする
- EMR サーバーレス使用状況メトリクス

EMR Serverless アプリケーションとジョブのモニタリング

EMR Serverless の Amazon CloudWatch メトリクスを使用すると、1 分間の CloudWatch メトリクスとアクセス CloudWatch ダッシュボードを受信して、EMRServerless アプリケーションのオペレーションとパフォーマンスを表示できます near-real-time。

EMR Serverless は 1 分 CloudWatch ごとにメトリクスを に送信します。EMR サーバーレスは、これらのメトリクスをアプリケーションレベル、ジョブ、ワーカータイプ、および capacity-allocation-typeレベルで出力します。

開始するには、EMRServerless EMR GitHub リポジトリで提供されている Serverless CloudWatch ダッシュボードテンプレートを使用してデプロイします。

> **Note**
>
> EMR サーバーレスインタラクティブワークロードでは、アプリケーションレベルのモニタリングのみが有効化され、新しいワーカータイプのディメンション があります`Spark_Kernel`。インタラクティブワークロードをモニタリングおよびデバッグするには、EMRStudio Workspace 内からログと Apache Spark UI を表示できます。

次の表は、`AWS/EMRServerless`名前空間内で使用可能なEMRサーバーレスディメンションを示しています。

EMR Serverless メトリクスのディメンション

ディメンション	説明
`ApplicationId`	EMR Serverless アプリケーションのすべてのメトリクスをフィルタリングします。
`JobId`	EMR Serverless ジョブ実行のすべてのメトリクスをフィルタリングします。
`WorkerType`	特定のワーカータイプのすべてのメトリクスをフィルタリングします。例えば、Spark ジョブの `SPARK_DRIVER` と `SPARK_EXECUTORS` をフィルタリングできます。
`CapacityAllocationType`	特定のキャパシティ割り当てタイプのすべてのメトリクスをフィルタリングします。例えば、 `PreInitCapacity` をフィルタリングして、事前に初期化された容量と、その他すべての容

ディメンション	説明	
	量OnDemandCapacity をフィルタリングできます。	

アプリケーションレベルのモニタリング

Amazon CloudWatch メトリクスを使用してEMR、サーバーレスアプリケーションレベルでキャパシティの使用状況をモニタリングできます。 CloudWatch ダッシュボードでアプリケーション容量の使用状況をモニタリングするように 1 つのビューを設定することもできます。

EMR サーバーレスアプリケーションメトリクス

メトリクス	説明	プライマリディメンション	セカンダリディメンション
CPUAllocated	vCPUs 割り当てられた の合計数。	ApplicationId	ApplicationId , WorkerType , CapacityAllocationType
IdleWorkerCount	アイドル状態のワーカーの合計数。	ApplicationId	ApplicationId , WorkerType , CapacityAllocationType
MaxCPUAllowed	アプリケーションでCPU許可される最大数。	ApplicationId	該当なし
MaxMemoryAllowed	アプリケーションで許可される GB の最大メモリ。	ApplicationId	該当なし
MaxStorageAllowed	アプリケーションで許可される GB の最大ストレージ。	ApplicationId	該当なし

メトリクス	説明	プライマリディメンション	セカンダリディメンション
MemoryAllocated	割り当てられた GB の合計メモリ。	ApplicationId	ApplicationId , WorkerType , CapacityAllocationType
PendingCreationWorkerCount	作成保留中のワーカーの合計数。	ApplicationId	ApplicationId , WorkerType , CapacityAllocationType
RunningWorkerCount	アプリケーションで使用されているワーカーの合計数。	ApplicationId	ApplicationId , WorkerType , CapacityAllocationType
StorageAllocated	割り当てられた GB の合計ディスクストレージ。	ApplicationId	ApplicationId , WorkerType , CapacityAllocationType
TotalWorkerCount	使用可能なワーカーの合計数。	ApplicationId	ApplicationId , WorkerType , CapacityAllocationType

ジョブレベルのモニタリング

Amazon EMR Serverless は、1 分 Amazon CloudWatch ごとに次のジョブレベルのメトリクスを に送信します。集計ジョブ実行のメトリクス値をジョブ実行状態別に表示できます。各メトリクスの単位はカウント です。

EMR サーバーレスジョブレベルのメトリクス

メトリクス	説明	プライマリディメンション
SubmittedJobs	送信済み状態のジョブの数。	ApplicationId
PendingJobs	保留中状態のジョブの数。	ApplicationId
ScheduledJobs	スケジュールされた状態のジョブの数。	ApplicationId
RunningJobs	実行中状態のジョブの数。	ApplicationId
SuccessJobs	成功状態のジョブの数。	ApplicationId
FailedJobs	失敗状態のジョブの数。	ApplicationId
CancellingJobs	キャンセル状態のジョブの数。	ApplicationId
CancelledJobs	キャンセルされた状態のジョブの数。	ApplicationId

エンジン固有のアプリケーション を使用して、実行中のサーバーレスジョブと完了したEMRサーバーレスジョブの両方のエンジン固有のメトリクスをモニタリングできますUIs。実行中のジョブの UI を表示すると、リアルタイムの更新を含むライブアプリケーション UI が表示されます。完了したジョブの UI を表示すると、永続アプリケーション UI が表示されます。

ジョブの実行

実行中の EMR Serverless ジョブでは、エンジン固有のメトリクスを提供するリアルタイムインターフェイスを表示できます。Apache Spark UI または Hive Tez UI を使用して、ジョブをモニタリングおよびデバッグできます。これらの にアクセスするにはUIs、EMRStudio コンソールを使用するか、 で安全なURLエンドポイントをリクエストします AWS Command Line Interface。

完了したジョブ

完了した EMR Serverless ジョブでは、Spark History Server または Persistent Hive Tez UI を使用して、Spark または Hive ジョブ実行のジョブの詳細、ステージ、タスク、メトリクスを表示できま

す。これらの にアクセスするにはUIs、EMRStudio コンソールを使用するか、 で安全なURLエンドポイントをリクエストします AWS Command Line Interface。

ジョブワーカーレベルのモニタリング

Amazon EMR Serverless は、AWS/EMRServerless名前空間およびメトリクスグループで利用可能な以下のジョブワーカーレベルのJob Worker Metricsメトリクスを Amazon に送信します CloudWatch。EMR Serverless は、ジョブレベル、ワーカータイプ、およびレベルでのジョブ実行中に個々のワーカーからデータポイントを収集します capacity-allocation-type。をディメンションApplicationIdとして使用して、同じアプリケーションに属する複数のジョブをモニタリングできます。

EMR サーバーレスジョブワーカーレベルのメトリクス

メトリクス	説明	単位	プライマリディメンション	セカンダリディメンション
WorkerCpuAllocated	ジョブ実行のワーカーに割り当てられた vCPU コアの合計数。	なし	JobId	ApplicationId 、WorkerType 、および CapacityAllocationType
WorkerCpuUsed	ジョブ実行でワーカーが使用する vCPU コアの合計数。	なし	JobId	ApplicationId 、WorkerType 、および CapacityAllocationType
WorkerMemoryAllocated	ジョブ実行のワーカーに割り当てられた GB の合計メモリ。	ギガバイト (GB)	JobId	ApplicationId 、WorkerType 、および CapacityAllocationType

メトリクス	説明	単位	プライマリディメンション	セカンダリディメンション
WorkerMemoryUsed	ジョブ実行でワーカーが使用する GB の合計メモリ。	ギガバイト (GB)	JobId	ApplicationId 、WorkerType 、および CapacityAllocationType
WorkerEphemeralStorageAllocated	ジョブ実行のワーカーに割り当てられたエフェメラルストレージのバイト数。	ギガバイト (GB)	JobId	ApplicationId 、WorkerType 、および CapacityAllocationType
WorkerEphemeralStorageUsed	ジョブ実行でワーカーが使用するエフェメラルストレージのバイト数。	ギガバイト (GB)	JobId	ApplicationId 、WorkerType 、および CapacityAllocationType
WorkerStorageReadBytes	ジョブ実行でワーカーがストレージから読み取ったバイト数。	バイト	JobId	ApplicationId 、WorkerType 、および CapacityAllocationType
WorkerStorageWriteBytes	ジョブ実行のワーカーからストレージに書き込まれたバイト数。	バイト	JobId	ApplicationId 、WorkerType 、および CapacityAllocationType

以下のステップでは、さまざまなタイプのメトリクスを表示する方法について説明します。

Console

コンソールを使用してアプリケーション UI にアクセスするには

1. コンソール から開始するの手順を使用して、EMRStudio の EMR Serverless アプリケーションに移動します。
2. 実行中のジョブのエンジン固有のアプリケーションUIsとログを表示するには：
   a. RUNNING ステータスのジョブを選択します。
   b. アプリケーションの詳細ページでジョブを選択するか、ジョブの詳細ページに移動します。
   c. Display UI ドロップダウンメニューで、Spark UI または Hive Tez UI のいずれかを選択して、ジョブタイプのアプリケーション UI に移動します。
   d. Spark エンジンログを表示するには、Spark UI の Executors タブに移動し、ドライバーの Logs リンクを選択します。Hive エンジンログを表示するには、Hive Tez UI DAG で適切な のログリンクを選択します。
3. 完了したジョブのエンジン固有のアプリケーションUIsとログを表示するには：
   a. SUCCESS ステータスのジョブを選択します。
   b. アプリケーションのアプリケーションの詳細ページでジョブを選択するか、ジョブのジョブの詳細ページに移動します。
   c. Display UI ドロップダウンメニューで、Spark History Server または Persistent Hive Tez UI のいずれかを選択して、ジョブタイプのアプリケーション UI に移動します。
   d. Spark エンジンログを表示するには、Spark UI の Executors タブに移動し、ドライバーの Logs リンクを選択します。Hive エンジンログを表示するには、Hive Tez UI DAG で適切なログリンクを選択します。

AWS CLI

を使用してアプリケーション UI にアクセスするには AWS CLI

- 実行中のジョブと完了したジョブの両方でアプリケーション UI にアクセスするために使用できる を生成するURLには、 GetDashboardForJobRun を呼び出しますAPI。

```
aws emr-serverless get-dashboard-for-job-run /
```

```
--application-id <application-id> /
--job-run-id <job-id>
```

生成URLした は 1 時間有効です。

Amazon Managed Service for Prometheus で Spark メトリクスをモニタリングする

Amazon EMRリリース 7.1.0 以降では、EMRServerless を Amazon Managed Service for Prometheus と統合してEMR、Serverless ジョブとアプリケーションの Apache Spark メトリクスを収集できます。この統合は、 AWS コンソール、EMRサーバーレス 、または を使用してジョブを送信するかAPI、アプリケーションを作成するときに使用できます AWS CLI。

前提条件

Spark メトリクスを Amazon Managed Service for Prometheus に配信する前に、次の前提条件を満たす必要があります。

- Amazon Managed Service for Prometheus ワークスペースを作成します。このワークスペースは、取り込みエンドポイントとして機能します。Endpoint - リモート書き込み URLURLに表示される を書き留めます。EMR Serverless アプリケーションを作成するURLときは、 を指定する必要があります。
- モニタリング目的で Amazon Managed Service for Prometheus にジョブへのアクセスを許可するには、ジョブ実行ロールに次のポリシーを追加します。

```
{
    "Sid": "AccessToPrometheus",
    "Effect": "Allow",
    "Action": ["aps:RemoteWrite"],
    "Resource": "arn:aws:aps:<AWS_REGION>:<AWS_ACCOUNT_ID>:workspace/<WORKSPACE_ID>"
}
```

セットアップ

AWS コンソールを使用して Amazon Managed Service for Prometheus と統合されたアプリケーションを作成するには

1. 「Amazon EMR Serverless の開始方法」を参照してアプリケーションを作成します。

2. アプリケーションの作成中に、カスタム設定を使用する を選択し、設定するフィールドに情報を指定してアプリケーションを設定します。
3. アプリケーションログとメトリクス で、Amazon Managed Service for Prometheus にエンジンメトリクスを配信を選択し、リモート書き込み を指定しますURL。
4. 必要な他の設定を指定し、アプリケーションの作成と起動 を選択します。

AWS CLI または EMR Serverless を使用する API

また、 AWS CLI または EMR Serverless を使用してAPI、 `create-application` または `start-job-run` コマンドの実行時に EMR Serverless アプリケーションを Amazon Managed Service for Prometheus と統合することもできます。

create-application

```
aws emr-serverless create-application \
--release-label emr-7.1.0 \
--type "SPARK" \
--monitoring-configuration '{
    "prometheusMonitoringConfiguration": {
        "remoteWriteUrl": "https://aps-workspaces.<AWS_REGION>.amazonaws.com/
workspaces/<WORKSPACE_ID>/api/v1/remote_write"
    }
}'
```

start-job-run

```
aws emr-serverless start-job-run \
--application-id <APPPLICATION_ID> \
--execution-role-arn <JOB_EXECUTION_ROLE> \
--job-driver '{
    "sparkSubmit": {
        "entryPoint": "local:///usr/lib/spark/examples/src/main/python/pi.py",
        "entryPointArguments": ["10000"],
        "sparkSubmitParameters": "--conf spark.dynamicAllocation.maxExecutors=10"
    }
}' \
--configuration-overrides '{
     "monitoringConfiguration": {
        "prometheusMonitoringConfiguration": {
            "remoteWriteUrl": "https://aps-workspaces.<AWS_REGION>.amazonaws.com/
workspaces/<WORKSPACE_ID>/api/v1/remote_write"
```

```
        }
    }
}'
```

コマンドprometheusMonitoringConfigurationに を含めると、EMRServerless は Spark メトリクスを収集し、Amazon Managed Service for Prometheus のremoteWriteUrlエンドポイントに書き込むエージェントを使用して Spark ジョブを実行する必要があります。その後、Amazon Managed Service for Prometheus の Spark メトリクスを使用して、視覚化、アラート、分析を行うことができます。

高度な設定プロパティ

EMR Serverless は、 という名前の Spark 内のコンポーネントを使用して Spark メトリクスPrometheusServletを収集し、パフォーマンスデータを Amazon Managed Service for Prometheus と互換性のあるデータに変換します。デフォルトでは、EMRServerless は Spark にデフォルト値を設定し、 を使用してジョブを送信するときにドライバーとエグゼキューターのメトリクスを解析しますPrometheusMonitoringConfiguration。

次の表は、Amazon Managed Service for Prometheus にメトリクスを送信する Spark ジョブを送信するときに設定できるすべてのプロパティを示しています。

Spark プロパティ	デフォルト値	説明
spark.metrics.conf.*.sink.prometheusServlet.class	org.apache.spark.metrics.sink。PrometheusServlet	Spark が Amazon Managed Service for Prometheus にメトリクスを送信するために使用するクラス。デフォルトの動作を上書きするには、独自のカスタムクラスを指定します。
spark.metrics.conf.*.source.jvm.class	org.apache.spark.metrics.source。JvmSource	Spark が基盤となる Java 仮想マシンから重要なメトリクスを収集して送信するために使用するクラス。JVM メトリクスの収集を停止するには、このプロパティを などの空の文字列に設定して無効にしま

Spark プロパティ	デフォルト値	説明
		す""。デフォルトの動作を上書きするには、独自のカスタムクラスを指定します。
`spark.metrics.conf.driver.sink.prometheusServlet.path`	/metrics/prometheus	Amazon Managed Service for Prometheus URLがドライバーからメトリクスを収集するために使用する特徴。デフォルトの動作を上書きするには、独自のパスを指定します。ドライバーメトリクスの収集を停止するには、このプロパティを などの空の文字列に設定して無効にします""。
`spark.metrics.conf.executor.sink.prometheusServlet.path`	/metrics/executor/prometheus	Amazon Managed Service for Prometheus URLがエグゼキューターからメトリクスを収集するために使用する個別の 。デフォルトの動作を上書きするには、独自のパスを指定します。エグゼキューターメトリクスの収集を停止するには、このプロパティを などの空の文字列に設定して無効にします""。

Spark メトリクスの詳細については、「Apache Spark メトリクス」を参照してください。

考慮事項と制約事項

Amazon Managed Service for Prometheus を使用して EMR Serverless からメトリクスを収集する場合は、次の考慮事項と制限事項を考慮してください。

- EMR Serverless での Amazon Managed Service for Prometheus の使用のサポートは、AWS リージョン Amazon Managed Service for Prometheus が一般的に利用可能な でのみ利用できます。

- Amazon Managed Service for Prometheus でエージェントを実行して Spark メトリクスを収集するには、ワーカーからのリソースがさらに必要です。1 人の vCPU ワーカーなど、より小さいワーカーサイズを選択すると、ジョブの実行時間が長くなる可能性があります。
- EMR Serverless での Amazon Managed Service for Prometheus の使用のサポートは、Amazon EMRリリース 7.1.0 以降でのみ利用できます。

EMR サーバーレス使用状況メトリクス

Amazon CloudWatch 使用状況メトリクスを使用して、アカウントが使用するリソースを可視化できます。これらのメトリクスを使用して、 CloudWatch グラフとダッシュボードのサービス使用状況を視覚化します。

EMR サーバーレス使用状況メトリクスは、Service Quotas に対応します。使用量がサービスクォータに近づいたときに警告するアラームを設定することもできます。詳細については、Service Quotas ユーザーガイド」の「Service Quotas」と「Amazon CloudWatch アラーム」を参照してください。Service Quotas

EMR Serverless サービスクォータの詳細については、「」を参照してくださいのエンドポイントとクォータ EMR Serverless。

EMR Serverless のサービスクォータ使用量メトリクス

EMR Serverless は、以下のサービスクォータ使用状況メトリクスをAWS/Usage名前空間に発行します。

メトリクス	説明
ResourceCount	アカウントで実行されている指定されたリソースの合計数。リソースは、メトリクスに関連付けられているディメンションによって定義されます。

EMR Serverless サービスクォータ使用量メトリクスのディメンション

次のディメンションを使用して、EMRServerless が公開する使用状況メトリクスを絞り込むことができます。

ディメンション	値	説明
`Service`	EMR サーバーレス	リソース AWS のサービス を含む の名前。
`Type`	リソース	EMR Serverless がレポートするエンティティのタイプ。
`Resource`	vCPU	EMR Serverless が追跡しているリソースのタイプ。
`Class`	なし	EMR Serverless が追跡しているリソースのクラス。

によるEMRサーバーレスの自動化 Amazon EventBridge

Amazon EventBridge を使用して を自動化 AWS のサービス し、アプリケーションの可用性の問題やリソースの変更などのシステムイベントに自動的に対応できます。 は、 AWS リソースの変更を記述するシステムイベントのほぼリアルタイムのストリーム EventBridge を提供します。簡単なルールを記述して、注目するイベントと、イベントがルールに一致した場合に自動的に実行するアクションを指定できます。を使用すると EventBridge、自動的に次のことを実行できます。

- AWS Lambda 関数を呼び出す
- イベントを Amazon Kinesis Data Streams にリレーする
- AWS Step Functions ステートマシンをアクティブ化する
- Amazon SNSトピックまたは Amazon SQSキューに通知する

例えば、 を EMR Serverless EventBridge で使用すると、ETLジョブが成功したときに関数をアクティブ化 AWS Lambda したり、ETLジョブが失敗したときに Amazon SNSトピックに通知したりできます。

EMR サーバーレスは、次の 3 種類のイベントを出力します。

- アプリケーション状態変更イベント – アプリケーションの状態変更をすべて実行するイベント。アプリケーションの状態の詳細については、「」を参照してくださいアプリケーションの状態。

- ジョブ実行状態変更イベント – ジョブ実行のすべての状態変更を出力するイベント。詳細については、「ジョブ実行状態」を参照してください。
- ジョブ実行の再試行イベント – Amazon EMR Serverless リリース 7.1.0 以降から実行されるジョブのすべての再試行を出力するイベント。

EMR サーバーレス EventBridgeイベントのサンプル

EMR Serverless によってレポートされるイベントには、次の例のように`source`、 に の値が`aws.emr-serverless`割り当てられます。

アプリケーション状態変更イベント

次のイベント例は、 `CREATING`状態のアプリケーションを示しています。

```
{
    "version": "0",
    "id": "9fd3cf79-1ff1-b633-4dd9-34508dc1e660",
    "detail-type": "EMR Serverless Application State Change",
    "source": "aws.emr-serverless",
    "account": "123456789012",
    "time": "2022-05-31T21:16:31Z",
    "region": "us-east-1",
    "resources": [],
    "detail": {
        "applicationId": "00f1cbsc6anuij25",
        "applicationName": "3965ad00-8fba-4932-a6c8-ded32786fd42",
        "arn": "arn:aws:emr-serverless:us-east-1:111122223333:/
applications/00f1cbsc6anuij25",
        "releaseLabel": "emr-6.6.0",
        "state": "CREATING",
        "type": "HIVE",
        "createdAt": "2022-05-31T21:16:31.547953Z",
        "updatedAt": "2022-05-31T21:16:31.547970Z",
        "autoStopConfig": {
            "enabled": true,
            "idleTimeout": 15
        },
        "autoStartConfig": {
            "enabled": true
        }
    }
```

```
}
```

ジョブ実行状態変更イベント

次のイベント例は、 SCHEDULED状態から RUNNING状態に移行するジョブ実行を示しています。

```
{
    "version": "0",
    "id": "00df3ec6-5da1-36e6-ab71-20f0de68f8a0",
    "detail-type": "EMR Serverless Job Run State Change",
    "source": "aws.emr-serverless",
    "account": "123456789012",
    "time": "2022-05-31T21:07:42Z",
    "region": "us-east-1",
    "resources": [],
    "detail": {
        "jobRunId": "00f1cbn5g4bb0c01",
        "applicationId": "00f1982r1uukb925",
        "arn": "arn:aws:emr-serverless:us-east-1:123456789012:/
applications/00f1982r1uukb925/jobruns/00f1cbn5g4bb0c01",
        "releaseLabel": "emr-6.6.0",
        "state": "RUNNING",
        "previousState": "SCHEDULED",
        "createdBy": "arn:aws:sts::123456789012:assumed-role/
TestRole-402dcef3ad14993c15d28263f64381e4cda34775/6622b6233b6d42f59c25dd2637346242",
        "updatedAt": "2022-05-31T21:07:42.299487Z",
        "createdAt": "2022-05-31T21:07:25.325900Z"
    }
}
```

ジョブ実行再試行イベント

以下は、ジョブ実行の再試行イベントの例です。

```
{
    "version": "0",
    "id": "00df3ec6-5da1-36e6-ab71-20f0de68f8a0",
    "detail-type": "EMR Serverless Job Run Retry",
    "source": "aws.emr-serverless",
    "account": "123456789012",
    "time": "2022-05-31T21:07:42Z",
    "region": "us-east-1",
    "resources": [],
```

```
    "detail": {
        "jobRunId": "00f1cbn5g4bb0c01",
        "applicationId": "00f1982r1uukb925",
        "arn": "arn:aws:emr-serverless:us-east-1:123456789012:/
applications/00f1982r1uukb925/jobruns/00f1cbn5g4bb0c01",
        "releaseLabel": "emr-6.6.0",
        "createdBy": "arn:aws:sts::123456789012:assumed-role/
TestRole-402dcef3ad14993c15d28263f64381e4cda34775/6622b6233b6d42f59c25dd2637346242",
        "updatedAt": "2022-05-31T21:07:42.299487Z",
        "createdAt": "2022-05-31T21:07:25.325900Z",
        //Attempt Details
        "previousAttempt": 1,
        "previousAttemptState": "FAILED",
        "previousAttemptCreatedAt": "2022-05-31T21:07:25.325900Z",
        "previousAttemptEndedAt": "2022-05-31T21:07:30.325900Z",
        "newAttempt": 2,
        "newAttemptCreatedAt": "2022-05-31T21:07:30.325900Z"
    }
}
```

リソースのタグ付け

EMR サーバーレスリソースの管理に役立つタグを使用して、各リソースに独自のメタデータを割り当てることができます。このセクションでは、タグ関数の概要とタグの作成方法について説明します。

トピック

タグとは

タグは、 に割り当てるラベルです。 AWS リソース。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。タグを使用すると、 を分類できます。 AWS 目的、所有者、環境などの属性別の リソース。同じ型のリソースが多い場合に、割り当てたタグに基づいて特定のリソースをすばやく識別できます。例えば、Amazon EMR Serverless アプリケーションの一連のタグを定義して、各アプリケーションの所有者とスタックレベルを追跡しやすくすることができます。リソースタイプごとに一貫した一連のタグキーを考案することをお勧めします。

タグは自動的にはリソースに割り当てられません。リソースにタグを追加した後、タグの値を変更したり、リソースからタグを削除したりできます。タグには Amazon EMR Serverless に対する意味論的な意味はなく、文字列として厳密に解釈されます。そのリソースの既存のタグと同じキーを持つタグを追加した場合、古い値は新しい値によって上書きされます。

を使用する場合IAM、 内のユーザーを制御できます。 AWS アカウントには、タグを管理するアクセス許可があります。タグベースのアクセスコントロールポリシーの例については、「タグベースのアクセスコントロールのポリシー」を参照してください。

リソースのタグ付け

新規または既存のアプリケーションとジョブ実行にタグを付けることができます。Amazon EMR Serverless を使用している場合はAPI、 AWS CLI、または AWS SDKでは、関連するAPIアクションの `tags`パラメータを使用して、新しいリソースにタグを適用できます。`TagResource` API アクションを使用して、既存のリソースにタグを適用できます。

リソースの作成時に、リソースのタグを指定するためのいくつかのリソース作成アクションを使用できます。その場合、リソースの作成中にタグを適用できないときは、リソースの作成は失敗します。これにより、作成時にタグ付けしたリソースが、指定したタグで作成されているか、まったく作成されていないかが確認できます。作成時にリソースにタグ付けする場合、リソースの作成後にカスタムタグ付けスクリプトを実行する必要はありません。

次の表に、タグ付けできる Amazon EMR Serverless リソースを示します。

リソース	タグをサポート	タグの伝播をサポート	作成時のタグ付けをサポート (Amazon EMR Serverless API、AWS CLIおよび AWS SDK)	API 作成用 (作成時にタグを追加可能)
アプリケーション	あり	いいえ。アプリケーションに関連付けられたタグは、そのアプリケーションに送信されたジョブ実行には伝達されません。	あり	`CreateApplication`
ジョブ実行	あり	いいえ	あり	`StartJobRun`

タグ付けの制限

タグには、次の基本的な制限が適用されます。

- 各リソースには、ユーザーが作成したタグを最大 50 個含めることができます。
- タグキーは、リソースごとにそれぞれ一意である必要があります。また、各タグキーに設定できる値は 1 つのみです。
- キーの最大長は UTF-8 で 128 Unicode 文字です。
- 値の最大長は UTF-8 で 256 Unicode 文字です。
- 使用できる文字は、UTF-8 で表される文字、数字、スペース、および _ . : / = + - @ です。

- タグキーを空の文字列にすることはできません。タグの値を空の文字列にすることはできますが、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。
- キーまたは値のプレフィックスなど、 のAWS:大文字または小文字の組み合わせは使用しないでください。これらは、 専用に予約されています。 AWS を使用します。

を使用したタグの操作 AWS CLI と Amazon EMR Serverless API

以下を使用する AWS CLI リソースのタグを追加、更新、一覧表示、削除する コマンドまたは Amazon EMR Serverless APIオペレーション。

リソース	タグをサポート	タグの伝播をサポート
1 つ以上のタグを追加、または上書きします	`tag-resource`	`TagResource`
リソースのタグの一覧表示	`list-tags-for-resource`	`ListTagsForResource`
1 つ以上のタグを削除します	`untag-resource`	`UntagResource`

次の例は、 を使用してリソースにタグを付ける、またはタグを解除する方法を示しています。 AWS CLI.

既存のアプリケーションのタグ付け

次のコマンドは、既存のアプリケーションにタグを付けます。

```
aws emr-serverless tag-resource --resource-arn resource_ARN --tags team=devs
```

既存のアプリケーションのタグを解除する

次のコマンドは、既存のアプリケーションからタグを削除します。

```
aws emr-serverless untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

リソースのタグを一覧表示する

次のコマンドは、既存のリソースに関連付けられているタグのリストを取得します。

```
aws emr-serverless list-tags-for-resource --resource-arn resource_ARN
```

EMR Serverless のチュートリアル

このセクションでは、EMRサーバーレスアプリケーションを使用する際の一般的なユースケースについて説明します。

トピック

Amazon EMR Serverless での Java 17 の使用

Amazon EMRリリース 6.11.0 以降では、Java 仮想マシン () に Java 17 ランタイムを使用するように EMR Serverless Spark ジョブを設定できますJVM。次のいずれかの方法を使用して、Java 17 で Spark を設定します。

`JAVA_HOME`

EMR Serverless 6.11.0 以降の JVM 設定を上書きするには、 `JAVA_HOME`設定を `spark.emr-serverless.driverEnv`および `spark.executorEnv`環境分類に指定します。

x86_64

必要なプロパティを設定して、Spark ドライバーとエグゼキューター`JAVA_HOME`の設定として Java 17 を指定します。

```
--conf spark.emr-serverless.driverEnv.JAVA_HOME=/usr/lib/jvm/java-17-amazon-
corretto.x86_64/
--conf spark.executorEnv.JAVA_HOME=/usr/lib/jvm/java-17-amazon-corretto.x86_64/
```

arm_64

必要なプロパティを設定して、Spark ドライバーとエグゼキューターJAVA_HOMEの設定として Java 17 を指定します。

```
--conf spark.emr-serverless.driverEnv.JAVA_HOME=/usr/lib/jvm/java-17-amazon-
corretto.aarch64/
--conf spark.executorEnv.JAVA_HOME=/usr/lib/jvm/java-17-amazon-corretto.aarch64/
```

spark-defaults

または、 spark-defaults 分類で Java 17 を指定して、EMRServerless 6.11.0 以降のJVM設定を上書きすることもできます。

x86_64

spark-defaults 分類で Java 17 を指定します。

```
{
"applicationConfiguration": [
         {
            "classification": "spark-defaults",
            "properties": {
               "spark.emr-serverless.driverEnv.JAVA_HOME" : "/usr/lib/jvm/java-17-
amazon-corretto.x86_64/",
               "spark.executorEnv.JAVA_HOME": "/usr/lib/jvm/java-17-amazon-
corretto.x86_64/"
            }
         }
      ]
}
```

arm_64

spark-defaults 分類で Java 17 を指定します。

```
{
```

```
"applicationConfiguration": [
         {
            "classification": "spark-defaults",
            "properties": {
               "spark.emr-serverless.driverEnv.JAVA_HOME" : "/usr/lib/jvm/java-17-
amazon-corretto.aarch64/",
               "spark.executorEnv.JAVA_HOME": "/usr/lib/jvm/java-17-amazon-
corretto.aarch64/"
            }
         }
      ]
}
```

Serverless での Apache Hudi EMR の使用

EMR サーバーレスアプリケーションで Apache Hudi を使用するには

1. 対応する Spark ジョブ実行に必要な Spark プロパティを設定します。

   ```
   spark.jars=/usr/lib/hudi/hudi-spark-bundle.jar
   spark.serializer=org.apache.spark.serializer.KryoSerializer
   ```

2. Hudi テーブルを設定済みカタログに同期するには、 AWS Glue Data Catalog をメタストアとして使用するか、外部メタストアを設定します。EMR サーバーレスは、Hudi ワークロードの Hive テーブルの同期モードhmsとして をサポートします。EMR Serverless は、このプロパティをデフォルトとしてアクティブ化します。メタストアの設定方法の詳細については、「」を参照してくださいメタストア設定。

   > **Important**
   >
   > EMR サーバーレスは、Hudi ワークロードを処理するための Hive テーブルの同期モードオプションJDBCとして HIVEQLまたは をサポートしていません。詳細については、「同期モード」を参照してください。

   を使用する場合 AWS Glue Data Catalog をメタストアとして指定すると、Hudi ジョブに次の設定プロパティを指定できます。

   ```
   --conf spark.jars=/usr/lib/hudi/hudi-spark-bundle.jar,
   ```

```
--conf spark.serializer=org.apache.spark.serializer.KryoSerializer,
--conf
 spark.hadoop.hive.metastore.client.factory.class=com.amazonaws.glue.catalog.metastore.AWSG
```

Amazon の Apache Hudi リリースの詳細についてはEMR、「Hudi リリース履歴」を参照してください。

Serverless での Apache Iceberg EMR の使用

EMR サーバーレスアプリケーションで Apache Iceberg を使用するには

1. 対応する Spark ジョブ実行に必要な Spark プロパティを設定します。

   ```
   spark.jars=/usr/share/aws/iceberg/lib/iceberg-spark3-runtime.jar
   ```

2. のいずれかを指定する AWS Glue Data Catalog をメタストアとして使用するか、外部メタストアを設定します。メタストアの設定の詳細については、「」を参照してくださいメタストア設定。

   Iceberg に使用するメタストアプロパティを設定します。例えば、 AWS Glue データカタログで、アプリケーション設定で次のプロパティを設定します。

   ```
   spark.sql.catalog.dev.warehouse=s3://DOC-EXAMPLE-BUCKET/EXAMPLE-PREFIX/
   spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions
   spark.sql.catalog.dev=org.apache.iceberg.spark.SparkCatalog
   spark.sql.catalog.dev.catalog-impl=org.apache.iceberg.aws.glue.GlueCatalog
   spark.hadoop.hive.metastore.client.factory.class=com.amazonaws.glue.catalog.metastore.AWSGl
   ```

   を使用する場合 AWS Glue Data Catalog をメタストアとして、Iceberg ジョブに次の設定プロパティを指定できます。

   ```
   --conf spark.jars=/usr/share/aws/iceberg/lib/iceberg-spark3-runtime.jar,
   --conf
    spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions,
   --conf spark.sql.catalog.dev=org.apache.iceberg.spark.SparkCatalog,
   --conf spark.sql.catalog.dev.catalog-impl=org.apache.iceberg.aws.glue.GlueCatalog,
   --conf spark.sql.catalog.dev.warehouse=s3://DOC-EXAMPLE-BUCKET/EXAMPLE-PREFIX/
   --conf
    spark.hadoop.hive.metastore.client.factory.class=com.amazonaws.glue.catalog.metastore.AWSG
   ```

Amazon の Apache Iceberg リリースの詳細についてはEMR、「Iceberg リリース履歴」を参照してください。

Serverless での Python EMR ライブラリの使用

Amazon EMR Serverless アプリケーションで PySpark ジョブを実行すると、さまざまな Python ライブラリを依存関係としてパッケージ化できます。これを行うには、ネイティブ Python 機能を使用するか、仮想環境を構築するか、Python ライブラリを使用するように PySpark ジョブを直接設定します。このページでは、各アプローチについて説明します。

ネイティブ Python 機能の使用

次の設定を行うと、 PySpark を使用して Python ファイル (`.py`) 、圧縮された Python パッケージ (`.zip`) 、および Egg ファイル (`.egg`) を Spark エグゼキューターにアップロードできます。

```
--conf spark.submit.pyFiles=s3://DOC-EXAMPLE-BUCKET/EXAMPLE-PREFIX/<.py|.egg|.zip file>
```

PySpark ジョブに Python 仮想環境を使用する方法の詳細については、 PySpark 「ネイティブ機能の使用」を参照してください。

Python 仮想環境の構築

PySpark ジョブに複数の Python ライブラリをパッケージ化するには、分離された Python 仮想環境を作成できます。

1. Python 仮想環境を構築するには、次のコマンドを使用します。次の例では、 パッケージ`scipy`と を仮想環境パッケージ`matplotlib`にインストールし、アーカイブを Amazon S3 の場所にコピーします。

   > **Important**
   >
   > EMR Serverless で使用するのと同じバージョンの Python、つまり Amazon EMRリリース 6.6.0 用の Python 3.7.10 を使用する類似の Amazon Linux 2 環境で、次のコマンドを実行する必要があります。Dockerfile の例は、EMRServerless Samples GitHub リポジトリにあります。

```
# initialize a python virtual environment
```

```
python3 -m venv pyspark_venvsource
source pyspark_venvsource/bin/activate

# optionally, ensure pip is up-to-date
pip3 install --upgrade pip

# install the python packages
pip3 install scipy
pip3 install matplotlib

# package the virtual environment into an archive
pip3 install venv-pack
venv-pack -f -o pyspark_venv.tar.gz

# copy the archive to an S3 location
aws s3 cp pyspark_venv.tar.gz s3://DOC-EXAMPLE-BUCKET/EXAMPLE-PREFIX/

# optionally, remove the virtual environment directory
rm -fr pyspark_venvsource
```

2. Python 仮想環境を使用するようにプロパティを設定した Spark ジョブを送信します。

```
--conf spark.archives=s3://DOC-EXAMPLE-BUCKET/EXAMPLE-PREFIX/
pyspark_venv.tar.gz#environment
--conf spark.emr-serverless.driverEnv.PYSPARK_DRIVER_PYTHON=./environment/bin/
python
--conf spark.emr-serverless.driverEnv.PYSPARK_PYTHON=./environment/bin/python
--conf spark.executorEnv.PYSPARK_PYTHON=./environment/bin/python
```

元の Python バイナリを上書きしない場合、前の設定シーケンスの 2 番目の設定は になること に注意してください`--conf spark.executorEnv.PYSPARK_PYTHON=python`。

PySpark ジョブに Python 仮想環境を使用する方法の詳細については、「Virtualenv の使用」を参照してください。Spark ジョブを送信する方法のその他の例については、「」を参照してくださいSpark ジョブ。

Python ライブラリを使用するように PySpark ジョブを設定する

Amazon EMRリリース 6.12.0 以降では、追加のセットアップPyArrowなしで、pandas 、、NumPyなどの一般的なデータサイエンス Python ライブラリを使用するようにEMRサーバーレス PySparkジョブを直接設定できます。

次の例は、ジョブの各 Python ライブラリを PySparkパッケージ化する方法を示しています。

NumPy

NumPy は、数学、ソート、ランダムシミュレーション、および基本統計のための多次元配列とオペレーションを提供する科学コンピューティング用の Python ライブラリです。を使用するには NumPy、次のコマンドを実行します。

```
import numpy
```

pandas

pandas は、 上に構築された Python ライブラリです NumPy。pandas ライブラリは、データサイエンティストにDataFrameデータ構造とデータ分析ツールを提供します。pandas を使用するには、次のコマンドを実行します。

```
import pandas
```

PyArrow

PyArrow は、ジョブのパフォーマンスを向上させるためにインメモリ列指向データを管理する Python ライブラリ PyArrow です。 は、列指向形式でデータを表現および交換する標準的な方法である Apache Arrow の言語間開発仕様に基づいています。を使用するには PyArrow、次のコマンドを実行します。

```
import pyarrow
```

Serverless でのさまざまな Python EMR バージョンの使用

のユースケースに加えてServerless での Python EMR ライブラリの使用、Python 仮想環境を使用して、Amazon EMR Serverless アプリケーションの Amazon EMRリリースにパッケージ化されているバージョンとは異なる Python バージョンを使用することもできます。これを行うには、使用する Python バージョンを使用して Python 仮想環境を構築する必要があります。

Python 仮想環境からジョブを送信するには

1. 次の例のコマンドを使用して仮想環境を構築します。この例では、Python 3.9.9 を仮想環境パッケージにインストールし、アーカイブを Amazon S3 の場所にコピーします。

> **Important**
>
> Amazon EMRリリース 7.0.0 以降を使用している場合は、EMRサーバーレスアプリケーションに使用するものと同様の Amazon Linux 2023 環境でコマンドを実行する必要があります。
> リリース 6.15.0 以前を使用している場合は、同様の Amazon Linux 2 環境で次のコマンドを実行する必要があります。

```
# install Python 3.9.9 and activate the venv
yum install -y gcc openssl-devel bzip2-devel libffi-devel tar gzip wget make
wget https://www.python.org/ftp/python/3.9.9/Python-3.9.9.tgz && \
tar xzf Python-3.9.9.tgz && cd Python-3.9.9 && \
./configure --enable-optimizations && \
make altinstall

# create python venv with Python 3.9.9
python3.9 -m venv pyspark_venv_python_3.9.9 --copies
source pyspark_venv_python_3.9.9/bin/activate

# copy system python3 libraries to venv
cp -r /usr/local/lib/python3.9/* ./pyspark_venv_python_3.9.9/lib/python3.9/

# package venv to archive.
# **Note** that you have to supply --python-prefix option
# to make sure python starts with the path where your
# copied libraries are present.
# Copying the python binary to the "environment" directory.
pip3 install venv-pack
venv-pack -f -o pyspark_venv_python_3.9.9.tar.gz --python-prefix /home/hadoop/
environment

# stage the archive in S3
aws s3 cp pyspark_venv_python_3.9.9.tar.gz s3://<path>

# optionally, remove the virtual environment directory
rm -fr pyspark_venv_python_3.9.9
```

2. Python 仮想環境を使用するようにプロパティを設定し、Spark ジョブを送信します。

```
# note that the archive suffix "environment" is the same as the directory where you
 copied the Python binary.
--conf spark.archives=s3://DOC-EXAMPLE-BUCKET/EXAMPLE-PREFIX/
pyspark_venv_python_3.9.9.tar.gz#environment
--conf spark.emr-serverless.driverEnv.PYSPARK_DRIVER_PYTHON=./environment/bin/
python
--conf spark.emr-serverless.driverEnv.PYSPARK_PYTHON=./environment/bin/python
--conf spark.executorEnv.PYSPARK_PYTHON=./environment/bin/python
```

PySpark ジョブに Python 仮想環境を使用する方法の詳細については、「Virtualenv の使用」を参照してください。Spark ジョブを送信する方法のその他の例については、「」を参照してくださいSpark ジョブ。

EMR サーバーレスOSSでの Delta Lake の使用

Amazon EMRバージョン 6.9.0 以降

> **Note**
>
> Amazon EMR 7.0.0 以降では、Delta Lake 3.0.0 を使用してdelta-core.jarファイルの名前を に変更しますdelta-spark.jar。Amazon 7EMR.0.0 以降を使用する場合は、設定delta-spark.jarで を指定してください。

Amazon EMR 6.9.0 以降には Delta Lake が含まれているため、Delta Lake を自分でパッケージ化したり、EMRサーバーレスジョブで --packages フラグを指定したりする必要がありません。

1. EMR サーバーレスジョブを送信するときは、次の設定プロパティがあり、sparkSubmitParametersフィールドに次のパラメータが含まれていることを確認してください。

   ```
   --conf spark.jars=/usr/share/aws/delta/lib/delta-core.jar,/usr/share/aws/delta/lib/
   delta-storage.jar
       --conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension
       --conf
    spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog
   ```

2. Delta テーブルの作成と読み取りdelta_sample.pyをテストするローカル を作成します。

```
# delta_sample.py
    from pyspark.sql import SparkSession

    import uuid

    url = "s3://DOC-EXAMPLE-BUCKET/delta-lake/output/%s/" % str(uuid.uuid4())
    spark = SparkSession.builder.appName("DeltaSample").getOrCreate()

    ## creates a Delta table and outputs to target S3 bucket
    spark.range(5).write.format("delta").save(url)

    ## reads a Delta table and outputs to target S3 bucket
    spark.read.format("delta").load(url).show
```

3. 以下を使用 AWS CLIで、delta_sample.pyファイルを Amazon S3 バケットにアップロードします。次に、 start-job-run コマンドを使用して、既存の EMR Serverless アプリケーションにジョブを送信します。

```
aws s3 cp delta_sample.py s3://DOC-EXAMPLE-BUCKET/code/

    aws emr-serverless start-job-run \
        --application-id application-id \
        --execution-role-arn job-role-arn \
        --name emr-delta \
        --job-driver '{
            "sparkSubmit": {
                "entryPoint": "s3://DOC-EXAMPLE-BUCKET/code/delta_sample.py",
                "sparkSubmitParameters": "--conf spark.jars=/usr/share/
aws/delta/lib/delta-core.jar,/usr/share/aws/delta/lib/delta-storage.jar --
conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension --conf
 spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog"
            }
        }'
```

Delta Lake で Python ライブラリを使用するには、ライブラリを依存関係としてパッケージ化するか、カスタムイメージ として使用することでdelta-coreライブラリを追加できます。

または、 を使用して、 delta-core JAR ファイルから Python ライブラリSparkContext.addPyFileを追加することもできます。

```
import glob
from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()
spark.sparkContext.addPyFile(glob.glob("/usr/share/aws/delta/lib/delta-core_*.jar")[0])
```

Amazon EMRバージョン 6.8.0 以前

Amazon 6EMR.8.0 以前を使用している場合は、以下の手順に従ってサーバーレスアプリケーションで Delta Lake OSS EMR を使用します。

1. Amazon EMR Serverless アプリケーションで Spark のバージョンと互換性のあるオープンソースバージョンの Delta Lake を構築するには、Delta GitHub に移動し、指示に従ってください。
2. Delta Lake ライブラリを の Amazon S3 バケットにアップロードする AWS アカウント.
3. アプリケーション設定でEMRサーバーレスジョブを送信するときは、バケットに現在存在する Delta Lake JAR ファイルを含めます。

   ```
   --conf spark.jars=s3://DOC-EXAMPLE-BUCKET/jars/delta-core_2.12-1.1.0.jar
   ```

4. Delta テーブルとの間で読み書きを確実にできるようにするには、サンプル PySparkテストを実行します。

   ```
   from pyspark import SparkConf, SparkContext
       from pyspark.sql import HiveContext, SparkSession

       import uuid

       conf = SparkConf()
       sc = SparkContext(conf=conf)
       sqlContext = HiveContext(sc)

       url = "s3://DOC-EXAMPLE-BUCKET/delta-lake/output/1.0.1/%s/" % str(uuid.uuid4())

       ## creates a Delta table and outputs to target S3 bucket
       session.range(5).write.format("delta").save(url)

       ## reads a Delta table and outputs to target S3 bucket
       session.read.format("delta").load(url).show
   ```

Airflow からのEMRサーバーレスジョブの送信

Apache Airflow の Amazon プロバイダーは、EMRサーバーレス演算子を提供します。演算子の詳細については、Apache Airflow ドキュメントの「Amazon EMR Serverless Operators」を参照してください。

`EmrServerlessCreateApplicationOperator` を使用して Spark または Hive アプリケーションを作成できます。`EmrServerlessStartJobOperator` を使用して、新しいアプリケーションで 1 つ以上のジョブを開始することもできます。

Airflow 2.2.2 で Amazon Managed Workflows for Apache Airflow (MWAA) で 演算子を使用するには、次の行を`requirements.txt`ファイルに追加し、新しいファイルを使用するようにMWAA環境を更新します。

```
apache-airflow-providers-amazon==6.0.0
boto3>=1.23.9
```

EMR サーバーレスサポートが Amazon プロバイダーのリリース 5.0.0 に追加されました。リリース 6.0.0 は、Airflow 2.2.2 と互換性のある最後のバージョンです。上の Airflow 2.4.3 では、以降のバージョンを使用できますMWAA。

次の省略例は、アプリケーションを作成し、複数の Spark ジョブを実行してから、アプリケーションを停止する方法を示しています。完全な例は、EMRServerless Samples GitHub リポジトリにあります。`sparkSubmit` 設定の詳細については、「」を参照してくださいSpark ジョブ。

```
from datetime import datetime

from airflow import DAG
from airflow.providers.amazon.aws.operators.emr import (
    EmrServerlessCreateApplicationOperator,
    EmrServerlessStartJobOperator,
    EmrServerlessDeleteApplicationOperator,
)

# Replace these with your correct values
JOB_ROLE_ARN = "arn:aws:iam::account-id:role/emr_serverless_default_role"
S3_LOGS_BUCKET = "amzn-s3-demo-bucket"

DEFAULT_MONITORING_CONFIG = {
    "monitoringConfiguration": {
        "s3MonitoringConfiguration": {"logUri": f"s3://amzn-s3-demo-bucket/logs/"}
```

```
    },
}

with DAG(
    dag_id="example_endtoend_emr_serverless_job",
    schedule_interval=None,
    start_date=datetime(2021, 1, 1),
    tags=["example"],
    catchup=False,
) as dag:
    create_app = EmrServerlessCreateApplicationOperator(
        task_id="create_spark_app",
        job_type="SPARK",
        release_label="emr-6.7.0",
        config={"name": "airflow-test"},
    )

    application_id = create_app.output

    job1 = EmrServerlessStartJobOperator(
        task_id="start_job_1",
        application_id=application_id,
        execution_role_arn=JOB_ROLE_ARN,
        job_driver={
            "sparkSubmit": {
                "entryPoint": "local:///usr/lib/spark/examples/src/main/python/
pi_fail.py",
            }
        },
        configuration_overrides=DEFAULT_MONITORING_CONFIG,
    )

    job2 = EmrServerlessStartJobOperator(
        task_id="start_job_2",
        application_id=application_id,
        execution_role_arn=JOB_ROLE_ARN,
        job_driver={
            "sparkSubmit": {
                "entryPoint": "local:///usr/lib/spark/examples/src/main/python/pi.py",
                "entryPointArguments": ["1000"]
            }
        },
        configuration_overrides=DEFAULT_MONITORING_CONFIG,
    )
```

```
    delete_app = EmrServerlessDeleteApplicationOperator(
        task_id="delete_app",
        application_id=application_id,
        trigger_rule="all_done",
    )

    (create_app >> [job1, job2] >> delete_app)
```

EMR Serverless での Hive ユーザー定義関数の使用

Hive ユーザー定義関数 (UDFs) を使用すると、レコードまたはレコードのグループを処理するカスタム関数を作成できます。このチュートリアルでは、既存の Amazon EMR Serverless UDF アプリケーションでサンプルを使用して、クエリ結果を出力するジョブを実行します。アプリケーションのセットアップ方法については、「」を参照してくださいAmazon EMR Serverless の開始方法。

Serverless UDFで EMR を使用するには

1. サンプル GitHubの に移動しますUDF。リポジトリをクローンし、使用する git ブランチに切り替えます。リポジトリの pom.xml ファイルmaven-compiler-plugin内の を更新して、ソースを作成します。また、ターゲットの Java バージョン設定を に更新します1.8。mvn package -DskipTests を実行して、サンプル を含む JAR ファイルを作成しますUDFs。
2. JAR ファイルを作成したら、次のコマンドを使用して S3 バケットにアップロードします。

   ```
   aws s3 cp brickhouse-0.8.2-JS.jar s3://DOC-EXAMPLE-BUCKET/jars/
   ```

3. サンプルUDF関数のいずれかを使用するサンプルファイルを作成します。このクエリを として保存しudf_example.q、S3 バケットにアップロードします。

   ```
   add jar s3://DOC-EXAMPLE-BUCKET/jars/brickhouse-0.8.2-JS.jar;
   CREATE TEMPORARY FUNCTION from_json AS 'brickhouse.udf.json.FromJsonUDF';
   select from_json('{"key1":[0,1,2], "key2":[3,4,5,6], "key3":[7,8,9]}', map("",
    array(cast(0 as int))));
   select from_json('{"key1":[0,1,2], "key2":[3,4,5,6], "key3":[7,8,9]}', map("",
    array(cast(0 as int))))["key1"][2];
   ```

4. 次の Hive ジョブを送信します。

   ```
   aws emr-serverless start-job-run \
     --application-id application-id \
   ```

```
  --execution-role-arn job-role-arn \
  --job-driver '{
    "hive": {
        "query": "s3://DOC-EXAMPLE-BUCKET/queries/udf_example.q",
        "parameters": "--hiveconf hive.exec.scratchdir=s3://DOC-EXAMPLE-BUCKET/emr-
serverless-hive/scratch --hiveconf hive.metastore.warehouse.dir=s3://'$BUCKET'/emr-
serverless-hive/warehouse"
    }
}' --configuration-overrides '{
    "applicationConfiguration": [{
        "classification": "hive-site",
        "properties": {
            "hive.driver.cores": "2",
            "hive.driver.memory": "6G"
        }
    }],
    "monitoringConfiguration": {
        "s3MonitoringConfiguration": {
            "logUri": "s3://DOC-EXAMPLE-BUCKET/logs/"
        }
    }
}'
```

5. get-job-run コマンドを使用して、ジョブの状態を確認します。状態が に変わるまで待ちますSUCCESS。

```
aws emr-serverless get-job-run --application-id application-id --job-run-id job-id
```

6. 次のコマンドを使用して出力ファイルをダウンロードします。

```
aws s3 cp --recursive s3://DOC-EXAMPLE-BUCKET/logs/applications/application-id/
jobs/job-id/HIVE_DRIVER/ .
```

stdout.gz ファイルは次のようになります。

```
{"key1":[0,1,2],"key2":[3,4,5,6],"key3":[7,8,9]}
2
```

EMR Serverless でのカスタムイメージの使用

トピック

カスタム Python バージョンを使用する

別のバージョンの Python を使用するようにカスタムイメージを構築できます。例えば、Spark ジョブに Python バージョン 3.10 を使用するには、次のコマンドを実行します。

```
FROM public.ecr.aws/emr-serverless/spark/emr-6.9.0:latest

USER root

# install python 3
RUN yum install -y gcc openssl-devel bzip2-devel libffi-devel tar gzip wget make
RUN wget https://www.python.org/ftp/python/3.10.0/Python-3.10.0.tgz && \
tar xzf Python-3.10.0.tgz && cd Python-3.10.0 && \
./configure --enable-optimizations && \
make altinstall

# EMRS will run the image as hadoop
USER hadoop:hadoop
```

Spark ジョブを送信する前に、次のように Python 仮想環境を使用するようにプロパティを設定します。

```
--conf spark.emr-serverless.driverEnv.PYSPARK_DRIVER_PYTHON=/usr/local/bin/python3.10
--conf spark.emr-serverless.driverEnv.PYSPARK_PYTHON=/usr/local/bin/python3.10
--conf spark.executorEnv.PYSPARK_PYTHON=/usr/local/bin/python3.10
```

カスタム Java バージョンを使用する

次の例は、Spark ジョブに Java 11 を使用するカスタムイメージを構築する方法を示しています。

```
FROM public.ecr.aws/emr-serverless/spark/emr-6.9.0:latest

USER root
```

```
# install JDK 11
RUN sudo amazon-linux-extras install java-openjdk11

# EMRS will run the image as hadoop
USER hadoop:hadoop
```

Spark ジョブを送信する前に、次のように Java 11 を使用するように Spark プロパティを設定します。

```
--conf spark.executorEnv.JAVA_HOME=/usr/lib/jvm/java-11-
openjdk-11.0.16.0.8-1.amzn2.0.1.x86_64
--conf spark.emr-serverless.driverEnv.JAVA_HOME=/usr/lib/jvm/java-11-
openjdk-11.0.16.0.8-
```

データサイエンスイメージの構築

次の例は、Pandas や などの一般的なデータサイエンス Python パッケージを含める方法を示しています NumPy。

```
FROM public.ecr.aws/emr-serverless/spark/emr-6.9.0:latest

USER root

# python packages
RUN pip3 install boto3 pandas numpy
RUN pip3 install -U scikit-learn==0.23.2 scipy
RUN pip3 install sk-dist
RUN pip3 install xgboost

# EMR Serverless will run the image as hadoop
USER hadoop:hadoop
```

Apache Sedona による地理空間データの処理

次の例は、地理空間処理に Apache Sedona を含めるイメージを構築する方法を示しています。

```
FROM public.ecr.aws/emr-serverless/spark/emr-6.9.0:latest

USER root

RUN yum install -y wget
```

```
RUN wget https://repo1.maven.org/maven2/org/apache/sedona/sedona-core-3.0_2.12/1.3.0-
incubating/sedona-core-3.0_2.12-1.3.0-incubating.jar -P /usr/lib/spark/jars/
RUN pip3 install apache-sedona

# EMRS will run the image as hadoop
USER hadoop:hadoop
```

Amazon EMR Serverless での Apache Spark の Amazon Redshift 統合の使用

Amazon EMRリリース 6.9.0 以降では、すべてのリリースイメージに Apache Spark と Amazon Redshift 間のコネクタが含まれています。このコネクタを使用すると、Amazon EMR Serverless で Spark を使用して Amazon Redshift に保存されているデータを処理できます。このインテグレーションは、spark-redshift オープンソースコネクタをベースにしています。Amazon EMR Serverless の場合、Apache Spark 用の Amazon Redshift 統合がネイティブ統合として含まれています。

トピック

- Apache Spark 用の Amazon Redshift 統合を使用した Spark アプリケーションの起動
- Amazon Redshift integration for Apache Spark による認証
- Amazon Redshift に対する読み書き
- Spark コネクタを使用する際の考慮事項と制限事項

Apache Spark 用の Amazon Redshift 統合を使用した Spark アプリケーションの起動

EMR Serverless 6.9.0 との統合を使用するには、必要な Spark-Redshift 依存関係を Spark ジョブに渡す必要があります。`--jars` を使用して Redshift コネクタ関連のライブラリを含めます。ファイルの保存先として `--jars` オプションでサポートされている他の場所を確認するには、Apache Spark ドキュメントの「Advanced Dependency Management」セクションを参照してください。

- `spark-redshift.jar`
- `spark-avro.jar`
- `RedshiftJDBC.jar`
- `minimal-json.jar`

Amazon EMRリリース 6.10.0 以降ではminimal-json.jar、依存関係は必要なく、デフォルトでは他の依存関係を各クラスターに自動的にインストールします。以下の例は、Apache Spark 用の Amazon Redshift インテグレーションを使用して Spark アプリケーションを起動する方法を示しています。

Amazon EMR 6.10.0 +

EMR Serverless リリース 6.10.0 以降の Apache Spark 用の Amazon Redshift 統合を使用して、Amazon EMR Serverless で Spark ジョブを起動します。

```
spark-submit my_script.py
```

Amazon EMR 6.9.0

EMR Serverless リリース 6.9.0 の Apache Spark 用の Amazon Redshift 統合を使用して Amazon EMR Serverless で Spark ジョブを起動するには、次の例に示すように --jarsオプションを使用します。--jars オプションでリストされているパスは、JARファイルのデフォルトパスであることに注意してください。

```
--jars
    /usr/share/aws/redshift/jdbc/RedshiftJDBC.jar,
    /usr/share/aws/redshift/spark-redshift/lib/spark-redshift.jar,
    /usr/share/aws/redshift/spark-redshift/lib/spark-avro.jar,
    /usr/share/aws/redshift/spark-redshift/lib/minimal-json.jar
```

```
spark-submit \
  --jars /usr/share/aws/redshift/jdbc/RedshiftJDBC.jar,/usr/share/aws/redshift/
spark-redshift/lib/spark-redshift.jar,/usr/share/aws/redshift/spark-redshift/lib/
spark-avro.jar,/usr/share/aws/redshift/spark-redshift/lib/minimal-json.jar \
  my_script.py
```

Amazon Redshift integration for Apache Spark による認証

使用アイテム AWS Secrets Manager 認証情報を取得して Amazon Redshift に接続するには

Secrets Manager に認証情報を保存して Amazon Redshift に対して安全に認証し、Spark ジョブでGetSecretValueAPIを呼び出して取得できます。

```
from pyspark.sql import SQLContextimport boto3

sc = # existing SparkContext
sql_context = SQLContext(sc)

secretsmanager_client = boto3.client('secretsmanager',
 region_name=os.getenv('AWS_REGION'))
secret_manager_response = secretsmanager_client.get_secret_value(
    SecretId='string',
    VersionId='string',
    VersionStage='string'
)
username = # get username from secret_manager_response
password = # get password from secret_manager_response
url = "jdbc:redshift://redshifthost:5439/database?user=" + username + "&password="
 + password

# Access to Redshift cluster using Spark
```

JDBC ドライバーを使用して Amazon Redshift を認証する

内でユーザー名とパスワードを設定する JDBC URL

で Amazon Redshift データベースの名前とパスワードを指定することで、Amazon Redshift クラスターに対して Spark ジョブを認証できますJDBCURL。

> **Note**
>
> でデータベース認証情報を渡すとURL、 にアクセスできるすべてのユーザーが認証情報にアクセスURLできるようになります。この方法は、安全な方法ではないため、一般的にはお勧めしません。

セキュリティがアプリケーションにとって問題でない場合は、次の形式を使用して、 JDBC でユーザー名とパスワードを設定できますURL。

```
jdbc:redshift://redshifthost:5439/database?user=username&password=password
```

Amazon EMR Serverless ジョブ実行ロールで IAM ベースの認証を使用する

Amazon EMR Serverless リリース 6.9.0 以降、Amazon Redshift JDBCドライバー 2.1 以降は環境にパッケージ化されています。JDBC ドライバー 2.1 以降では、raw ユーザー名とパスワードを含めJDBCURLずに を指定できます。

代わりに、`jdbc:redshift:iam://` スキームを指定できます。これにより、EMRサーバーレスジョブ実行ロールを使用して認証情報を自動的に取得するようにJDBCドライバーに指示します。詳細については、「Amazon Redshift 管理ガイド」のIAM「認証情報を使用するように JDBCまたはODBC接続を設定する」を参照してください。 その例URLを次に示します。

```
jdbc:redshift:iam://examplecluster.abc123xyz789.us-west-2.redshift.amazonaws.com:5439/
dev
```

指定された条件が満たされた場合、ジョブ実行ロールには次のアクセス許可が必要です。

アクセス許可	ジョブ実行ロールで必要になる条件
`redshift:GetClusterCredentials`	JDBC ドライバーが Amazon Redshift から認証情報を取得するために必要です
`redshift:DescribeCluster`	Amazon Redshift クラスターと AWS リージョン エンドポイントJDBCURLではなく の
`redshift-serverless:GetCredentials`	JDBC ドライバーが Amazon Redshift Serverless から認証情報を取得するために必要です
`redshift-serverless:GetWorkgroup`	Amazon Redshift Serverless を使用していて、ワークグループ名とリージョンの点URLで を指定する場合に必要です

別の 内の Amazon Redshift への接続 VPC

でプロビジョニングされた Amazon Redshift クラスターまたは Amazon Redshift Serverless ワークグループを設定するときはVPC、Amazon EMR Serverless アプリケーションが リソースにアクセスするようにVPC接続を設定する必要があります。EMR Serverless アプリケーションでVPC接続を設定する方法の詳細については、「」を参照してくださいVPC アクセスの設定。

- プロビジョニングされた Amazon Redshift クラスターまたは Amazon Redshift Serverless ワークグループがパブリックアクセス可能な場合は、EMRサーバーレスアプリケーションの作成時にNATゲートウェイがアタッチされた 1 つ以上のプライベートサブネットを指定できます。
- プロビジョニングされた Amazon Redshift クラスターまたは Amazon Redshift Serverless ワークグループがパブリックアクセス可能でない場合は、「」の説明に従って、Amazon Redshift クラスターの Amazon Redshift マネージドVPCエンドポイントを作成する必要がありますVPC アクセスの設定。または、「Amazon Redshift 管理ガイド」の「Amazon Redshift Serverless への接続」の説明に従って、Amazon Redshift Serverless ワークグループを作成することもできます。EMR Serverless アプリケーションの作成時に指定したプライベートサブネットにクラスターまたはサブグループを関連付ける必要があります。

> **Note**
>
> IAM ベース認証を使用し、EMRサーバーレスアプリケーションのプライベートサブネットにNATゲートウェイがアタッチされていない場合は、Amazon Redshift または Amazon Redshift Serverless のサブネットにもVPCエンドポイントを作成する必要があります。これにより、JDBCドライバーは認証情報を取得できます。

Amazon Redshift に対する読み書き

次のコード例では PySpark 、 を使用して、データソースと Spark を使用して Amazon Redshift データベースとの間でサンプルデータの読み取りAPIと書き込みを行いますSQL。

Data source API

PySpark を使用して、データソース を使用して Amazon Redshift データベースとの間でサンプルデータを読み書きしますAPI。

```
import boto3
from pyspark.sql import SQLContext

sc = # existing SparkContext
sql_context = SQLContext(sc)

url = "jdbc:redshift:iam://redshifthost:5439/database"
aws_iam_role_arn = "arn:aws:iam::account-id:role/role-name"

df = sql_context.read \
```

```
    .format("io.github.spark_redshift_community.spark.redshift") \
    .option("url", url) \
    .option("dbtable", "table-name") \
    .option("tempdir", "s3://path/for/temp/data") \
    .option("aws_iam_role", "aws-iam-role-arn") \
    .load()

df.write \
    .format("io.github.spark_redshift_community.spark.redshift") \
    .option("url", url) \
    .option("dbtable", "table-name-copy") \
    .option("tempdir", "s3://path/for/temp/data") \
    .option("aws_iam_role", "aws-iam-role-arn") \
    .mode("error") \
    .save()
```

SparkSQL

Spark PySpark を使用して Amazon Redshift データベースとの間でサンプルデータを読み書きするには、 を使用しますSQL。

```
import boto3
import json
import sys
import os
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .enableHiveSupport() \
    .getOrCreate()

url = "jdbc:redshift:iam://redshifthost:5439/database"
aws_iam_role_arn = "arn:aws:iam::account-id:role/role-name"

bucket = "s3://path/for/temp/data"
tableName = "table-name" # Redshift table name

s = f"""CREATE TABLE IF NOT EXISTS {table-name} (country string, data string)
    USING io.github.spark_redshift_community.spark.redshift
    OPTIONS (dbtable '{table-name}', tempdir '{bucket}', url '{url}', aws_iam_role
 '{aws-iam-role-arn}' ); """
```

```
spark.sql(s)

columns = ["country" ,"data"]
data = [("test-country","test-data")]
df = spark.sparkContext.parallelize(data).toDF(columns)

# Insert data into table
df.write.insertInto(table-name, overwrite=False)
df = spark.sql(f"SELECT * FROM {table-name}")
df.show()
```

Spark コネクタを使用する際の考慮事項と制限事項

- Amazon SSLの Spark JDBCから Amazon Redshift EMRへの接続を有効にすることをお勧めします。
- で Amazon Redshift クラスターの認証情報を管理することをお勧めします。 AWS Secrets Manager ベストプラクティスです。「 の使用」 AWS Secrets Manager 例の Amazon Redshift に接続するための認証情報を取得するには、 を使用します。
- Amazon Redshift 認証パラメータaws_iam_roleの パラメータを使用して IAMロールを渡すことをお勧めします。
- 現在、パラメータ tempformat は Parquet 形式をサポートしていません。
- は Amazon S3 の場所tempdirURIを指します。この一時ディレクトリは、自動的にはクリーンアップされないため、追加コストが発生する可能性があります。
- Amazon Redshift については、次の推奨事項を検討してください。
  - Amazon Redshift クラスターにパブリックにアクセスできないようにすることをお勧めします。
  - Amazon Redshift 監査ログ作成を有効にすることをお勧めします。
  - Amazon Redshift 保管時の暗号化を有効にすることをお勧めします。
- Amazon S3 については、次の推奨事項を検討してください。
  - Amazon S3 バケットへのパブリックアクセスをブロックすることをお勧めします。
  - Amazon S3 サーバー側の暗号化を使用して、使用する Amazon S3 バケットを暗号化することをお勧めします。
  - Amazon S3 ライフサイクルポリシーを使用して、Amazon S3 バケットの保持ルールを定義することをお勧めします。
  - Amazon は、オープンソースからイメージにインポートされたコードEMRを常に検証します。セキュリティのため、Spark から Amazon S3 への次の認証方法はサポートされていません。

- 設定 AWS hadoop-env 設定分類の アクセスキー
- エンコード AWS の アクセスキー tempdir URI

コネクタとそのサポートされているパラメータの使用方法の詳細については、次のリソースを参照してください。

- 「Amazon Redshift 管理ガイド」の「Amazon Redshift integration for Apache Spark」
- Github の spark-redshift コミュニティリポジトリ

Amazon EMR Serverless を使用した DynamoDB への接続

このチュートリアルでは、米国地名委員会から Amazon S3 バケットにデータのサブセットをアップロードし、Amazon EMR Serverless の Hive または Spark を使用して、クエリできる Amazon DynamoDB テーブルにデータをコピーします。

ステップ 1: Amazon S3 バケットにデータをアップロードする

Amazon S3 バケットを作成するには、「Amazon Simple Storage Service コンソールユーザーガイド」の「バケットの作成」の手順に従います。への参照*amzn-s3-demo-bucket*を、新しく作成したバケットの名前に置き換えます。これで、EMRServerless アプリケーションはジョブを実行する準備が整いました。

1. 次のコマンドfeatures.zipを使用して、サンプルデータアーカイブをダウンロードします。

```
wget https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/
features.zip
```

2. アーカイブからfeatures.txtファイルを抽出し、ファイル内の最初の数行を表示します。

```
unzip features.zip
head features.txt
```

結果は次のように表示されます。

```
1535908|Big Run|Stream|WV|38.6370428|-80.8595469|794
875609|Constable Hook|Cape|NJ|40.657881|-74.0990309|7
1217998|Gooseberry Island|Island|RI|41.4534361|-71.3253284|10
26603|Boone Moore Spring|Spring|AZ|34.0895692|-111.410065|3681
```

```
1506738|Missouri Flat|Flat|WA|46.7634987|-117.0346113|2605
1181348|Minnow Run|Stream|PA|40.0820178|-79.3800349|1558
1288759|Hunting Creek|Stream|TN|36.343969|-83.8029682|1024
533060|Big Charles Bayou|Bay|LA|29.6046517|-91.9828654|0
829689|Greenwood Creek|Stream|NE|41.596086|-103.0499296|3671
541692|Button Willow Island|Island|LA|31.9579389|-93.0648847|98
```

各行のフィールドは、一意の識別子、名前、自然特徴のタイプ、状態、緯度、経度、高さをフィート単位で示しています。

3. Amazon S3 にデータをアップロードする

```
aws s3 cp features.txt s3://amzn-s3-demo-bucket/features/
```

ステップ 2: Hive テーブルを作成する

Apache Spark または Hive を使用して、Amazon S3 にアップロードされたデータを含む新しい Hive テーブルを作成します。

Spark

Spark で Hive テーブルを作成するには、次のコマンドを実行します。

```
import org.apache.spark.sql.SparkSession

val sparkSession = SparkSession.builder().enableHiveSupport().getOrCreate()

sparkSession.sql("CREATE TABLE hive_features \
    (feature_id BIGINT, \
    feature_name STRING, \
    feature_class STRING, \
    state_alpha STRING, \
    prim_lat_dec DOUBLE, \
    prim_long_dec DOUBLE, \
    elev_in_ft BIGINT) \
    ROW FORMAT DELIMITED \
    FIELDS TERMINATED BY '|' \
    LINES TERMINATED BY '\n' \
    LOCATION 's3://DOC-EXAMPLE_BUCKET/features';")
```

これで、features.txtファイルからのデータを含む Hive テーブルが入力されました。データがテーブルにあることを確認するには、次の例に示すように Spark SQLクエリを実行します。

```
sparkSession.sql(
    "SELECT state_alpha, COUNT(*) FROM hive_features GROUP BY state_alpha;")
```

Hive

Hive で Hive テーブルを作成するには、次のコマンドを実行します。

```
CREATE TABLE hive_features
    (feature_id             BIGINT,
    feature_name            STRING ,
    feature_class           STRING ,
    state_alpha             STRING,
    prim_lat_dec            DOUBLE ,
    prim_long_dec           DOUBLE ,
    elev_in_ft              BIGINT)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY '|'
    LINES TERMINATED BY '\n'
    LOCATION 's3://amzn-s3-demo-bucket/features';
```

これで、features.txtファイルからのデータを含む Hive テーブルが作成されました。データがテーブルにあることを確認するには、次の例に示すように HiveQL クエリを実行します。

```
SELECT state_alpha, COUNT(*) FROM hive_features GROUP BY state_alpha;
```

ステップ 3: DynamoDB にデータをコピーする

Spark または Hive を使用して、データを新しい DynamoDB テーブルにコピーします。

Spark

前のステップで作成した Hive テーブルから DynamoDB にデータをコピーするには、「データを DynamoDB にコピーする」のステップ 1～3 に従います。これにより、 という名前の新しい DynamoDB テーブルが作成されますFeatures。その後、次の例に示すように、テキストファイルから直接データを読み取って DynamoDB テーブルにコピーできます。

```
import com.amazonaws.services.dynamodbv2.model.AttributeValue
```

```
import org.apache.hadoop.dynamodb.DynamoDBItemWritable
import org.apache.hadoop.dynamodb.read.DynamoDBInputFormat
import org.apache.hadoop.io.Text
import org.apache.hadoop.mapred.JobConf
import org.apache.spark.SparkContext

import scala.collection.JavaConverters._

object EmrServerlessDynamoDbTest {

    def main(args: Array[String]): Unit = {

        jobConf.set("dynamodb.input.tableName", "Features")
        jobConf.set("dynamodb.output.tableName", "Features")
        jobConf.set("dynamodb.region", "region")

        jobConf.set("mapred.output.format.class",
 "org.apache.hadoop.dynamodb.write.DynamoDBOutputFormat")
        jobConf.set("mapred.input.format.class",
 "org.apache.hadoop.dynamodb.read.DynamoDBInputFormat")

        val rdd = sc.textFile("s3://amzn-s3-demo-bucket/ddb-connector/")
            .map(row => {
                val line = row.split("\\|")
                val item = new DynamoDBItemWritable()

                val elevInFt = if (line.length > 6) {
                    new AttributeValue().withN(line(6))
                } else {
                    new AttributeValue().withNULL(true)
                }

                item.setItem(Map(
                    "feature_id" -> new AttributeValue().withN(line(0)),
                    "feature_name" -> new AttributeValue(line(1)),
                    "feature_class" -> new AttributeValue(line(2)),
                    "state_alpha" -> new AttributeValue(line(3)),
                    "prim_lat_dec" -> new AttributeValue().withN(line(4)),
                    "prim_long_dec" -> new AttributeValue().withN(line(5)),
                    "elev_in_ft" -> elevInFt)
                    .asJava)
                (new Text(""), item)
        })
        rdd.saveAsHadoopDataset(jobConf)
```

```
    }
}
```

Hive

前のステップで作成した Hive テーブルから DynamoDB にデータをコピーするには、「データを DynamoDB にコピーする」の手順に従います。

ステップ 4: DynamoDB からデータをクエリする

Spark または Hive を使用して DynamoDB テーブルをクエリします。

Spark

前のステップで作成した DynamoDB テーブルからデータをクエリするには、Spark SQLまたは Spark のいずれかを使用できます MapReduce API。

Example – Spark を使用して DynamoDB テーブルをクエリする SQL

次の Spark SQLクエリは、すべての特徴量タイプのリストをアルファベット順に返します。

```
val dataFrame = sparkSession.sql("SELECT DISTINCT feature_class \
    FROM ddb_features \
    ORDER BY feature_class;")
```

次の Spark SQLクエリは、M で始まるすべてのレイクのリストを返します。

```
val dataFrame = sparkSession.sql("SELECT feature_name, state_alpha \
    FROM ddb_features \
    WHERE feature_class = 'Lake' \
    AND feature_name LIKE 'M%' \
    ORDER BY feature_name;")
```

次の Spark SQLクエリは、1 マイルを超える 3 つ以上の特徴を持つすべての状態のリストを返します。

```
val dataFrame = sparkSession.dql("SELECT state_alpha, feature_class, COUNT(*) \
    FROM ddb_features \
    WHERE elev_in_ft > 5280 \
```

```
    GROUP by state_alpha, feature_class \
    HAVING COUNT(*) >= 3 \
    ORDER BY state_alpha, feature_class;")
```

Example – Spark を使用して DynamoDB テーブルをクエリする MapReduce API

次の MapReduce クエリは、すべての特徴量タイプのリストをアルファベット順に返します。

```
val df = sc.hadoopRDD(jobConf, classOf[DynamoDBInputFormat], classOf[Text],
 classOf[DynamoDBItemWritable])
    .map(pair => (pair._1, pair._2.getItem))
    .map(pair => pair._2.get("feature_class").getS)
    .distinct()
    .sortBy(value => value)
    .toDF("feature_class")
```

次の MapReduce クエリは、M で始まるすべてのレイクのリストを返します。

```
val df = sc.hadoopRDD(jobConf, classOf[DynamoDBInputFormat], classOf[Text],
 classOf[DynamoDBItemWritable])
    .map(pair => (pair._1, pair._2.getItem))
    .filter(pair => "Lake".equals(pair._2.get("feature_class").getS))
    .filter(pair => pair._2.get("feature_name").getS.startsWith("M"))
    .map(pair => (pair._2.get("feature_name").getS,
 pair._2.get("state_alpha").getS))
    .sortBy(_._1)
    .toDF("feature_name", "state_alpha")
```

次の MapReduce クエリは、1 マイルを超える少なくとも 3 つの特徴を持つすべての状態のリストを返します。

```
val df = sc.hadoopRDD(jobConf, classOf[DynamoDBInputFormat], classOf[Text],
 classOf[DynamoDBItemWritable])
    .map(pair => pair._2.getItem)
    .filter(pair => pair.get("elev_in_ft").getN != null)
    .filter(pair => Integer.parseInt(pair.get("elev_in_ft").getN) > 5280)
    .groupBy(pair => (pair.get("state_alpha").getS, pair.get("feature_class").getS))
    .filter(pair => pair._2.size >= 3)
    .map(pair => (pair._1._1, pair._1._2, pair._2.size))
    .sortBy(pair => (pair._1, pair._2))
    .toDF("state_alpha", "feature_class", "count")
```

Hive

前のステップで作成した DynamoDB テーブルからデータをクエリするには、DynamoDB テーブル のデータをクエリする」の手順に従います。

クロスアカウントアクセスのセットアップ

EMR Serverless のクロスアカウントアクセスを設定するには、次の手順を実行します。この例では、 AccountAは Amazon EMR Serverless アプリケーションを作成したアカウントであり、AccountBは Amazon DynamoDB があるアカウントです。

1. で DynamoDB テーブルを作成しますAccountB。詳細については、「ステップ 1: テーブルを作成する」を参照してください。
2. DynamoDB テーブルAccountBにアクセスできるCross-Account-Role-BIAMロールを に作成します。

   a. 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/。

   b. ロール を選択し、 という名前の新しいロールを作成しますCross-Account-Role-B。IAM ロールの作成方法の詳細については、「 ユーザーガイド」のIAM「ロールの作成」を参照してください。

   c. クロスアカウント DynamoDB テーブルへのアクセス許可を付与するIAMポリシーを作成します。次に、IAMポリシーを にアタッチしますCross-Account-Role-B。

      以下は、DynamoDB テーブル へのアクセスを許可するポリシーですCrossAccountTable。

      ```
      {"Version": "2012-10-17",
          "Statement": [
              {"Effect": "Allow",
                  "Action": "dynamodb:*",
                  "Resource": "arn:aws:dynamodb:region:AccountB:table/
      CrossAccountTable"
              }
          ]
      }
      ```

   d. Cross-Account-Role-B ロールの信頼関係を編集します。

ロールの信頼関係を設定するには、ステップ 2: Cross-Account-Role-B で作成したロールのIAMコンソールで信頼関係タブを選択します。

信頼関係の編集 を選択し、次のポリシードキュメントを追加します。このドキュメントでは、 Job-Execution-Role-A がこのCross-Account-Role-Bロール AccountAを引き受けることを許可します。

```
{"Version": "2012-10-17",
  "Statement": [
    {"Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::AccountA:role/Job-Execution-Role-A"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

e. Job-Execution-Role-A を引き受ける- STS Assume roleアクセス許可AccountAを持つ を付与しますCross-Account-Role-B。

のIAMコンソールで AWS アカウント AccountA、 を選択しますJob-Execution-Role-A。次のポリシーステートメントを Job-Execution-Role-A に追加して、Cross-Account-Role-B ロールに対して AssumeRole アクションを許可します。

```
{"Version": "2012-10-17",
    "Statement": [
        {"Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::AccountB:role/Cross-Account-Role-B"
        }
    ]
}
```

f. コアサイト分類com.amazonaws.emr.AssumeRoleAWSCredentialsProviderで 値をとしてdynamodb.customAWSCredentialsProviderプロパティを設定します。ARN 環境変数を の値ASSUME_ROLE_CREDENTIALS_ROLE_ARNで設定しますCross-Account-Role-B。

3. を使用して Spark または Hive ジョブを実行しますJob-Execution-Role-A。

考慮事項

Apache Spark で DynamoDB コネクタを使用する場合の考慮事項

- Spark SQLは、ストレージハンドラーオプションを使用した Hive テーブルの作成をサポートしていません。詳細については、Apache Spark ドキュメントの「Hive テーブルのストレージ形式の指定」を参照してください。
- Spark SQLは、ストレージハンドラーによる `STORED BY` オペレーションをサポートしていません。外部 Hive テーブルを介して DynamoDB テーブルを操作する場合は、まず Hive を使用してテーブルを作成します。
- クエリを DynamoDB クエリに変換するには、DynamoDB コネクタは述語プッシュダウン を使用します。述語プッシュダウンは、DynamoDB テーブルのパーティションキーにマッピングされた列によってデータをフィルタリングします。述語プッシュダウンは、Spark でコネクタを使用する場合のみ動作しSQL、 では動作しません MapReduce API。

Apache Hive で DynamoDB コネクタを使用する場合の考慮事項

マッパーの最大数の調整

- `SELECT` クエリを使用して DynamoDB にマッピングされる外部 Hive テーブルからデータを読み取る場合、EMRServerless のマップタスクの数は、DynamoDB テーブルに設定された合計読み取りスループットとして計算され、マップタスクあたりのスループットで割られます。マップタスクあたりのデフォルトのスループットは 100 です。
- Hive ジョブは、DynamoDB 用に設定された読み取りスループットに応じて、EMRサーバーレスアプリケーションごとに設定されたコンテナの最大数を超えるマップタスクの数を使用できます。また、長時間実行される Hive クエリは、DynamoDB テーブルのプロビジョニングされた読み取り容量をすべて消費できます。これは他のユーザーに悪影響を及ぼします。
- `dynamodb.max.map.tasks` プロパティを使用して、マップタスクの上限を設定できます。このプロパティを使用して、タスクコンテナのサイズに基づいて、各マップタスクによって読み取られるデータの量を調整することもできます。
- `dynamodb.max.map.tasks`プロパティは、Hive クエリレベル、または start-job-run コマンドの`hive-site`分類で設定できます。この値は、1 以上にする必要があります。Hive がクエリを処理すると、結果の Hive ジョブは、DynamoDB テーブルから読み取る`dynamodb.max.map.tasks`ときに の値を超えることはありません。

タスクあたりの書き込みスループットの調整

- EMR Serverless のタスクあたりの書き込みスループットは、DynamoDB テーブル用に設定された合計書き込みスループットを `mapreduce.job.maps`プロパティの値で割って計算されます。Hive の場合、このプロパティのデフォルト値は 2 です。したがって、Hive ジョブの最終ステージの最初の 2 つのタスクは、すべての書き込みスループット を消費できます。これにより、同じジョブまたは他のジョブ内の他のタスクの書き込みがスロットリングされます。
- 書き込みスロットリングを回避するには、最終ステージのタスク数またはタスクごとに割り当てる書き込みスループットに基づいて`mapreduce.job.maps`プロパティの値を設定できます。このプロパティを EMR Serverless の start-job-run コマンドの`mapred-site`分類に設定します。

セキュリティ

でのクラウドセキュリティ AWS が最優先事項です。として AWS のお客様は、セキュリティを最も重視する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からメリットを得られます。

セキュリティは、 間で共有される責任です。 AWS とユーザー。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 は、 が実行するインフラストラクチャを保護する責任があります。 AWS の サービス AWS クラウド。 AWS は、安全に使用できる サービスも提供します。サードパーティーの監査者は、 の一環として当社のセキュリティの有効性を定期的にテストおよび検証します。 AWS コンプライアンスプログラム 。Amazon EMR Serverless に適用されるコンプライアンスプログラムの詳細については、「」を参照してください。 AWS コンプライアンスプログラム の対象となる のサービス。
- クラウド内のセキュリティ — お客様の責任は によって決まります。 AWS 使用する サービス。また、お客様は、お客様のデータの機密性、企業の要件、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、Amazon EMR Serverless を使用する際の責任共有モデルの適用方法を理解するのに役立ちます。この章のトピックでは、Amazon EMR Serverless を設定し、他の AWS セキュリティとコンプライアンスの目的を満たす のサービス。

トピック

- Amazon EMR Serverless のセキュリティのベストプラクティス
- データ保護
- Amazon EMR Serverless での ID とアクセスの管理 (IAM)
- AWS Lake Formation でのEMRサーバーレスを使用したきめ細かなアクセスコントロール
- ワーカー間の暗号化
- EMR Serverless によるデータ保護のための Secrets Manager
- EMR Serverless での Amazon S3 Access Grants の使用
- を使用した Amazon EMR Serverless API呼び出しのログ記録 AWS CloudTrail
- Amazon EMR Serverless のコンプライアンス検証
- Amazon EMR Serverless の耐障害性

- Amazon EMR Serverless のインフラストラクチャセキュリティ
- Amazon EMR Serverless での設定と脆弱性の分析

Amazon EMR Serverless のセキュリティのベストプラクティス

Amazon EMR Serverless 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いくつかのセキュリティ機能が用意されています。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必ずしも適切または十分でない可能性があるので、処方箋ではなく、あくまで有用な考慮事項とお考えください。

最小特権の原則を適用する

EMR Serverless は、実行IAMロールなどのロールを使用するアプリケーションにきめ細かいアクセスポリシーを提供します。実行ロールには、アプリケーションのカバーやログ送信先へのアクセスなど、ジョブに必要な最小限の特権セットのみを付与することをお勧めします。また、定期的に、およびアプリケーションコードに変更があったときに、ジョブの許可を監査することをお勧めします。

信頼できないアプリケーションコードを分離する

EMR サーバーレスは、異なるEMRサーバーレスアプリケーションに属するジョブ間で完全なネットワーク分離を作成します。ジョブレベルの分離が必要な場合は、ジョブを異なるEMRサーバーレスアプリケーションに分離することを検討してください。

ロールベースのアクセスコントロール (RBAC) アクセス許可

管理者は、EMRサーバーレスアプリケーションのロールベースのアクセスコントロール (RBAC) のアクセス許可を厳密に制御する必要があります。

データ保護

AWS 責任共有モデルは、Amazon EMR Serverless で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 AWS クラウド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。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。このコンテンツには、使用する AWS サービスのセキュリティ設定および管理タスクが含まれます。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、「 責任 AWS 共有モデル」とGDPR AWS 「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、 AWS アカウントの認証情報を保護し、 AWS Identity and Access Management () を使用して個々のアカウント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許可のみを各ユーザーに付与できます。また、次の方法でデータを保護することをお勧めします。

- 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 TLS 以降をお勧めします。
- 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。
- AWS 暗号化ソリューションと、サービス内のすべての AWS デフォルトのセキュリティコントロールを使用します。
- Amazon Macie などのアドバンストマネージドセキュリティサービスを使用します。これは、Amazon S3 に保存されている個人データの検出と保護を支援します。
- Amazon EMR Serverless 暗号化オプションを使用して、保管中および転送中のデータを暗号化します。
- 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 140-2 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) 140-2」を参照してください。

顧客のアカウント番号などの機密の識別情報は、[名前] フィールドなどの自由形式の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 Amazon EMR Serverless やその他の AWS サービスを使用する場合も含まれます AWS SDKs。Amazon EMR Serverless やその他のサービスに入力したデータは、診断ログに含めるために取得される可能性があります。URL を外部サーバーに提供する場合、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 に含めないでください。

保管中の暗号化

データの暗号化は、承認されていないユーザーがクラスターおよび関連するデータストレージシステムのデータを読み取れないようにするのに役立ちます。このデータには、保管中のデータと呼ばれる、永続的なメディアに保存されているデータや、転送中のデータと呼ばれる、ネットワークを介した転送の間に傍受される可能性のあるデータが含まれます。

データの暗号化には、キーと証明書が必要です。によって管理されるキー、Amazon S3 によって管理されるキー AWS Key Management Service、提供するカスタムプロバイダーのキーと証明書

など、いくつかのオプションから選択できます。をキープロバイダー AWS KMS として使用する場合、暗号化キーのストレージと使用には料金が適用されます。詳細については、AWS KMS の料金を参照してください。

暗号化オプションを指定する前に、使用するキーと証明書の管理システムを決定します。次に、暗号化設定の一部として指定するカスタムプロバイダーのキーと証明書を作成します。

Amazon S3 EMRFSのデータの保存時の暗号化

各 EMR Serverless アプリケーションは、 EMRFS (EMR ファイルシステム) を含む特定のリリースバージョンを使用します。Amazon S3 暗号化は、Amazon S3 との間で読み書きされるEMRファイルシステム (EMRFS) オブジェクトと連携します。保管中の暗号化を有効にする場合、Amazon S3 サーバー側の暗号化 (SSE) またはクライアント側の暗号化 (CSE) をデフォルトの暗号化モードとして指定できます。オプションで、[Per bucket encryption overrides (バケットごとの暗号化オーバーライド)] を使用して、バケットごとに異なる暗号化方法を指定できます。Amazon S3 暗号化が有効になっているかどうかにかかわらず、Transport Layer Security (TLS) はEMRクラスターノードとAmazon S3 間で転送中のEMRFSオブジェクトを暗号化します。カスタマーマネージドキーCSEでAmazon S3 を使用する場合、EMRサーバーレスアプリケーションでジョブを実行するために使用される実行ロールは、キーにアクセスできる必要があります。Amazon S3 暗号化の詳細については、Amazon Simple Storage Service デベロッパーガイドの「暗号化を使用したデータの保護」を参照してください。

> **Note**
>
> を使用する場合 AWS KMS、暗号化キーのストレージと使用には料金が適用されます。詳細については、AWS KMS の料金を参照してください。

Amazon S3 のサーバー側の暗号化

Amazon S3 のサーバー側の暗号化をセットアップすると、Amazon S3 はデータをディスクに書き込むときにオブジェクトレベルで暗号化し、アクセスするときに復号します。の詳細についてはSSE、「Amazon Simple Storage Service デベロッパーガイド」の「サーバー側の暗号化を使用したデータの保護」を参照してください。

Amazon EMR Serverless SSEで を指定すると、2 つの異なるキー管理システムから選択できます。

- SSE-S3 - Amazon S3 はキーを管理します。EMR Serverless では追加のセットアップは必要ありません。

- SSE-KMS - EMR Serverless に適したポリシーで をセットアップ AWS KMS key するには、 を使用します。EMR Serverless では追加のセットアップは必要ありません。

Amazon S3 に書き込むデータに AWS KMS 暗号化を使用するには、 StartJobRun を使用するときに 2 つのオプションがありますAPI。 Amazon S3 Amazon S3 に書き込むすべてのデータの暗号化を有効にするか、特定のバケットに書き込むデータの暗号化を有効にすることができます。の詳細についてはAPI、EMR「サーバーレスAPIリファレンスStartJobRun」を参照してください。

Amazon S3 に書き込むすべてのデータの AWS KMS 暗号化を有効にするには、 を呼び出すときに次のコマンドを使用しますStartJobRunAPI。

```
--conf spark.hadoop.fs.s3.enableServerSideEncryption=true
--conf spark.hadoop.fs.s3.serverSideEncryption.kms.keyId=<kms_id>
```

特定のバケットに書き込むデータの AWS KMS 暗号化を有効にするには、 を呼び出すときに次のコマンドを使用しますStartJobRunAPI。

```
--conf spark.hadoop.fs.s3.bucket.<amzn-s3-demo-bucket1>.enableServerSideEncryption=true
--conf spark.hadoop.fs.s3.bucket.<amzn-s3-demo-
bucket1>.serverSideEncryption.kms.keyId=<kms-id>
```

SSE と は、お客様が用意したキー (SSE-C) では EMR Serverless では使用できません。

Amazon S3 クライアント側の暗号化

Amazon S3 クライアント側の暗号化では、Amazon S3 の暗号化と復号は、すべての Amazon EMR リリースで利用可能なEMRFSクライアントで行われます。オブジェクトは Amazon S3 にアップロードされる前に暗号化され、ダウンロード後に復号化されます。指定するプロバイダーが、クライアントが使用する暗号化キーを提供します。クライアントは、 AWS KMS （ CSE-KMS) が提供するキー、またはクライアント側のルートキー (CSE-C) を提供するカスタム Java クラスを使用できます。暗号化の詳細は、指定されたプロバイダーと復号化または暗号化されるオブジェクトのメタデータに応じて、CSE-KMS と CSE-C でわずかに異なります。カスタマーマネージドキーCSEで Amazon S3 を使用する場合、EMRサーバーレスアプリケーションでジョブを実行するために使用される実行ロールは、キーにアクセスできる必要があります。KMS 追加料金が発生する場合があります。これらの違いの詳細については、Amazon Simple Storage Service デベロッパーガイドの「クライアント側の暗号化を使用したデータの保護」を参照してください。

ローカルディスク暗号化

エフェメラルストレージに保存されているデータは、業界標準の AES-256 暗号化アルゴリズムを使用して、サービス所有のキーで暗号化されます。

キー管理

KMS キーを自動的にローテーションKMSするように を設定できます。これにより、古いキーを無期限に保存しながら、年に一度、キーをローテーションして、データを復号することができます。詳細については、「カスタマーマスターキーのローテーション」を参照してください。

転送中の暗号化

Amazon EMR Serverless では、以下のアプリケーション固有の暗号化機能を使用できます。

- Spark
  - デフォルトでは、Spark ドライバーとエグゼキュター間の通信は認証され、内部的に行われます。RPC ドライバーとエグゼキュター間の通信は暗号化されます。
- [Hive]
  - AWS Glue メタストアとEMRサーバーレスアプリケーション間の通信は、 を介して行われます TLS。

Amazon S3 バケットIAMポリシーの aws:SecureTransport condition を使用してHTTPS、 (TLS) 経由の暗号化された接続のみを許可する必要があります。

Amazon EMR Serverless での ID とアクセスの管理 (IAM)

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、Amazon EMR Serverless リソースの使用を認証 (サインイン) および承認 (アクセス許可を持つ) できるユーザーを制御します。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

- EMR Serverless の での仕組み IAM
- EMR Serverless のサービスにリンクされたロールの使用
- Amazon EMR Serverless のジョブランタイムロール
- EMR Serverless のユーザーアクセスポリシーの例
- タグベースのアクセスコントロールのポリシー
- EMR Serverless のアイデンティティベースのポリシーの例
- AWS マネージドポリシーへの Amazon EMR Serverless 更新
- Amazon EMR Serverless の ID とアクセス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、Amazon EMR Serverless で行う作業によって異なります。

サービスユーザー – Amazon EMR Serverless サービス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。Amazon EMR Serverless の機能をさらに使用して作業を行うと、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。Amazon EMR Serverless の機能にアクセスできない場合は、「」を参照してくださいAmazon EMR Serverless の ID とアクセスのトラブルシューティング。

サービス管理者 – 社内の Amazon EMR Serverless リソースを担当している場合は、Amazon EMR Serverless へのフルアクセスが許可されている可能性があります。サービスユーザーがどの Amazon EMR Serverless 機能やリソースにアクセスする必要があるかを判断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。会社が Amazon EMR Serverless IAMで を使用する方法の詳細については、「」を参照してくださいAmazon EMR Serverless での ID とアクセスの管理 (IAM) 。

IAM 管理者 – IAM管理者の場合は、Amazon EMR Serverless へのアクセスを管理するポリシーの作成方法の詳細を知りたい場合があります。で使用できる Amazon EMR Serverless 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IAM、「」を参照してくださいEMR Serverless のアイデンティティベースのポリシーの例。

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) 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を自分で署名する方法の詳細については、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。

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「多要素認証の使用」および「 ユーザーガイド」の「多要素認証の使用 (MFA ） AWS」を参照してください。 IAM

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、アカウント内のすべての および リソースへの AWS のサービス 完全なアクセス権を持つ 1 つのサインイン ID から始めます。この ID は AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインしてアクセスします。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、IAM「 ユーザーガイド」の「ルートユーザーの認証情報を必要とするタスク」を参照してください。

フェデレーティッドアイデンティティ

ベストプラクティスとして、 では、管理者アクセスを必要とするユーザーを含む人間のユーザーに、一時的な認証情報を使用してアクセスするために ID プロバイダーとのフェデレーション AWS のサービス の使用を要求します。

フェデレーティッド ID は、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダー、AWS Directory Service、アイデンティティセンターディレクトリ、または ID ソースを通じて提供された認証情報 AWS のサービス を使用してアクセスするユーザーです。フェデレーティッド ID が にアクセスすると AWS アカウント、それらはロールを引き受け、ロールは一時的な認証情報を提供します。

アクセスを一元管理する場合は、 AWS IAM Identity Centerを使用することをお勧めします。IAM Identity Center でユーザーとグループを作成するか、独自の ID ソース内のユーザーとグループのセットに接続して同期し、すべての AWS アカウント とアプリケーションで使用できます。IAM Identity Center の詳細については、AWS IAM Identity Center 「 ユーザーガイド」のIAM「Identity Center とは」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、 IAM ユーザーガイドの （ロールではなく) IAM ユーザーを作成するタイミングを参照してください。

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。IAM ロール を切り替える AWS Management Console ことで、 でロールを一時的に引き受けることができます。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。 AWS ロールをEC2インスタンスに割り当て、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。

IAM ロールとIAMユーザーのどちらを使用するかについては、IAM「 ユーザーガイド」の「 (ユーザーではなく) IAMロールを作成するタイミング」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 アイデンティティ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IAM「ポリシーの作成」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されません。

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) をアカウントのいずれか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

EMR Serverless の での仕組み IAM

IAM を使用して Amazon EMR Serverless へのアクセスを管理する前に、Amazon EMR Serverless で使用できるIAM機能について説明します。

IAM EMR Serverless で使用できる機能

IAM 機能	Amazon EMR Serverless のサポート
アイデンティティベースのポリシー	あり
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

EMR Serverless やその他の 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。

EMR Serverless 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、「 ユーザーガイド」のIAM「ポリシーの作成」を参照してください。 IAM

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。

EMR Serverless のアイデンティティベースのポリシーの例

Amazon EMR Serverless 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、「」を参照してくださいEMR Serverless のアイデンティティベースのポリシーの例。

EMR Serverless 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、「 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。 IAM

EMR Serverless 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。

EMR サーバーレスアクションのリストを確認するには、「サービス認証リファレンス」の「Amazon EMR Serverless 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。

EMR Serverless 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。

```
emr-serverless
```

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": [
      "emr-serverless:action1",
      "emr-serverless:action2"
         ]
```

Amazon EMR Serverless 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、「」を参照してくださいEMR Serverless のアイデンティティベースのポリシーの例。

EMR Serverless 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": "*"
```

Amazon EMR Serverless リソースタイプとその 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「Amazon EMR Serverless 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、「Amazon EMR Serverless 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。

Amazon EMR Serverless 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、「」を参照してくださいEMR Serverless のアイデンティティベースのポリシーの例。

EMR Serverless のポリシー条件キー

サービス固有のポリシー条件キーのサポート	不可

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

Amazon EMR Serverless 条件キーのリストを表示し、条件キーを使用できるアクションとリソースを確認するには、「サービス認証リファレンス」の「Amazon EMR Serverless 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。

すべての Amazon EC2アクションは、 aws:RequestedRegion および ec2:Region条件キーをサポートしています。詳細については、 「例: 特定のリージョンへのアクセスの制限」を参照してください。

EMR Serverless 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されません。

EMR Serverless 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 (ポリシーのタグ)	可能

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。タグは、IAMエンティティ (ユーザーまたは

ロール) および多くの AWS リソースにアタッチできます。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場合に、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、 IAM ユーザーガイドの「 とはABAC」を参照してください。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。

EMR Serverless 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

一部の AWS のサービス は、一時的な認証情報を使用してサインインすると機能しません。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IAM方法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「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。

EMR Serverless 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

ユーザーIAM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を AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストリクエストと組み合わせて使用します。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。

EMR Serverless のサービスロール

サービスロールのサポート	不可

EMR Serverless 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	可能

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。

EMR Serverless のサービスにリンクされたロールの使用

Amazon EMR Serverless が を使用する AWS Identity and Access Management （ IAM ） 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、EMRサーバーレスに直接リンクされた一意のタイプのIAMロールです。サービスにリンクされたロールは EMR Serverless によって事前定義されており、サービスが他の 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ています。AWS ユーザーに代わって のサービス。

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、EMRサーバーレスの設定が簡単になります。EMR サーバーレスは、サービスにリンクされたロールのアクセス許可を定義します。特に定義されている場合を除き、EMRサーバーレスのみがそのロールを引き受けることができます。定義されたアクセス許可には、信頼ポリシーとアクセス許可ポリシーが含まれ、そのアクセス許可ポリシーを他のIAMエンティティにアタッチすることはできません。

サービスリンクロールは、まずその関連リソースを削除しなければ削除できません。これにより、リソースにアクセスするためのアクセス許可を誤って削除することがないため、EMRサーバーレス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については、「」を参照してください。AWS と連携する のサービスIAMで、サービスにリンクされたロール列に「はい」がある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。

EMR Serverless のサービスにリンクされたロールのアクセス許可

EMR Serverless は、 という名前のサービスにリンクされたロールAWSServiceRoleForAmazonEMRServerlessを使用して、 を呼び出すことを可能にします。 AWS APIs ユーザーに代わって。

AWSServiceRoleForAmazonEMRServerless サービスにリンクされたロールは、次のサービスを信頼してロールを引き受けます。

- `ops.emr-serverless.amazonaws.com`

という名前のロールアクセス許可ポリシーAmazonEMRServerlessServiceRolePolicyにより、EMRサーバーレス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。

> **Note**
>
> 管理ポリシーの内容は変更されるため、ここに示すポリシーは古くなっている可能性があります。で最も多くの up-to-date ポリシー A mazonEMRServerlessServiceRolePolicyを表示する AWS Management Console.

- アクション: `ec2:CreateNetworkInterface`
- アクション: `ec2:DeleteNetworkInterface`
- アクション: `ec2:DescribeNetworkInterfaces`
- アクション: `ec2:DescribeSecurityGroups`
- アクション: `ec2:DescribeSubnets`
- アクション: `ec2:DescribeVpcs`

- アクション: ec2:DescribeDhcpOptions
- アクション: ec2:DescribeRouteTables
- アクション: cloudwatch:PutMetricData

完全なAmazonEMRServerlessServiceRolePolicyポリシーは次のとおりです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EC2PolicyStatement",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeRouteTables"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchPolicyStatement",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": [
                        "AWS/EMRServerless",
                        "AWS/Usage"
                    ]
                }
            }
        }
```

```
    ]
}
```

サーバーEMRレスプリンシパルがこのロールを引き受けることができるように、次の信頼ポリシーがこのロールにアタッチされます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "ops.emr-serverless.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

IAM エンティティ (ユーザー、グループ、ロールなど) がサービスにリンクされたロールを作成、編集、または削除できるようにするには、アクセス許可を設定する必要があります。詳細については、「 ユーザーガイド」の「サービスにリンクされたロールのアクセス許可IAM」を参照してください。

EMR Serverless のサービスにリンクされたロールの作成

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。で新しいEMRサーバーレスアプリケーションを作成する場合 AWS Management Console （ EMRStudio を使用 ）、 AWS CLI、または AWS API、EMRサーバーレスはサービスにリンクされたロールを作成します。IAM エンティティ (ユーザー、グループ、ロールなど) がサービスにリンクされたロールを作成、編集、または削除できるようにするには、アクセス許可を設定する必要があります。

を使用して AWSServiceRoleForAmazonEMRServerless サービスにリンクされたロールを作成するには IAM

サービスにリンクされたロールを作成する必要があるIAMエンティティのアクセス許可ポリシーに、次のステートメントを追加します。

```
{
```

```
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/ops.emr-serverless.amazonaws.com/
AWSServiceRoleForAmazonEMRServerless*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "ops.emr-
serverless.amazonaws.com"}}
}
```

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。新しい EMR Serverless アプリケーションを作成すると、EMRServerless によってサービスにリンクされたロールが再度作成されます。

IAM コンソールを使用して、EMRサーバーレスユースケースでサービスにリンクされたロールを作成することもできます。左 AWS CLI または AWS API、サービス名を使用してops.emr-serverless.amazonaws.comサービスにリンクされたロールを作成します。詳細については、「 ユーザーガイド」の「サービスにリンクされたロールの作成IAM」を参照してください。このサービスリンクロールを削除しても、同じ方法でロールを再作成できます。

EMR Serverless のサービスにリンクされたロールの編集

EMR サーバーレスでは、さまざまなエンティティがロールを参照する可能性があるため、AWSServiceRoleForAmazonEMRServerless サービスにリンクされたロールを編集することはできません。を編集することはできません AWS EMR Serverless サービスにリンクされたロールが使用する 所有IAMポリシー。サーバーレスに必要なすべてのアクセス許可が含まれているためですEMR。ただし、 を使用してロールの説明を編集することはできますIAM。

を使用して AWSServiceRoleForAmazonEMRServerless サービスにリンクされたロールの説明を編集するには IAM

サービスにリンクされたロールの説明を編集する必要があるIAMエンティティのアクセス許可ポリシーに、次のステートメントを追加します。

```
{
    "Effect": "Allow",
    "Action": [
        "iam: UpdateRoleDescription"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/ops.emr-serverless.amazonaws.com/
AWSServiceRoleForAmazonEMRServerless*",
```

```
    "Condition": {"StringLike": {"iam:AWSServiceName": "ops.emr-
serverless.amazonaws.com"}}
}
```

詳細については、「 IAMユーザーガイド」の「サービスにリンクされたロールの編集」を参照してください。

EMR Serverless のサービスにリンクされたロールの削除

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。これは、アクティブにモニタリングまたは保守されていない未使用のエンティティが存在しないためです。ただし、サービスにリンクされたロールを削除する前に、すべてのリージョンのすべてのEMRサーバーレスアプリケーションを削除する必要があります。

> **Note**
>
> ロールに関連付けられたリソースを削除しようとしたときに EMR Serverless サービスがロールを使用している場合、削除が失敗する可能性があります。失敗した場合は、数分待ってから操作を再試行してください。

を使用して AWSServiceRoleForAmazonEMRServerless サービスにリンクされたロールを削除するには IAM

サービスにリンクされたロールを削除する必要があるIAMエンティティのアクセス許可ポリシーに、次のステートメントを追加します。

```
{
    "Effect": "Allow",
    "Action": [
        "iam:DeleteServiceLinkedRole",
        "iam:GetServiceLinkedRoleDeletionStatus"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/ops.emr-serverless.amazonaws.com/
AWSServiceRoleForAmazonEMRServerless*",
    "Condition": {"StringLike": {"iam:AWSServiceName": "ops.emr-
serverless.amazonaws.com"}}
}
```

を使用してサービスにリンクされたロールを手動で削除するには IAM

IAM コンソール、 AWS CLI、または AWS API AWSServiceRoleForAmazonEMRServerless サービスにリンクされたロールを削除するには、 にします。詳細については、「 ユーザーガイド」の「サービスにリンクされたロールの削除IAM」を参照してください。

EMR Serverless サービスにリンクされたロールでサポートされているリージョン

EMR Serverless 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、「」を参照してくださいAWS リージョンとエンドポイント 。

Amazon EMR Serverless のジョブランタイムロール

EMR サーバーレスジョブの実行が、ユーザーに代わって他の サービスを呼び出すときに引き受けることができるIAMロールアクセス許可を指定できます。これには、データソース、ターゲット、Amazon Redshift クラスターや DynamoDB テーブルなどの他の AWS リソースに対する Amazon S3 へのアクセスが含まれます。ロールの作成方法の詳細については、「」を参照してくださいジョブランタイムロールを作成する。

サンプルランタイムポリシー

ジョブランタイムロールには、次のようなランタイムポリシーをアタッチできます。次のジョブランタイムポリシーでは、 を許可します。

- EMR サンプルを使用した Amazon S3 バケットへの読み取りアクセス。
- S3 バケットへのフルアクセス。
- AWS Glue Data Catalog への作成および読み取りアクセス。

DynamoDB などの他の AWS リソースへのアクセスを追加するには、ランタイムロールを作成するときにポリシーにそれらのアクセス許可を含める必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadAccessForEMRSamples",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
```

```
            "Resource": [
                "arn:aws:s3:::*.elasticmapreduce",
                "arn:aws:s3:::*.elasticmapreduce/*"
            ]
        },
        {
            "Sid": "FullAccessToS3Bucket",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket",
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        },
        {
            "Sid": "GlueCreateAndReadDataCatalog",
            "Effect": "Allow",
            "Action": [
                "glue:GetDatabase",
                "glue:CreateDatabase",
                "glue:GetDataBases",
                "glue:CreateTable",
                "glue:GetTable",
                "glue:UpdateTable",
                "glue:DeleteTable",
                "glue:GetTables",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:CreatePartition",
                "glue:BatchCreatePartition",
                "glue:GetUserDefinedFunctions"
            ],
            "Resource": ["*"]
        }
    ]
}
```

ロール権限を渡す

アクセスIAM許可ポリシーをユーザーのロールにアタッチして、ユーザーが承認されたロールのみを渡すことを許可できます。これにより、管理者は特定のジョブランタイムロールをEMRサーバーレスジョブに渡すことができるユーザーを制御できます。アクセス許可の設定の詳細については、AWS「サービス にロールを渡すアクセス許可をユーザーに付与する」を参照してください。

以下は、ジョブランタイムロールを EMR Serverless サービスプリンシパルに渡すことを許可するポリシーの例です。

```
{
     "Effect": "Allow",
     "Action": "iam:PassRole",
     "Resource": "arn:aws:iam::1234567890:role/JobRuntimeRoleForEMRServerless",
        "Condition": {
                "StringLike": {
                    "iam:PassedToService": "emr-serverless.amazonaws.com"
                }
            }
}
```

EMR Serverless のユーザーアクセスポリシーの例

EMR Serverless アプリケーションを操作するときに各ユーザーに実行するアクションに応じて、ユーザーのきめ細かなポリシーを設定できます。次のポリシーは、ユーザーに適したアクセス許可を設定するのに役立つ可能性のある例です。このセクションでは、EMRサーバーレスポリシーのみに焦点を当てます。EMR Studio ユーザーポリシーのサンプルについては、EMR「Studio ユーザーアクセス許可の設定」を参照してください。IAM ユーザー (プリンシパル) にポリシーをアタッチする方法については、IAM「 ユーザーガイド」のIAM「ポリシーの管理」を参照してください。

パワーユーザーポリシー

EMR Serverless に必要なすべてのアクションを付与するには、AmazonEMRServerlessFullAccessポリシーを作成して、必要なIAMユーザー、ロール、またはグループにアタッチします。

以下は、パワーユーザーが EMR Serverless アプリケーションを作成および変更したり、ジョブの送信やデバッグなどのアクションを実行したりできるようにするサンプルポリシーです。EMR Serverless が他のサービスに必要とするすべてのアクションが表示されます。

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "EMRServerlessActions",
            "Effect": "Allow",
            "Action": [
                "emr-serverless:CreateApplication",
                "emr-serverless:UpdateApplication",
                "emr-serverless:DeleteApplication",
                "emr-serverless:ListApplications",
                "emr-serverless:GetApplication",
                "emr-serverless:StartApplication",
                "emr-serverless:StopApplication",
                "emr-serverless:StartJobRun",
                "emr-serverless:CancelJobRun",
                "emr-serverless:ListJobRuns",
                "emr-serverless:GetJobRun"
            ],
            "Resource": "*"
        }
    ]
}
```

へのネットワーク接続を有効にするとVPC、EMRサーバーレスアプリケーションはVPCリソースと通信するための Amazon EC2Elastic Network Interface (ENIs) を作成します。次のポリシーは、新しい EC2ENIsが EMR Serverless アプリケーションのコンテキストでのみ作成されることを保証します。

> **Note**
>
> このポリシーを設定して、EMRサーバーレスアプリケーションを起動するコンテキスト EC2ENIs以外でユーザーが を作成できないようにすることを強くお勧めします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowEC2ENICreationWithEMRTags",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface"
            ],
```

```
            "Resource": [
                "arn:aws:ec2:*:*:network-interface/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:CalledViaLast": "ops.emr-serverless.amazonaws.com"
                }
            }
        }
}
```

特定のサブネットへのEMRサーバーレスアクセスを制限する場合は、各サブネットにタグ条件を付けます。このIAMポリシーにより、EMRサーバーレスアプリケーションは許可されたサブネットEC2ENIs内でのみ作成できます。

```
{
    "Sid": "AllowEC2ENICreationInSubnetAndSecurityGroupWithEMRTags",
    "Effect": "Allow",
    "Action": [
        "ec2:CreateNetworkInterface"
    ],
    "Resource": [
        "arn:aws:ec2:*:*:subnet/*",
        "arn:aws:ec2:*:*:security-group/*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:ResourceTag/KEY": "VALUE"
        }
    }
}
```

> **Important**
>
> 管理者またはパワーユーザーが最初のアプリケーションを作成している場合は、EMRサーバーレスサービスにリンクされたロールを作成できるようにアクセス許可ポリシーを設定する必要があります。詳細については、「EMR Serverless のサービスにリンクされたロールの使用」を参照してください。

次のIAMポリシーでは、アカウントのEMRサーバーレスサービスリンクロールを作成できます。

```
{
   "Sid":"AllowEMRServerlessServiceLinkedRoleCreation",
   "Effect":"Allow",
   "Action":"iam:CreateServiceLinkedRole",
   "Resource":"arn:aws:iam::account-id:role/aws-service-role/ops.emr-
serverless.amazonaws.com/AWSServiceRoleForAmazonEMRServerless"
}
```

データエンジニアポリシー

以下は、EMRサーバーレスアプリケーションに対する読み取り専用アクセス許可と、ジョブの送信とデバッグをユーザーに許可するサンプルポリシーです。このポリシーは明示的にアクションを拒否しないため、別のポリシーステートメントが指定したアクションへのアクセス許可に使用される場合があることに注意してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EMRServerlessActions",
            "Effect": "Allow",
            "Action": [
                "emr-serverless:ListApplications",
                "emr-serverless:GetApplication",
                "emr-serverless:StartApplication",
                "emr-serverless:StartJobRun",
                "emr-serverless:CancelJobRun",
                "emr-serverless:ListJobRuns",
                "emr-serverless:GetJobRun"
            ],
            "Resource": "*"
        }
    ]
}
```

アクセスコントロールにタグを使用する

きめ細かなアクセスコントロールにはタグ条件を使用できます。例えば、1 つのチームからのユーザーを制限して、チーム名でタグ付けされた EMR Serverless アプリケーションにジョブを送信できるようにすることができます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EMRServerlessActions",
            "Effect": "Allow",
            "Action": [
                "emr-serverless:ListApplications",
                "emr-serverless:GetApplication",
                "emr-serverless:StartApplication",
                "emr-serverless:StartJobRun",
                "emr-serverless:CancelJobRun",
                "emr-serverless:ListJobRuns",
                "emr-serverless:GetJobRun"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Team": "team-name"
                }
            }
        }
    ]
}
```

タグベースのアクセスコントロールのポリシー

ID ベースのポリシーの条件を使用して、タグに基づいてアプリケーションとジョブの実行へのアクセスを制御できます。

次の例は、EMRサーバーレス条件キーで条件演算子を使用するさまざまなシナリオと方法を示しています。これらのIAMポリシーステートメントはデモンストレーションのみを目的としており、本番環境では使用しないでください。要件に応じて、アクセス権限を付与または拒否するようにポリシーステートメントを組み合わせる複数の方法があります。IAM ポリシーの計画とテストの詳細については、IAM「 ユーザーガイド」を参照してください。

> **Important**
>
> アクションをタグ付けするための権限を明示的に拒否することは重要な考慮事項です。これにより、ユーザーがリソースをタグ付けして意図せずにアクセス許可を付与することを防ぎます。リソースのタグ付けアクションが拒否されない場合、ユーザーはタグを変更して、タ

グベースのポリシーの意図を回避できます。タグ付けアクションを拒否するポリシーの例については、「タグを追加および削除するためのアクセス権限を拒否する」を参照してください。

以下の例は、EMRServerless アプリケーションで許可されるアクションを制御するために使用されるアイデンティティベースのアクセス許可ポリシーを示しています。

特定のタグの値があるリソースでのみアクションを許可する

次のポリシー例では、StringEquals条件演算子はタグ部門の値devと照合しようとします。タグ部門がアプリケーションに追加されていない場合、または値 が含まれていない場合dev、ポリシーは適用されず、アクションはこのポリシーでは許可されません。他のポリシーステートメントでアクションを許可しない場合、ユーザーはこの値でこのタグを持つアプリケーションでのみ操作できます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-serverless:GetApplication"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "emr-serverless:ResourceTag/department": "dev"
        }
      }
    }
  ]
}
```

条件付き演算子を使用して複数のタグ値を指定できます。例えば、departmentタグに 値devまたは が含まれているアプリケーションでアクションを許可するにはtest、前の例の条件ブロックを以下に置き換えます。

```
"Condition": {
        "StringEquals": {
```

```
          "emr-serverless:ResourceTag/department": ["dev", "test"]
        }
      }
```

リソースの作成時にタグ付けを要求する

次の例では、アプリケーションの作成時に タグを適用する必要があります。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-serverless:CreateApplication"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "emr-serverless:RequestTag/department": "dev"
        }
      }
    }
  ]
}
```

次のポリシーステートメントでは、アプリケーションにdepartment任意の値を含むタグがある場合にのみ、アプリケーションを作成できます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-serverless:CreateApplication"
      ],
      "Resource": "*",
      "Condition": {
        "Null": {
          "emr-serverless:RequestTag/department": "false"
        }
```

```
      }
    }
  ]
}
```

タグを追加および削除するためのアクセス権限を拒否する

このポリシーは、値が ではないタグを持つ EMR Serverless アプリケーションにdepartmentユーザーがタグを追加または削除することを防ぎますdev。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "emr-serverless:TagResource",
        "emr-serverless:UntagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "emr-serverless:ResourceTag/department": "dev"
        }
      }
    }
  ]
}
```

EMR Serverless 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 Amazon EMR Serverless リソース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」を参照してください。

ARNs 各リソースタイプの の形式など、Amazon EMR Serverless 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 EMR Serverless 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
- 自分の権限の表示をユーザーに許可する

ポリシーのベストプラクティス

> **Note**
>
> EMR Serverless は マネージドポリシーをサポートしていないため、以下に記載されている最初のプラクティスは適用されません。

ID ベースのポリシーは、誰かがアカウントで Amazon EMR Serverless リソースを作成、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 管理ポリシーを開始し、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にアクセス許可を付与するには、多くの一般的なユースケースにアクセス許可を付与するAWS 管理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 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、IAM「 ユーザーガイド」のMFA「保護APIアクセスの設定」を参照してください。

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

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": "*"
        }
    ]
}
```

AWS マネージドポリシーへの Amazon EMR Serverless 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 Amazon EMR Serverless 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、Amazon EMR Serverless 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
A mazonEMRServerless ServiceRolePolicy – 既存のポリシーの更新	Amazon EMR Serverless は、新しい `SidCloudWatchPolicyStatement` と を A mazonEMRServerless ServiceRolePolicy ポリシー `EC2PolicyStatement` に追加しました。	2024 年 1 月 25 日
A mazonEMRServerless ServiceRolePolicy – 既存のポリシーの更新	Amazon EMR Serverless は、Amazon EMR Serverless	2023 年 4 月 20 日

変更	説明	日付
	が"AWS/Usage" 名前空間での vCPU 使用量の集計アカウントメトリクスを発行できるようにする新しいアクセス許可を追加しました。	
Amazon EMR Serverless が変更の追跡を開始	Amazon EMR Serverless は AWS 、管理ポリシーの変更の追跡を開始しました。	2023 年 4 月 20 日

Amazon EMR Serverless 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、Amazon EMR Serverless と の使用時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

- Amazon EMR Serverless でアクションを実行する権限がありません
- iam を実行する権限がありません。PassRole
- AWS アカウント外のユーザーに Amazon EMR Serverless リソースへのアクセスを許可したい

Amazon EMR Serverless 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 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。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 emr-serverless:*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: emr-serverless:GetWidget on resource: my-example-widget
```

この場合、Mateo は、emr-serverless:*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。

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、Amazon EMR Serverless 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。

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 Amazon EMR Serverless `marymajor` でアクションを実行しようとすると発生します。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。

AWS アカウント外のユーザーに Amazon EMR Serverless 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。

- Amazon EMR Serverless 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、「」を参照してくださいAmazon EMR Serverless での ID とアクセスの管理 (IAM) 。
- 所有 AWS アカウント している 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。

- リソースへのアクセスをサードパーティー に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通じ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。

AWS Lake Formation でのEMRサーバーレスを使用したきめ細かなアクセスコントロール

概要

Amazon EMRリリース 7.2.0 以降では、 AWS Lake Formation を活用して、S3 でサポートされている Data Catalog テーブルにきめ細かなアクセスコントロールを適用できます。この機能を使用すると、 のテーブル、行、列、およびセルレベルのアクセスコントロールを設定できます。read Amazon EMR Serverless Spark ジョブ内のクエリ。Apache Spark バッチジョブとインタラクティブセッションのきめ細かなアクセスコントロールを設定するには、EMRStudio を使用します。Lake Formation の詳細と EMR Serverless での使用方法については、以下のセクションを参照してください。

で Amazon EMR Serverless を使用すると、追加料金 AWS Lake Formation が発生します。詳細については、「Amazon EMRの料金」を参照してください。

EMR Serverless の での仕組み AWS Lake Formation

Lake Formation で EMR Serverless を使用すると、各 Spark ジョブにアクセス許可のレイヤーを適用して、EMRServerless がジョブを実行するときに Lake Formation アクセス許可制御を適用できます。EMR Serverless は Spark リソースプロファイルを使用して 2 つのプロファイルを作成し、ジョブを効果的に実行します。ユーザープロファイルはユーザー提供のコードを実行し、システムプロファイルは Lake Formation ポリシーを適用します。詳細については、「What is AWS Lake Formation and Considerations and limitations」を参照してください。

Lake Formation で事前初期化された容量を使用する場合は、少なくとも 2 つの Spark ドライバーを使用することをお勧めします。Lake Formation が有効な各ジョブでは、ユーザープロファイル用に

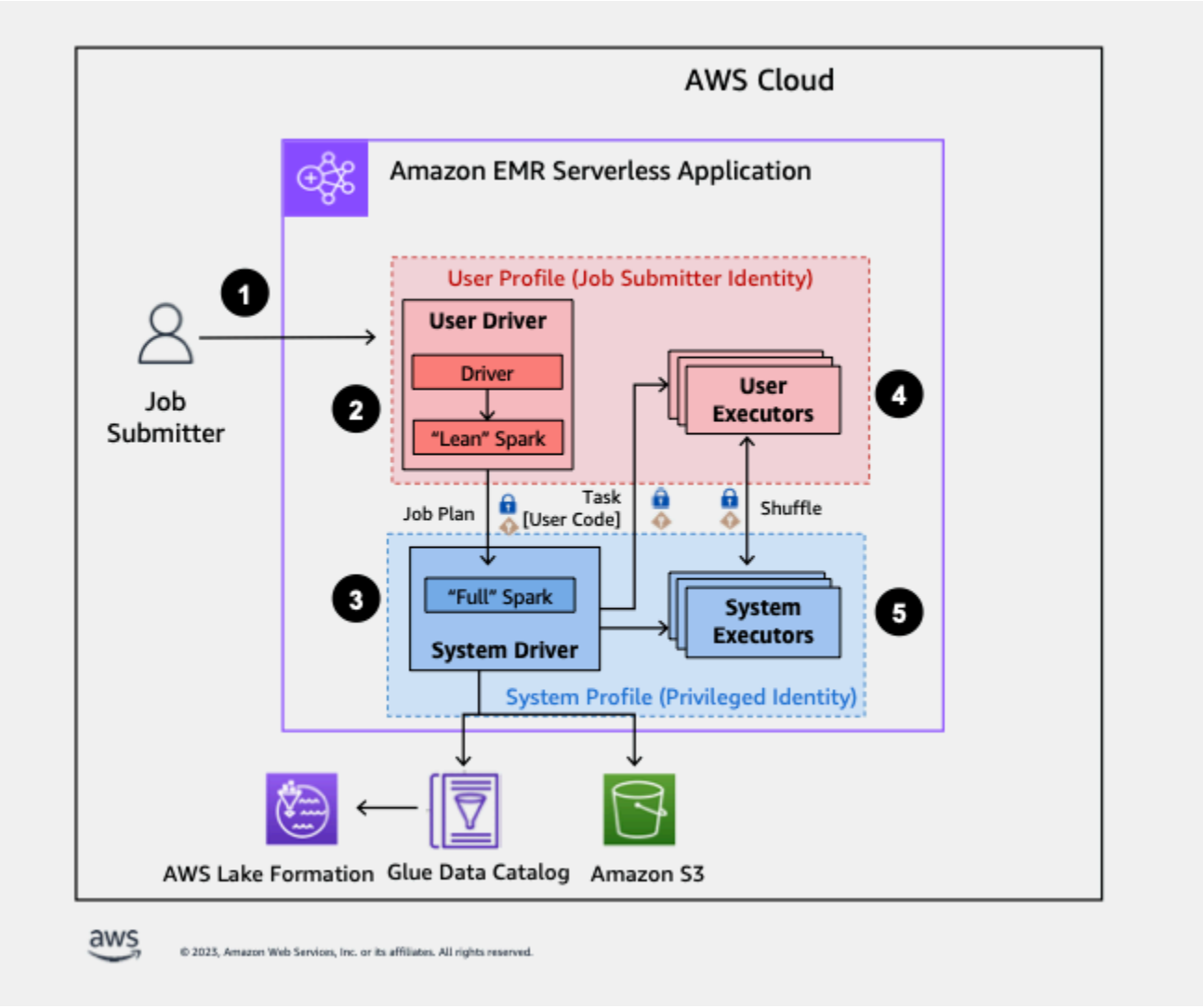
1 つとシステムプロファイル用に 1 つの 2 つの Spark ドライバーを使用します。最高のパフォーマンスを得るには、Lake Formation を使用しない場合と比較して、Lake Formation 対応ジョブのドライバーの数を 2 倍にする必要があります。

EMR Serverless で Spark ジョブを実行するときは、リソース管理とクラスターのパフォーマンスに対する動的割り当ての影響も考慮する必要があります。リソースプロファイルあたりのエグゼキューターspark.dynamicAllocation.maxExecutorsの最大数の設定は、ユーザーとシステムエグゼキューターの両方に適用されます。その数を最大許容エグゼキュター数と等しく設定すると、使用可能なすべてのリソースを使用する 1 つのタイプのエグゼキュターが原因でジョブの実行が停止し、ジョブの実行時に他のエグゼキュターが使用できなくなる可能性があります。

したがって、リソースが不足しないように、EMRServerless はリソースプロファイルあたりのデフォルトの最大エグゼキュター数をspark.dynamicAllocation.maxExecutors値の 90% に設定します。この設定は、 を 0 から 1 までの値spark.dynamicAllocation.maxExecutorsRatioで指定すると上書きできます。さらに、リソースの割り当てと全体的なパフォーマンスを最適化するために、次のプロパティを設定することもできます。

- spark.dynamicAllocation.cachedExecutorIdleTimeout
- spark.dynamicAllocation.shuffleTracking.timeout
- spark.cleaner.periodicGC.interval

以下は、EMRServerless が Lake Formation セキュリティポリシーで保護されたデータにアクセスする方法の概要です。



1. ユーザーは Spark ジョブを AWS Lake Formation対応の EMR Serverless アプリケーションに送信します。
2. EMR Serverless はジョブをユーザードライバーに送信し、ユーザープロファイルでジョブを実行します。ユーザードライバーは、タスクの起動、エグゼキュターのリクエスト、S3 または Glue Catalog へのアクセスができない Spark のリーンバージョンを実行します。ジョブプランを構築します。
3. EMR Serverless は、システムドライバーと呼ばれる 2 番目のドライバーを設定し、システムプロファイルで (特権 ID を使用して) 実行します。EMR サーバーレスは、通信用に 2 つのドライバー間に暗号化されたTLSチャネルを設定します。ユーザードライバーはチャネルを使用して、ジョブプランをシステムドライバーに送信します。システムドライバーは、ユーザーが送信したコードを実行しません。フル Spark を実行し、S3 およびデータカタログと通信してデータにア

クセスします。実行者をリクエストし、ジョブプランを一連の実行ステージにコンパイルします。

4. EMR その後、サーバーレスはユーザードライバーまたはシステムドライバーを使用してエグゼキュターでステージを実行します。どのステージのユーザーコードも、ユーザープロファイルエグゼキューターでのみ実行されます。
5. で保護された Data Catalog テーブルからデータを読み取るステージ AWS Lake Formation 、またはセキュリティフィルターを適用するステージは、システムエグゼキューターに委任されます。

Amazon での Lake Formation の有効化 EMR

Lake Formation を有効にするには、EMRサーバーレスアプリケーション を作成するときに、ランタイム設定パラメータのspark-defaults分類を spark.emr-serverless.lakeformation.enabled true に設定する必要があります。

```
aws emr-serverless create-application \
    --release-label emr-7.2.0 \
    --runtime-configuration '{
     "classification": "spark-defaults",
     "properties": {
      "spark.emr-serverless.lakeformation.enabled": "true"
      }
    }' \
    --type "SPARK"
```

EMR Studio で新しいアプリケーションを作成するときに Lake Formation を有効にすることもできます。「追加の設定」で利用可能なきめ細かなアクセスコントロール に Lake Formation を使用する」を選択します。

Lake Formation を EMR Serverless で使用すると、ワーカー間の暗号化がデフォルトで有効になるため、ワーカー間の暗号化を再度明示的に有効にする必要はありません。

Spark ジョブの Lake Formation の有効化

個々の Spark ジョブの Lake Formation を有効にするには、 を使用するときに true spark.emr-serverless.lakeformation.enabledに設定しますspark-submit。

```
--conf spark.emr-serverless.lakeformation.enabled=true
```

ジョブランタイムロールIAMのアクセス許可

Lake Formation のアクセス許可は、 AWS Glue Data Catalog リソース、Amazon S3 の場所、およびそれらの場所の基礎となるデータへのアクセスを制御します。IAM アクセス許可は、Lake Formation と AWS Glue APIsおよび リソースへのアクセスを制御します。データカタログ (SELECT) のテーブルにアクセスする Lake Formation のアクセス許可がある場合がありますが、 オペレーションに対するアクセスIAM許可がない場合、`glue:Get*`APIオペレーションは失敗します。

以下は、S3 のスクリプトへのアクセスIAM許可、S3 へのログのアップロード、 AWS Glue API アクセス許可、Lake Formation へのアクセス許可を提供する方法のポリシー例です。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ScriptAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
            "arn:aws:s3:::*.amzn-s3-demo-bucket/scripts",
            "arn:aws:s3:::*.amzn-s3-demo-bucket/*" ]
        },
        {
         "Sid": "LoggingAccess",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": [
               "arn:aws:s3:::amzn-s3-demo-bucket/logs/*"
            ]
        },
        {
            "Sid": "GlueCatalogAccess",
            "Effect": "Allow",
            "Action": [
                 "glue:Get*",
                 "glue:Create*",
                 "glue:Update*"
```

```
            ],
            "Resource": ["*"]
        },
        {
            "Sid": "LakeFormationAccess",
            "Effect": "Allow",
            "Action": [
             "lakeformation:GetDataAccess"
             ],
            "Resource": ["*"]
        }
    ]
}
```

ジョブランタイムロールの Lake Formation アクセス許可の設定

まず、Hive テーブルの場所を Lake Formation に登録します。次に、目的のテーブルにジョブランタイムロールのアクセス許可を作成します。Lake Formation の詳細については、 AWS Lake Formation デベロッパーガイドの「 とは AWS Lake Formation」を参照してください。

Lake Formation アクセス許可を設定したら、Amazon EMR Serverless で Spark ジョブを送信できます。Spark ジョブの詳細については、「Spark の例」を参照してください。

ジョブ実行の送信

Lake Formation 許可の設定が完了したら、EMRServerless で Spark ジョブを送信できます。 Iceberg ジョブを実行するには、次のspark-submitプロパティを指定する必要があります。

```
--conf spark.sql.catalog.spark_catalog=org.apache.iceberg.spark.SparkSessionCatalog
--conf spark.sql.catalog.spark_catalog.warehouse=<S3_DATA_LOCATION>
--conf spark.sql.catalog.spark_catalog.glue.account-id=<ACCOUNT_ID>
--conf spark.sql.catalog.spark_catalog.client.region=<REGION>
--conf spark.sql.catalog.spark_catalog.glue.endpoint=https://
glue.<REGION>.amazonaws.com
```

オープンテーブル形式のサポート

Amazon EMRリリース 7.2.0 には、Lake Formation に基づくきめ細かなアクセスコントロールのサポートが含まれています。EMR Serverless は、Hive テーブルタイプと Iceberg テーブルタイプをサポートしています。次の表は、サポートされているすべてのオペレーションを示しています。

オペレーション	[Hive]	Iceberg
DDL コマンド	IAM ロールアクセス許可のみ	IAM ロールアクセス許可のみ
増分クエリ	該当しない	完全サポートされています
タイムトラベルクエリ	このテーブル形式には適用されません	完全サポートされています
メタデータテーブル	このテーブル形式には適用されません	サポートされていますが、特定のテーブルは非表示になっています。詳細については、「考慮事項と制限事項」を参照してください。
DML INSERT	アクセスIAM許可のみ	アクセスIAM許可のみ
DML UPDATE	このテーブル形式には適用されません	アクセスIAM許可のみ
DML DELETE	このテーブル形式には適用されません	アクセスIAM許可のみ
読み込みオペレーション	完全サポートされています	完全サポートされています
ストアドプロシージャ	該当しない	`register_table` および の例外でサポートされますmigrate。詳細については、「考慮事項と制限事項」を参照してください。

考慮事項と制約事項

Lake Formation を EMR Serverless で使用する場合は、次の考慮事項と制限事項を考慮してください。

> **Note**
>
> EMR Serverless で Spark ジョブの Lake Formation を有効にすると、ジョブはシステムドライバーとユーザードライバーを起動します。起動時に事前初期化された容量を指定した場合、ドライバーは事前初期化された容量からプロビジョニングし、システムドライバーの数は指定したユーザードライバーの数と等しくなります。オンデマンドキャパシティを選択すると、EMRサーバーレスはユーザードライバーに加えてシステムドライバーを起動します。Lake Formation ジョブで EMR Serverless に関連するコストを見積もるには、 を使用しますAWS Pricing Calculator。

Amazon EMR Serverless with Lake Formation は、 AWS GovCloud （ 米国東部) と AWS GovCloud （ 米国西部) を除く、サポートされているすべてのEMRサーバーレスリージョンで使用できます。

- Amazon EMR Serverless は、Apache Hive および Apache Iceberg テーブルに対してのみ、Lake Formation を介したきめ細かなアクセスコントロールをサポートします。Apache Hive 形式には、Parquet、ORC、および xSV が含まれます。
- Lake Formation 対応アプリケーションは、カスタマイズされた EMR Serverless イメージ の使用をサポートしていません。
- Lake Formation ジョブ`DynamicResourceAllocation`をオフにすることはできません。
- Lake Formation は Spark ジョブでのみ使用できます。
- EMR Lake Formation を使用したサーバーレスは、ジョブ全体で 1 つの Spark セッションのみをサポートします。
- EMR Lake Formation を使用したサーバーレスは、リソースリンクを介して共有されるクロスアカウントテーブルクエリのみをサポートします。
- 以下はサポートされていません。
  - レジリエント分散データセット (RDD)
  - Spark ストリーミング
  - Lake Formation に付与されたアクセス許可で書き込み
  - ネストされた列のアクセスコントロール
- EMR サーバーレスは、以下を含むシステムドライバーの完全な分離を損なう可能性のある機能をブロックします。
  - UDTs、H iveUDFs、およびカスタムクラスを含むユーザー定義関数
  - カスタムデータソース

  - Spark 拡張機能、コネクタ、メタストア用の追加のジャーの供給
  - ANALYZE TABLE コマンド
- アクセスコントロールEXPLAIN PLAN、 などのDDLオペレーションを強制するには、制限された情報を公開DESCRIBE TABLEしません。
- EMR Serverless は、Lake Formation 対応アプリケーションのシステムドライバー Spark ログへのアクセスを制限します。システムドライバーはより多くのアクセスで実行されるため、システムドライバーが生成するイベントとログには機密情報が含まれる可能性があります。許可されていないユーザーまたはコードがこの機密データにアクセスできないように、EMRサーバーレスはシステムドライバーログへのアクセスを無効にします。トラブルシューティングについては、 AWS サポートにお問い合わせください。
- Lake Formation でテーブルの場所を登録した場合、データアクセスパスは、EMRサーバーレスジョブランタイムロールのIAMアクセス許可に関係なく、Lake Formation の保存された認証情報を経由します。テーブルの場所に登録されたロールを誤って設定すると、S3 アクセスIAM許可を持つロールを使用する送信ジョブがテーブルの場所に対して失敗します。
- Lake Formation テーブルへの書き込みでは、Lake Formation に付与されたIAMアクセス許可ではなくアクセス許可が使用されます。ジョブランタイムロールに必要な S3 アクセス許可がある場合は、それを使用して書き込みオペレーションを実行できます。

Apache Iceberg を使用する際の考慮事項と制限事項は次のとおりです。

- Apache Iceberg はセッションカタログでのみ使用でき、任意の名前のカタログでは使用できません。
- Lake Formation に登録されている Iceberg filesテーブルは、メタデータテーブル history、metadata_log_entries、、snapshots、manifests、および のみをサポートしますrefs。Amazon は、、、 などの機密データを持つ可能性のある列をEMR非表示partitionspathにしますsummaries。この制限は、Lake Formation に登録されていない Iceberg テーブルには適用されません。
- Lake Formation に登録していないテーブルは、すべての Iceberg ストアドプロシージャをサポートしています。register_table および migrateプロシージャは、どのテーブルでもサポートされていません。
- V1 の代わりに V1 DataFrameWriterV2 を使用することをお勧めします。

トラブルシューティング

トラブルシューティングソリューションについては、以下のセクションを参照してください。

ログ記録

EMR Serverless は、Spark リソースプロファイルを使用してジョブ実行を分割します。EMR Serverless はユーザープロファイルを使用して指定したコードを実行しますが、システムプロファイルは Lake Formation ポリシーを適用します。ユーザープロファイルとして実行されたタスクのログにアクセスできます。

ライブ UI と Spark 履歴サーバー

Live UI と Spark History Server には、ユーザープロファイルから生成されたすべての Spark イベントと、システムドライバーから生成された編集済みイベントがあります。

ユーザードライバーとシステムドライバーの両方からすべてのタスクが Executors タブに表示されます。ただし、ログリンクはユーザープロファイルでのみ使用できます。また、出力レコードの数など、一部の情報は Live UI から編集されます。

Lake Formation のアクセス許可が不十分なため、ジョブが失敗しました

ジョブランタイムロールには、アクセスするテーブルDESCRIBEで SELECTと を実行するアクセス許可があることを確認してください。

RDD 実行が失敗したジョブ

EMR サーバーレスは現在、Lake Formation 対応ジョブでの回復力のある分散データセット (RDD) オペレーションをサポートしていません。

Amazon S3 のデータファイルにアクセスできない

Lake Formation にデータレイクの場所が登録されていることを確認します。

セキュリティ検証の例外

EMR サーバーレスがセキュリティ検証エラーを検出しました。 AWS サポートにお問い合わせください。

アカウント間で AWS Glue Data Catalog とテーブルを共有する

データベースとテーブルをアカウント間で共有し、Lake Formation を引き続き使用できます。詳細については、「Lake Formation でのクロスアカウントデータ共有」および「 を使用して AWS Glue Data Catalog とテーブルのクロスアカウントを共有する方法 AWS Lake Formation」を参照してください。

ワーカー間の暗号化

Amazon EMRバージョン 6.15.0 TLS以降では、Spark ジョブ実行のワーカー間の相互暗号化通信を有効にできます。有効にすると、EMRサーバーレスはジョブ実行でプロビジョニングされたワーカーごとに一意の証明書を自動的に生成して配布します。これらのワーカーが通信して制御メッセージの交換やシャッフルデータの転送を行う場合、相互TLS接続を確立し、設定された証明書を使用して相互のアイデンティティを検証します。ワーカーが別の証明書を検証できない場合、TLSハンドシェイクは失敗し、EMRサーバーレスはそれらの間の接続を中止します。

EMR サーバーレスで Lake Formation を使用している場合、相互TLS暗号化はデフォルトで有効になっています。

EMR Serverless TLSでの相互暗号化の有効化

spark アプリケーションで相互TLS暗号化を有効にするには、サーバーレスアプリケーション を作成するときに `spark.ssl.internode.enabled`を true に設定します。 EMR を使用している場合 AWS コンソールを使用してEMRサーバーレスアプリケーションを作成し、カスタム設定を使用するを選択し、アプリケーション設定 を展開して、 を入力します`runtimeConfiguration`。

```
aws emr-serverless create-application \
--release-label emr-6.15.0 \
--runtime-configuration '{
  "classification": "spark-defaults",
  "properties": {"spark.ssl.internode.enabled": "true"}
}' \
--type "SPARK"
```

個々のSparkジョブ実行の相互TLS暗号化を有効にする場合は、 を使用するときに true `spark.ssl.internode.enabled`に設定します`spark-submit`。

```
--conf spark.ssl.internode.enabled=true
```

EMR Serverless によるデータ保護のための Secrets Manager

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。この概要については、「AWS Secrets Manager ユーザーガイド」を参照してください

Secrets Manager は、 AWS Key Management Service キーを使用してシークレットを暗号化します。詳細については、AWS Secrets Manager 「 ユーザーガイド」の「シークレット暗号化と復号」を参照してください。

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。詳細については、AWS Secrets Manager ユーザーガイド の「 シークレット AWS Secrets Manager のローテーション」を参照してください。

Amazon EMR Serverless は と統合 AWS Secrets Manager されているため、データを Secrets Manager に保存し、設定でシークレット ID を使用できます。

EMR Serverless がシークレットを使用する方法

データを Secrets Manager に保存し、EMRServerless の設定でシークレット ID を使用すると、機密性の高い設定データをプレーンテキストで EMR Serverless に渡さず、外部 に公開しませんAPIs。キーと値のペアに Secrets Manager に保存したシークレットのシークレット ID が含まれていることを示すと、EMRServerless はジョブを実行するために設定データをワーカーに送信するときにシークレットを取得します。

設定のキーと値のペアに Secrets Manager に保存されているシークレットへの参照が含まれていることを示すには、設定値に`EMR.secret@`注釈を追加します。シークレット ID 注釈を持つ設定プロパティの場合、EMRServerless は Secrets Manager を呼び出し、ジョブの実行時にシークレットを解決します。

シークレットを作成する方法

シークレットを作成するには、AWS Secrets Manager 「 ユーザーガイド」のAWS Secrets Manager「シークレットの作成」のステップに従います。ステップ 3 で、プレーンテキストフィールドを選択して、機密性の高い値を入力します。

設定分類にシークレットを指定する

次の例は、 の設定分類でシークレットを提供する方法を示していますStartJobRun。アプリケーションレベルで Secrets Manager の分類を設定する場合は、「」を参照してくださいEMR Serverless のデフォルトのアプリケーション設定。

この例では、 を、取得するシークレットの名前*SecretName*に置き換えます。ハイフン、Secrets Manager がシークレットの末尾に追加する 6 文字を含めますARN。詳細については、「シークレットを作成する方法」を参照してください。

このセクションの内容

- シークレット参照の指定 - Spark
- シークレットリファレンスの指定 - Hive

シークレット参照の指定 - Spark

Example – Spark の外部 Hive メタストア設定でシークレットリファレンスを指定する

```
aws emr-serverless start-job-run \
  --application-id "application-id" \
  --execution-role-arn "job-role-arn" \
  --job-driver '{
        "sparkSubmit": {
            "entryPoint": "s3://amzn-s3-demo-bucket/scripts/spark-jdbc.py",
            "sparkSubmitParameters": "--jars s3://amzn-s3-demo-bucket/mariadb-
connector-java.jar
            --conf
 spark.hadoop.javax.jdo.option.ConnectionDriverName=org.mariadb.jdbc.Driver
            --conf spark.hadoop.javax.jdo.option.ConnectionUserName=connection-user-
name
            --conf
 spark.hadoop.javax.jdo.option.ConnectionPassword=EMR.secret@SecretName
            --conf spark.hadoop.javax.jdo.option.ConnectionURL=jdbc:mysql://db-host:db-
port/db-name
            --conf spark.driver.cores=2
            --conf spark.executor.memory=10G
            --conf spark.driver.memory=6G
            --conf spark.executor.cores=4"
        }
    }' \
    --configuration-overrides '{
```

```
        "monitoringConfiguration": {
        "s3MonitoringConfiguration": {
            "logUri": "s3://amzn-s3-demo-bucket/spark/logs/"
        }
    }
}'
```

Example – **spark-defaults**分類で外部 Hive メタストア設定のシークレットリファレンスを指定する

```
{
        "classification": "spark-defaults",
        "properties": {

 "spark.hadoop.javax.jdo.option.ConnectionDriverName":"org.mariadb.jdbc.Driver"
            "spark.hadoop.javax.jdo.option.ConnectionURL":"jdbc:mysql://db-host:db-
port/db-name"
            "spark.hadoop.javax.jdo.option.ConnectionUserName":"connection-user-name"
            "spark.hadoop.javax.jdo.option.ConnectionPassword":
 "EMR.secret@SecretName",
        }
    }
```

シークレットリファレンスの指定 - Hive

Example – Hive の外部 Hive メタストア設定でシークレットリファレンスを指定する

```
aws emr-serverless start-job-run \
  --application-id "application-id" \
  --execution-role-arn "job-role-arn" \
    --job-driver '{
        "hive": {
        "query": "s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-query.ql",
        "parameters": "--hiveconf hive.exec.scratchdir=s3://amzn-s3-demo-bucket/emr-
serverless-hive/hive/scratch
                    --hiveconf hive.metastore.warehouse.dir=s3://amzn-s3-demo-bucket/
emr-serverless-hive/hive/warehouse
                    --hiveconf javax.jdo.option.ConnectionUserName=username
                    --hiveconf
 javax.jdo.option.ConnectionPassword=EMR.secret@SecretName
                    --hiveconf
 hive.metastore.client.factory.class=org.apache.hadoop.hive.ql.metadata.SessionHiveMetaStoreCli
```

```
                    --hiveconf
 javax.jdo.option.ConnectionDriverName=org.mariadb.jdbc.Driver
                    --hiveconf javax.jdo.option.ConnectionURL=jdbc:mysql://db-host:db-
port/db-name"
        }
    }' \
    --configuration-overrides '{
        "monitoringConfiguration": {
        "s3MonitoringConfiguration": {
            "logUri": "s3://EXAMPLE-LOG-BUCKET"
        }
    }
}'
```

Example – **hive-site**分類で外部 Hive メタストア設定のシークレットリファレンスを指定する

```
{
    "classification": "hive-site",
    "properties": {
        "hive.metastore.client.factory.class":
 "org.apache.hadoop.hive.ql.metadata.SessionHiveMetaStoreClientFactory",
        "javax.jdo.option.ConnectionDriverName": "org.mariadb.jdbc.Driver",
        "javax.jdo.option.ConnectionURL": "jdbc:mysql://db-host:db-port/db-name",
        "javax.jdo.option.ConnectionUserName": "username",
        "javax.jdo.option.ConnectionPassword": "EMR.secret@SecretName"
    }
}
```

EMR Serverless がシークレットを取得するためのアクセスを許可する

EMR Serverless が Secrets Manager からシークレット値を取得できるようにするには、シークレットの作成時に次のポリシーステートメントを追加します。EMR Serverless がシークレット値を読み取るには、カスタマー管理KMSキーを使用してシークレットを作成する必要があります。詳細については、AWS Secrets Manager 「 ユーザーガイド」の「 KMSキーのアクセス許可」を参照してください。

次のポリシーでは、 をアプリケーションの ID *applicationId* に置き換えます。

シークレットのリソースポリシー

EMR Serverless がシークレット値を取得 AWS Secrets Manager できるようにするには、 のシークレットのリソースポリシーに次のアクセス許可を含める必要があります。特定のアプリケーション

のみがこのシークレットを取得できるようにするには、オプションでポリシーの条件として EMR Serverless アプリケーション ID を指定できます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret"
      ],
      "Principal": {
        "Service": [
          "emr-serverless.amazonaws.com"
        ]
      },
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceArn": "arn:aws:emr-serverless:AWS #####:aws_account_id:/
applications/applicationId"
        }
      }
    }
  ]
}
```

カスタマーマネージド AWS Key Management Service（ AWS KMS) キーの次のポリシーを使用してシークレットを作成します。

カスタマーマネージド AWS KMS キーのポリシー

```
{
    "Sid": "Allow EMR Serverless to use the key for decrypting secrets",
    "Effect": "Allow",
    "Principal": {
        "Service": [
            "emr-serverless.amazonaws.com"
        ]
    },
```

```
    "Action": [
        "kms:Decrypt",
        "kms:DescribeKey"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "kms:ViaService": "secretsmanager.AWS #####.amazonaws.com"
        }
    }
}
```

シークレットのローテーション

ローテーションは、シークレットを定期的に更新するときです。指定したスケジュールでシークレットを自動的にローテーション AWS Secrets Manager するように を設定できます。これにより、長期シークレットを短期シークレットに置き換えることができます。これにより、侵害のリスクを減らすことができます。EMR Serverless は、ジョブが実行中の状態に移行すると、注釈付き設定からシークレット値を取得します。お客様またはプロセスが Secrets Manager のシークレット値を更新する場合は、ジョブが更新された値を取得できるように、新しいジョブを送信する必要があります。

> **Note**
>
> 既に実行中の状態のジョブは、更新されたシークレット値を取得できません。これにより、ジョブが失敗する可能性があります。

EMR Serverless での Amazon S3 Access Grants の使用

EMR Serverless の S3 Access Grants の概要

Amazon EMRリリース 6.15.0 以降では、Amazon S3 Access Grants は、EMRServerless からの Amazon S3 データへのアクセスを強化するために使用できるスケーラブルなアクセスコントロールソリューションを提供します。S3 データのアクセス許可設定が複雑または大規模な場合は、Access Grants を使用して、ユーザー、ロール、アプリケーションの S3 データ権限をスケーリングできます。

S3 Access Grants を使用して、ランタイムロールまたは EMR Serverless アプリケーションにアクセスできる ID にアタッチされたIAMロールによって付与されたアクセス許可を超えて Amazon S3 データへのアクセスを強化します。

詳細については、「Amazon 管理ガイド」のS3 Access Grants for Amazon によるアクセスEMRの管理」および「Amazon Simple Storage Service ユーザーガイド」のS3 Access Grants によるアクセスの管理」を参照してください。 EMR

このセクションでは、S3 Access Grants を使用して Amazon S3 のデータへのアクセスを提供する EMR Serverless アプリケーションを起動する方法について説明します。 Amazon S3 S3 Access Grants を他の Amazon EMRデプロイで使用する手順については、以下のドキュメントを参照してください。

- Amazon での S3 Access Grants の使用 EMR
- EMRでの Amazon での S3 Access Grants の使用 EKS

データ管理用の S3 Access Grants を使用してEMRサーバーレスアプリケーションを起動する

EMR Serverless で S3 Access Grants を有効にし、Spark アプリケーションを起動できます。アプリケーションが S3 データをリクエストすると、Amazon S3 は特定のバケット、プレフィックス、またはオブジェクトを対象とする一時的な認証情報を提供します。

1. EMR Serverless アプリケーションのジョブ実行ロールを設定します。Spark ジョブを実行して S3 Access Grants を使用するために必要なIAMアクセス許可`s3:GetDataAccess`と を含めます`s3:GetAccessGrantsInstanceForPrefix`。

```
{
    "Effect": "Allow",
    "Action": [
    "s3:GetDataAccess",
    "s3:GetAccessGrantsInstanceForPrefix"
    ],
    "Resource": [     //LIST ALL INSTANCE ARNS THAT THE ROLE IS ALLOWED TO QUERY
         "arn:aws_partition:s3:Region:account-id1:access-grants/default",
         "arn:aws_partition:s3:Region:account-id2:access-grants/default"
    ]
}
```

> **Note**
>
> S3 に直接アクセスする追加のアクセス許可を持つジョブ実行のIAMロールを指定すると、ユーザーは S3 Access Grants からのアクセス許可がない場合でも、そのロールによって許可されたデータにアクセスできます。

2. 次の例に示すように、Amazon EMRリリースラベルが 6.15.0 以降でspark-defaults分類された EMR Serverless アプリケーションを起動します。*red text* の値を使用シナリオに適した値に置き換えます。

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
        "sparkSubmit": {
            "entryPoint": "s3://us-east-1.elasticmapreduce/emr-containers/samples/
wordcount/scripts/wordcount.py",
            "entryPointArguments": ["s3://amzn-s3-demo-destination-bucket1/
wordcount_output"],
            "sparkSubmitParameters": "--conf spark.executor.cores=1 --conf
 spark.executor.memory=4g --conf spark.driver.cores=1 --conf spark.driver.memory=4g
 --conf spark.executor.instances=1"
        }
    }' \
  --configuration-overrides '{
    "applicationConfiguration": [{
        "classification": "spark-defaults",
        "properties": {
          "spark.hadoop.fs.s3.s3AccessGrants.enabled": "true",
          "spark.hadoop.fs.s3.s3AccessGrants.fallbackToIAM": "false"
         }
      }]
}'
```

EMR Serverless での S3 Access Grants に関する考慮事項

Amazon S3 Access Grants with EMR Serverless を使用する際の重要なサポート、互換性、動作情報については、「Amazon EMR管理ガイド」の「Amazon での S3 Access Grants に関する考慮事項 EMR」を参照してください。

を使用した Amazon EMR Serverless API呼び出しの口グ記録 AWS CloudTrail

Amazon EMR Serverless は と統合されています。 AWS CloudTrail、ユーザー、ロール、または によって実行されたアクションを記録するサービス AWS EMR Serverless. の サービスでは、EMRServerless のすべてのAPI呼び出しをイベントとして CloudTrail キャプチャします。キャプチャされた呼び出しには、EMRサーバーレスコンソールからの呼び出しとEMR、サーバーレスAPIオペレーション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、EMRServerless の CloudTrailイベントなど、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。証跡を設定しない場合でも、 CloudTrail コンソールのイベント履歴 で最新のイベントを表示できます。で収集された情報を使用して CloudTrail、EMRサーバーレスに対するリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。

の詳細については CloudTrail、「」を参照してください。 AWS CloudTrail ユーザーガイド。

EMR のサーバーレス情報 CloudTrail

CloudTrail が で有効になっている AWS アカウント アカウントを作成するとき。EMR Serverless でアクティビティが発生すると、そのアクティビティは他の とともに CloudTrail イベントに記録されます。 AWS イベント履歴 の サービスイベント。で最近のイベントを表示、検索、ダウンロードできます。 AWS アカウント。 詳細については、「イベント履歴 を使用した CloudTrail イベントの表示」を参照してください。

のイベントを継続的に記録するには AWS アカウント Serverless のイベントを含む EMR 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 に適用されます。 AWS リージョン。 証跡は、 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ます。 AWS パーティション分割し、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、他の AWS は、CloudTrail ログで収集されたイベントデータをさらに分析し、それに基づいて行動するための サービスです。詳細については、次を参照してください:

- 追跡を作成するための概要
- CloudTrail がサポートするサービスと統合
- の Amazon SNS通知の設定 CloudTrail
- 複数のリージョンからの CloudTrail ログファイルの受信と複数のアカウントからの CloudTrail ログファイルの受信

すべてのEMRサーバーレスアクションは によってログに記録 CloudTrail され、EMR「サーバーレスAPIリファレンス」に記載されています。例えば、、、および CancelJobRunアクションを呼び出すStartJobRunとCreateApplication、 CloudTrail ログファイルにエントリが生成されます。

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 root または で行われたかどうか AWS Identity and Access Management （ IAM) ユーザー認証情報。
- リクエストがロールまたはフェデレーションユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。
- リクエストが別の によって行われたかどうか AWS サービス。

詳細については、CloudTrail userIdentity 「」要素を参照してください。

EMR Serverless ログファイルエントリについて

証跡は、指定した Amazon S3 バケットにイベントをログファイルとして配信できるようにする設定です。 CloudTrail ログファイルには 1 つ以上のログエントリが含まれます。イベントは任意のソースからの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストパラメータなどに関する情報が含まれます。 CloudTrail ログファイルはパブリックAPIコールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

次の例は、 CreateApplicationアクションを示す CloudTrail ログエントリを示しています。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE:admin",
        "arn": "arn:aws:sts::012345678910:assumed-role/Admin/admin",
        "accountId": "012345678910",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::012345678910:role/Admin",
                "accountId": "012345678910",
                "userName": "Admin"
```

```
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-06-01T23:46:52Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-06-01T23:49:28Z",
    "eventSource": "emr-serverless.amazonaws.com",
    "eventName": "CreateApplication",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "203.0.113.0",
    "userAgent": "PostmanRuntime/7.26.10",
    "requestParameters": {
        "name": "my-serverless-application",
        "releaseLabel": "emr-6.6",
        "type": "SPARK",
        "clientToken": "0a1b234c-de56-7890-1234-567890123456"
    },
    "responseElements": {
        "name": "my-serverless-application",
        "applicationId": "1234567890abcdef0",
        "arn": "arn:aws:emr-serverless:us-west-2:555555555555:/
applications/1234567890abcdef0"
    },
    "requestID": "890b8639-e51f-11e7-b038-EXAMPLE",
    "eventID": "874f89fa-70fc-4798-bc00-EXAMPLE",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "012345678910",
    "eventCategory": "Management"
}
```

Amazon EMR Serverless のコンプライアンス検証

EMR Serverless のセキュリティとコンプライアンスは、複数の の一部としてサードパーティーの監査者によって評価されます。 AWS 以下を含む コンプライアンスプログラム :

- システムと組織のコントロール (SOC)
- Payment Card Industry Data Security Standard (PCI DSS)

- Federal Risk and Authorization Management Program (Fed RAMP) Moderate
- 健康保険の相互運用性と説明責任に関する法律 (HIPAA)

AWS は、頻繁に更新される のリストを提供します。 AWS の特定のコンプライアンスプログラムの対象となる のサービス 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。

サードパーティーの監査レポートは、 を使用してダウンロードできます。 AWS Artifact。 詳細については、「 でのレポートのダウンロード」を参照してください。 AWS アーティファクト 。

の詳細については、「」を参照してください。 AWS コンプライアンスプログラムについては、「」を参照してください。 AWS コンプライアンスプログラム 。

EMR Serverless を使用する際のお客様のコンプライアンス責任は、お客様のデータの機密性、組織のコンプライアンス目的、適用可能な法律および規制によって決まります。EMR Serverless の使用が HIPAA、、PCIまたは FedRAMP Moderate などの標準に準拠していることを前提としている場合、 AWS は、以下に役立つリソースを提供します。

- 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を にデプロイするためのアーキテクチャ上の考慮事項と手順について説明する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 AWS.
- AWS カスタマーコンプライアンスガイドは、コンプライアンスの観点から責任共有モデルを理解するのに役立ちます。このガイドでは、 を保護するためのベストプラクティスをまとめています。 AWS のサービス と は、ガイダンスを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 (PCI) 、国際標準化機構 (ISO) を含む) のセキュリティコントロールにマッピングします。
- AWS Config を使用すると、社内プラクティス、業界ガイドライン、およびに対するリソースの構成の準拠状態を評価できます。
- AWS コンプライアンスリソースは、ワークブックとガイドのコレクションであり、お客様の業界や地域に適用される場合があります。
- AWS Security Hub では、 内のセキュリティ状態を包括的に確認できます。 AWS と は、セキュリティ業界標準とベストプラクティスへの準拠を確認するのに役立ちます。
- AWS Audit Manager – これ AWS のサービス は、 の継続的な監査に役立ちます。 AWS を使用して、リスクの管理方法と規制や業界標準への準拠を簡素化します。

Amazon EMR Serverless 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。 AWS リージョンには、低レイテンシー、高いスループット、そして高度の冗長ネットワークで接続されている複数の物理的に独立および隔離されたアベイラビリティーゾーンがあります。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。

の詳細については、「」を参照してください。 AWS リージョンとアベイラビリティーゾーンについては、「」を参照してください。 AWS グローバルインフラストラクチャ 。

に加えて AWS グローバルインフラストラクチャである Amazon EMR Serverless は、 を介して Amazon S3 との統合を提供しEMRFS、データの耐障害性とバックアップのニーズをサポートします。

Amazon EMR Serverless 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 EMR は によって保護されています。 AWS グローバルネットワークセキュリティ。参考情報 AWS セキュリティサービスとその方法 AWS インフラストラクチャを保護するには、「」を参照してください。 AWS クラウドセキュリティ 。を設計するには AWS インフラストラクチャセキュリティのベストプラクティスを使用する 環境、「セキュリティの柱」の「インフラストラクチャの保護」を参照してください。 AWS Well-Architected フレームワーク 。

を使用する AWS は、ネットワークEMR経由で Amazon にアクセスするためのAPI呼び出しを発行しました。クライアントは以下をサポートする必要があります:

- Transport Layer Security (TLS) 。1TLS.2 が必要で、1.3 TLS をお勧めします。
- (Ephemeral Diffie-HellmanPFS) や DHE (Elliptic Curve Ephemeral Diffie-Hellman) などの完全前方秘匿性 ECDHE () を備えた暗号スイート。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、 IAMプリンシパルに関連付けられたアクセスキー ID と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) リクエストに署名するための一時的なセキュリティ認証情報を生成します。

Amazon EMR Serverless での設定と脆弱性の分析

AWS は、ゲストオペレーティングシステム (OS) やデータベースのパッチ適用、ファイアウォール設定、ディザスタリカバリなどの基本的なセキュリティタスクを処理します。これらの手順は適切な第三者によって確認され、証明されています。詳細については、以下のリソース を参照してください。

- Amazon EMR Serverless のコンプライアンス検証
- 責任共有モデル
- Amazon Web Services: セキュリティプロセスの概要 (ホワイトペーパー)

のエンドポイントとクォータ EMR Serverless

サービスエンドポイント

プログラムで に接続するには AWS のサービス、エンドポイント を使用します。エンドポイントは、 AWS ウェブサービスのエントリポイントURLの です。標準 AWS エンドポイントに加えて、一部の では AWS のサービス 、選択したリージョンでFIPSエンドポイントを提供しています。次の表に、EMRServerless のサービスエンドポイントを示します。詳細については、AWS のサービス エンドポイントを参照してください。

リージョン名	リージョン	エンドポイント	プロトコル
米国東部 (オハイオ)	us-east-2 (次のアベイラビリティーゾーンに限定: use2-az1、use2-az2、および use2-az3)	emr-serverless.us-east-2.amazonaws.com	HTTPS
米国東部 (バージニア北部)	us-east-1 (次のアベイラビリティーゾーンに限定: use1-az1、use1-az2、use1-az4、use1-az5、および use1-az6)	emr-serverless.us-east-1.amazonaws.com emr-serverless-fips.us-east-1.amazonaws.com	HTTPS
米国西部 (北カリフォルニア)	us-west-1	emr-serverless.us-west-1.amazonaws.com	HTTPS
米国西部 (オレゴン)	us-west-2	emr-serverless.us-west-2.amazonaws.com	HTTPS

リージョン名	リージョン	エンドポイント	プロトコル
		emr-serverless-fips.us-west-2.amazonaws.com	
アフリカ (ケープタウン)	af-south-1	emr-serverless.af-south-1.amazonaws.com	HTTPS
アジアパシフィック (香港)	ap-east-1	emr-serverless.ap-east-1.amazonaws.com	HTTPS
アジアパシフィック (ジャカルタ)	ap-southeast-3	emr-serverless.ap-southeast-3.amazonaws.com	HTTPS
アジアパシフィック (ムンバイ)	ap-south-1	emr-serverless.ap-south-1.amazonaws.com	HTTPS
アジアパシフィック (大阪)	ap-northeast-3	emr-serverless.ap-northeast-3.amazonaws.com	HTTPS

リージョン名	リージョン	エンドポイント	プロトコル
アジアパシフィック (ソウル)	ap-northeast-2	emr-serverless.ap-northeast-2.amazonaws.com	HTTPS
アジアパシフィック (シンガポール)	ap-southeast-1	emr-serverless.ap-southeast-1.amazonaws.com	HTTPS
アジアパシフィック (シドニー)	ap-southeast-2	emr-serverless.ap-southeast-2.amazonaws.com	HTTPS
アジアパシフィック (東京)	ap-northeast-1	emr-serverless.ap-northeast-1.amazonaws.com	HTTPS
カナダ (中部)	ca-central-1 (次のアベイラビリティーゾーンに限定: cac1-az1および cac1-az2)	emr-serverless.ca-central-1.amazonaws.com	HTTPS
欧州 (フランクフルト)	eu-central-1	emr-serverless.eu-central-1.amazonaws.com	HTTPS

リージョン名	リージョン	エンドポイント	プロトコル
欧州 (アイルランド)	eu-west-1	emr-serverless.eu-west-1.amazonaws.com	HTTPS
欧州 (ロンドン)	eu-west-2	emr-serverless.eu-west-2.amazonaws.com	HTTPS
欧州 (ミラノ)	eu-south-1	emr-serverless.eu-south-1.amazonaws.com	HTTPS
欧州 (パリ)	eu-west-3	emr-serverless.eu-west-3.amazonaws.com	HTTPS
欧州 (スペイン)	eu-south-2	emr-serverless.eu-south-2.amazonaws.com	HTTPS
欧州 (ストックホルム)	eu-north-1	emr-serverless.eu-north-1.amazonaws.com	HTTPS
中東 (バーレーン)	me-south-1	emr-serverless.me-south-1.amazonaws.com	HTTPS

リージョン名	リージョン	エンドポイント	プロトコル
中東 (UAE)	me-central-1	emr-serverless.me-central-1.amazonaws.com	HTTPS
南米 (サンパウロ)	sa-east-1	emr-serverless.sa-east-1.amazonaws.com	HTTPS
AWS GovCloud （米国東部）	us-gov-east-1	emr-serverless.us-gov-east-1.amazonaws.com	HTTPS
AWS GovCloud （米国西部）	us-gov-west-1	emr-serverless.us-gov-west-1.amazonaws.com	HTTPS

Service Quotas

制限 とも呼ばれるサービスクォータ は、 が AWS アカウント 使用できるサービスリソースまたはオペレーションの最大数です。EMR Serverless は、サービスクォータ使用量メトリクスを 1 分ごとに収集し、AWS/Usage名前空間に発行します。

> **Note**
>
> 新しい AWS アカウントでは、時間の経過とともに増加する可能性のある初期の低いクォータがある場合があります。Amazon EMR Serverless は、各 内のアカウント使用状況を監視し AWS リージョン、使用状況に基づいてクォータを自動的に増加します。

次の表に、EMRServerless のサービスクォータを示します。詳細については、AWS のサービスクォータを参照してください。

名前	デフォルトの制限	引き上げ可能?	説明
アカウント vCPUs あたりの最大同時実行数	16	可能	現在の のアカウントで vCPUs 同時に実行できる の最大数 AWS リージョン。
アカウントあたりの最大キュージョブ数	2000	可能	現在の のアカウントのキューに入れられたジョブの最大数 AWS リージョン。

API 制限

以下は、 のリージョンごとのAPI制限について説明しています AWS アカウント。

リソース	デフォルトのクォータ
ListApplications	1 秒あたり 10 件のトランザクション。1 秒あたり 50 件のトランザクションのバースト。
CreateApplication	1 秒あたり 1 トランザクション。1 秒あたり 25 件のトランザクションのバースト。
DeleteApplication	1 秒あたり 1 トランザクション。1 秒あたり 25 件のトランザクションのバースト。
GetApplication	1 秒あたり 10 件のトランザクション。1 秒あたり 50 件のトランザクションのバースト。
UpdateApplication	1 秒あたり 1 トランザクション。1 秒あたり 25 件のトランザクションのバースト。
ListJobRuns	1 秒あたり 1 トランザクション。1 秒あたり 25 件のトランザクションのバースト。

リソース	デフォルトのクォータ
StartJobRun	1 秒あたり 1 トランザクション。1 秒あたり 25 件のトランザクションのバースト。
GetDashboardForJobRun	1 秒あたり 1 トランザクション。1 秒あたり 2 件のトランザクションのバースト。
CancelJobRun	1 秒あたり 1 トランザクション。1 秒あたり 25 件のトランザクションのバースト。
GetJobRun	1 秒あたり 10 件のトランザクション。1 秒あたり 50 件のトランザクションのバースト。
StartApplication	1 秒あたり 1 トランザクション。1 秒あたり 25 件のトランザクションのバースト。
StopApplication	1 秒あたり 1 トランザクション。1 秒あたり 25 件のトランザクションのバースト。

その他の考慮事項

次のリストには、EMRServerless に関するその他の考慮事項が含まれています。

- EMR サーバーレスは、以下でご利用いただけます。 AWS リージョン:
  - 米国東部(オハイオ)
  - 米国東部 (バージニア北部)
  - 米国西部 (北カリフォルニア)
  - 米国西部 (オレゴン)
  - アフリカ (ケープタウン)
  - アジアパシフィック (香港)
  - アジアパシフィック (ジャカルタ)
  - アジアパシフィック (ムンバイ)
  - アジアパシフィック (大阪)
  - アジアパシフィック (ソウル)
  - アジアパシフィック (シンガポール)
  - アジアパシフィック (シドニー)
  - アジアパシフィック (東京)
  - カナダ (中部)
  - 欧州 (フランクフルト)
  - 欧州 (アイルランド)
  - 欧州 (ロンドン)
  - 欧州 (ミラノ)
  - 欧州 (パリ)
  - 欧州 (スペイン)
  - 欧州 (ストックホルム)
  - 中東 (バーレーン)
  - 中東 (UAE)
  - 南米 (サンパウロ)
  - AWS GovCloud （米国東部）
  - AWS GovCloud （米国西部）

これらのリージョンに関連付けられているエンドポイントのリストについては、「」を参照してくださいサービスエンドポイント。

- ジョブ実行のデフォルトのタイムアウトは 12 時間です。この設定は、 startJobRunAPIまたはの executionTimeoutMinutesプロパティを使用して変更できます。 AWS SDK。ジョブの実行executionTimeoutMinutesがタイムアウトしないようにする場合は、 を 0 に設定できます。例えば、ストリーミングアプリケーションがある場合、ストリーミングジョブを継続的に実行できるように を executionTimeoutMinutes 0 に設定できます。
- の billedResourceUtilizationプロパティgetJobRunAPIには、 の集計 vCPU、メモリ、ストレージが表示されます。 AWS は、ジョブ実行に対して を請求しました。請求対象リソースには、ワーカーの最小使用量が 1 分、ワーカーあたり 20 GB を超える追加のストレージが含まれます。これらのリソースには、アイドル状態の事前初期化されたワーカーの使用は含まれません。
- VPC 接続しない場合、ジョブは の一部にアクセスできます。 AWS のサービス 同じ の エンドポイント AWS リージョン。 これらのサービスには、Amazon S3、 AWS Glue、Amazon CloudWatch Logs、 AWS KMS, AWS Security Token Service, Amazon DynamoDBおよび AWS Secrets Manager。 VPC 接続を有効にして、他の にアクセスできます。 AWS のサービス から AWS PrivateLink。ただし、これを行う必要はありません。外部サービスにアクセスするには、 を使用してアプリケーションを作成しますVPC。
- EMR サーバーレスは をサポートしていませんHDFS。ワーカーのローカルディスクは、ジョブ実行中に EMR Serverless がデータをシャッフルして処理するために使用する一時ストレージです。
- EMR サーバーレスは既存の をサポートしていませんemr-dynamodb-connector。

Amazon EMR Serverless リリースバージョン

Amazon EMRリリースは、ビッグデータエコシステムからの一連のオープンソースアプリケーションです。各リリースには、ジョブの実行時に Amazon EMR Serverless をデプロイして設定するために選択したビッグデータアプリケーション、コンポーネント、および機能が含まれています。

Amazon 6EMR.6.0 以降では、EMRサーバーレスをデプロイできます。このデプロイオプションは、以前の Amazon EMRリリースバージョンでは使用できません。ジョブを送信するときは、次のいずれかのサポートされているリリースを指定する必要があります。

トピック

EMR Serverless 7.2.0

次の表に、 で利用可能なアプリケーションバージョンを示します。EMR Serverless 7.2.0。

アプリケーション	Version
Apache Spark	3.5.1

アプリケーション	Version
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless 7.2.0 リリースノート

- Lake Formation with EMR Serverless – を使用できるようになりました AWS Lake Formation は、S3 によってバックアップされた Data Catalog テーブルにきめ細かなアクセスコントロールを適用します。この機能を使用すると、EMRServerless Spark ジョブ内の読み取りクエリのテーブル、行、列、セルレベルのアクセスコントロールを設定できます。詳細については、「the section called “の Lake Formation FGAC”」および「the section called “考慮事項”」を参照してください。

EMR Serverless 7.1.0

次の表に、 で利用可能なアプリケーションバージョンを示します。EMR Serverless 7.1.0。

アプリケーション	Version
Apache Spark	3.5.0
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless 7.0.0

次の表に、 で利用可能なアプリケーションバージョンを示します。EMR Serverless 7.0.0。

アプリケーション	Version
Apache Spark	3.5.0
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless 6.15.0

次の表に、 で利用可能なアプリケーションバージョンを示します。EMR Serverless 6.15.0。

アプリケーション	Version
Apache Spark	3.4.1
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless 6.15.0 リリースノート

- TLS サポート — Amazon EMR Serverless リリース 6.15.0 TLS以降では、Spark ジョブ実行のワーカー間の相互暗号化通信を有効にできます。有効にすると、EMRサーバーレスはワーカーごとに一意の証明書を自動的に生成します。この証明書は、ワーカーがTLSハンドシェイク中に相互に認証し、データを安全に処理するための暗号化されたチャネルを確立するために使用するジョブ実行でプロビジョニングされます。相互暗号化の詳細については、TLS「ワーカー間の暗号化」を参照してください。

EMR Serverless 6.14.0

次の表に、 で利用可能なアプリケーションバージョンを示します。EMR Serverless 6.14.0。

アプリケーション	Version
Apache Spark	3.4.1
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless 6.13.0

次の表に、 で利用可能なアプリケーションバージョンを示します。EMR Serverless 6.13.0。

アプリケーション	Version
Apache Spark	3.4.1
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless 6.12.0

次の表に、 で利用可能なアプリケーションバージョンを示します。EMR Serverless 6.12.0。

アプリケーション	Version
Apache Spark	3.4.0
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless 6.11.0

次の表に、 で利用可能なアプリケーションバージョンを示します。EMR Serverless 6.11.0。

アプリケーション	Version
Apache Spark	3.3.2
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless 6.11.0 リリースノート

- 他のアカウントの S3 リソースにアクセスする - リリース 6.11.0 以降では、異なる の Amazon S3 バケットにアクセスするときに引き受ける複数のIAMロールを設定できます。 AWS EMR Serverless の アカウント。

EMR Serverless 6.10.0

次の表に、 で利用可能なアプリケーションバージョンを示します。EMR Serverless 6.10.0。

アプリケーション	Version
Apache Spark	3.3.1
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless 6.10.0 リリースノート

- リリース 6.10.0 以降のEMRサーバーレスアプリケーションの場合、`spark.dynamicAllocation.maxExecutors`プロパティのデフォルト値は ですinfinity。以前のリリースはデフォルトで です100。詳細については、「Spark ジョブのプロパティ」を参照してください。

EMR Serverless 6.9.0

次の表に、 で利用可能なアプリケーションバージョンを示します。EMR Serverless 6.9.0。

アプリケーション	Version
Apache Spark	3.3.0
Apache Hive	3.1.3
Apache Tez	0.10.2

EMR Serverless 6.9.0 リリースノート

- Apache Spark 用の Amazon Redshift 統合は、Amazon EMRリリース 6.9.0 以降に含まれています。以前はオープンソースツールであったこのネイティブインテグレーションは Spark コネクタと呼ばれるもので、これを使用して Apache Spark アプリケーションを構築することで、Amazon Redshift と Amazon Redshift Serverless 内のデータを読み書きできます。詳細については、

「Amazon EMR Serverless での Apache Spark の Amazon Redshift 統合の使用」を参照してください。

- EMR Serverless リリース 6.9.0 で のサポートが追加されました AWS Graviton2 (arm64) アーキテクチャ。`create-application` および の `architecture`パラメータを使用して `update-application` APIs arm64 アーキテクチャを選択できます。詳細については、「Amazon EMR Serverless アーキテクチャオプション」を参照してください。
- EMR Serverless Spark および Hive アプリケーションから直接 Amazon DynamoDB テーブルをエクスポート、インポート、クエリ、結合できるようになりました。詳細については、「Amazon EMR Serverless を使用した DynamoDB への接続」を参照してください。

既知の問題

- Amazon Redshift integration for Apache Spark を使用している場合に、time、timetz、timestamp、timestamptz のいずれかにマイクロ秒の精度を Parquet 形式で設定していると、コネクタがその時間値を最も近いミリ秒値に四捨五入します。回避策として、テキストアンロード形式 `unload_s3_format` パラメータを使用してください。

EMR Serverless 6.8.0

次の表に、 で利用可能なアプリケーションバージョンを示します。EMR Serverless 6.8.0。

アプリケーション	Version
Apache Spark	3.3.0
Apache Hive	3.1.3
Apache Tez	0.9.2

EMR Serverless 6.7.0

次の表に、 で利用可能なアプリケーションバージョンを示します。EMR Serverless 6.7.0。

アプリケーション	Version
Apache Spark	3.2.1

アプリケーション	Version
Apache Hive	3.1.3
Apache Tez	0.9.2

エンジン固有の変更、機能強化、解決された問題

次の表に、エンジン固有の新しい機能を示します。

変更	説明
機能	Tez スケジューラで、コンテナのプリエンプトではなく Tez タスクのプリエンプションがサポートされるようになりました

EMR Serverless 6.6.0

次の表に、 で利用可能なアプリケーションバージョンを示します。EMR Serverless 6.6.0。

アプリケーション	Version
Apache Spark	3.2.0
Apache Hive	3.1.2
Apache Tez	0.9.2

EMR サーバーレスの初期リリースノート

- EMR Serverless は Spark 設定分類 をサポートしていますspark-defaults。この分類は Spark の spark-defaults.conf XML ファイルの値を変更します。設定分類を使用すると、アプリケーションをカスタマイズできます。詳細については、「アプリケーションの設定」を参照してください。
- EMR Serverless は、Hive 設定分類 hive-site、tez-site、emrfs-site、および をサポートしていますcore-site。この分類では、Hive の hive-site.xml ファイル、Tez の tez-

site.xml ファイル、Amazon EMRの EMRFS設定、または Hadoop の core-site.xml ファイルの値をそれぞれ変更できます。設定分類を使用すると、アプリケーションをカスタマイズできます。詳細については、「アプリケーションの設定」を参照してください。

エンジン固有の変更、機能強化、解決された問題

- 次の表に、Hive と Tez のバックポートを示します。

  Hive と Tez の変更点

変更	説明
バックポート	TEZ-4430: tez.task.launch.cmd-optsプロパティの問題を修正しました
バックポート	HIVE-25971: キャッシュされたスレッドプールが開いていることによる Tez タスクのシャットダウン遅延を修正しました

ドキュメント履歴

次の表は、EMRServerless の前回のリリース以降のドキュメントの重要な変更点を示しています。このドキュメントの更新の詳細については、 RSSフィードをサブスクライブしてください。

変更	説明	日付
新規リリース	EMR Serverless 7.2.0	2024 年 7 月 25 日
新規リリース	EMR Serverless 7.1.0	2024 年 4 月 17 日
既存のポリシーを更新します。	新しい `SidCloudWatchPolicyStatement` と `EC2PolicyStatement` を A mazonEMRServerless ServiceRolePolicy ポリシーに追加しました。	2024 年 1 月 25 日
新規リリース	EMR Serverless 7.0.0	2023 年 12 月 29 日
新規リリース	EMR Serverless 6.15.0	2023 年 11 月 17 日
新機能	EMR Serverless (6.11 以降) とは異なるアカウントの Amazon S3 バケットにアクセスするときに引き受ける複数のIAMロールを設定する	2023 年 10 月 18 日
新規リリース	EMR Serverless 6.14.0	2023 年 10 月 17 日
新機能	EMR Serverless のデフォルトのアプリケーション設定	2023 年 9 月 25 日
デフォルトの Hive プロパティの更新	、`hive.driver.disk` 、`hive.tez.disk.size` 、`hive.tez.auto.reducer.parallelism` および `tez.grouping.min-size` Hive ジョ	2023 年 9 月 12 日

	ブプロパティ のデフォルト値を更新しました。	
新規リリース	EMR Serverless 6.13.0	2023 年 9 月 11 日
新規リリース	EMR Serverless 6.12.0	2023 年 7 月 21 日
新規リリース	EMR Serverless 6.11.0	2023 年 6 月 8 日
サービスにリンクされたロールポリシーの更新	"AWS/Usage" 名前空間でアカウントレベルの使用状況を公開するようにAmazonEMRServerlessServiceRolePolicy SLRロールを更新しました。	2023 年 4 月 20 日
EMR Serverless 一般提供 (GA)	これは EMR Serverless の最初のパブリックリリースです。	2022 年 6 月 1 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。